T.C
KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR/YAZILIM MÜHENDİSLİĞİ

PROJE KONUSU: VERİ TABANI YÖNETİM SİSTEMLERİ PROJESİ

ÖĞRENCİ ADI: Aybüke Türidi Elif Beyza BEYAZ
ÖĞRENCİ NUMARASI: 220502005 220502033
Github Linkleri
Aybüke Türidi : https://github.com/220502005
Elif Beyza Beyaz : https://github.com/beyzabeyaz0

DERS SORUMLUSU:
PROF. DR./DR. ÖĞR. ÜYESİ Nevcihan Duru
TARİH: 05.05.2024

1 GİRİŞ

1.1 Projenin amacı

Bu proje, Gezgin Gemi Şirketi olarak verilen bir şirketin seferlerini ve gemilerini yönetmek için bir veri tabanı yönetim sistemi oluşturmayı amaçlamaktadır. Bu şirkete hizmet veren çeşitli tipte gemiler vardır: yolcu gemileri, petrol tankerleri, konteyner gemileri vb. Bu veri tabanında her gemi için gemi bilgileri seri numarası, adı, ağırlığı, yapım yılı, sefer bilgileri ID, yola çıkış tarihi, dönüş tarihi, yola çıkış limanı, kaptan ve mürettebat üyeleri ID, adı, soyadı, adres, vatandaşlık, doğum tarihi, işe giriş tarihi, lisans/meslek ve liman bilgileri liman adı, ülke, nüfus, pasaport gerekli, demirleme ücreti bulunur. Anlamına gelmek. Bu veri tabanı, bu bilgilere erişimi mümkün olan bir yazılımın oluşturulmaya çalışıldığı yerdir. Bu yazılım, gemi, sefer, kaptan, mürettebat ve liman bilgileri eklemek, düzenlemek, silmek ve sorgulamak için bir arayüz sağlamalıdır. Veriler SQL serverda tutulur ve bu sebeple nesneler aracılığıyla veri akımı gerçekleşir.

Her bir varlık için (gemi, sefer, kaptan, mürettebat, liman) sınıflar oluşturulacak ve bu sınıflar üzerinden ilgili işlemler gerçekleştirilecektir. Veri tabanında da her bir varlık için uygun tablolar oluşturularak verilerin doğru bir şekilde tutulması sağlanacaktır. Tüm veri tabanı işlemleri yazılım tarafından otomatik olarak yapılır. Bu projenin amacı, Gezgin Gemi Şirketi'nin operasyonlarını artırmak, gemi ve sefer yönetimini kolaylaştırmak ve limanlarla ilişkisini daha iyi yönetmektir. Ayrıca, veri tabanı yönetim sistemi sağlanarak verilerin güvenliği ve doğruluğu ile şirketin karar alma süreçlerinde rol oynamayı amaçlamaktadır.

2 GEREKSİNİM ANALİZİ

2.1 Arayüz gereksinimleri

• Yönetim sistemleri:

Her bir veri tabanı için bunu form üzerinden doldurmaya yarayacak bir yönetim sistemi oluşturulmalıdır.

Gemi Yönetimi Ekranı, gemi verilerini görüntülemek, eklemek, düzenlemek ve silmek için tasarlanmıştır. Bu ekrandan ayrıca, kullanıcıların filtre yapabilmesi ve gemi detaylarını düzenleme imkânı sunulmalıdır. Sefer Yönetim Ekranı, sefer verilerini görüntülemek ve sefer eklemek, düzenlemek ve silmek için tasarlanmıştır. Sefer tarihlerine göre kullanıcının filtre yapabilmesi ve sefer detaylarını düzenleme eklenebilmelidir.

Kaptan ve Mürettebat Yönetimi Ekranı, kullanıcılar kaptan ve mürettebat verilerini görüntülemek, eklemek, düzenlemek ve silmek için bu ekrandan faydalanmalıdır. Ek olarak, kullanıcıların kaptan lisanslarını görmeleri ve mürettebatın görev düzenlemeleri yapabilecektir. Liman Yönetimi Ekranı, liman verilerini görüntülemek, liman eklemek, düzenlemek ve silmek için tasarlanmıştır. Kullanıcıların ayrıca, liman özelliklerini görmesi veya liman demirleme ücretleri düzenlemeleri de yapılabilir.

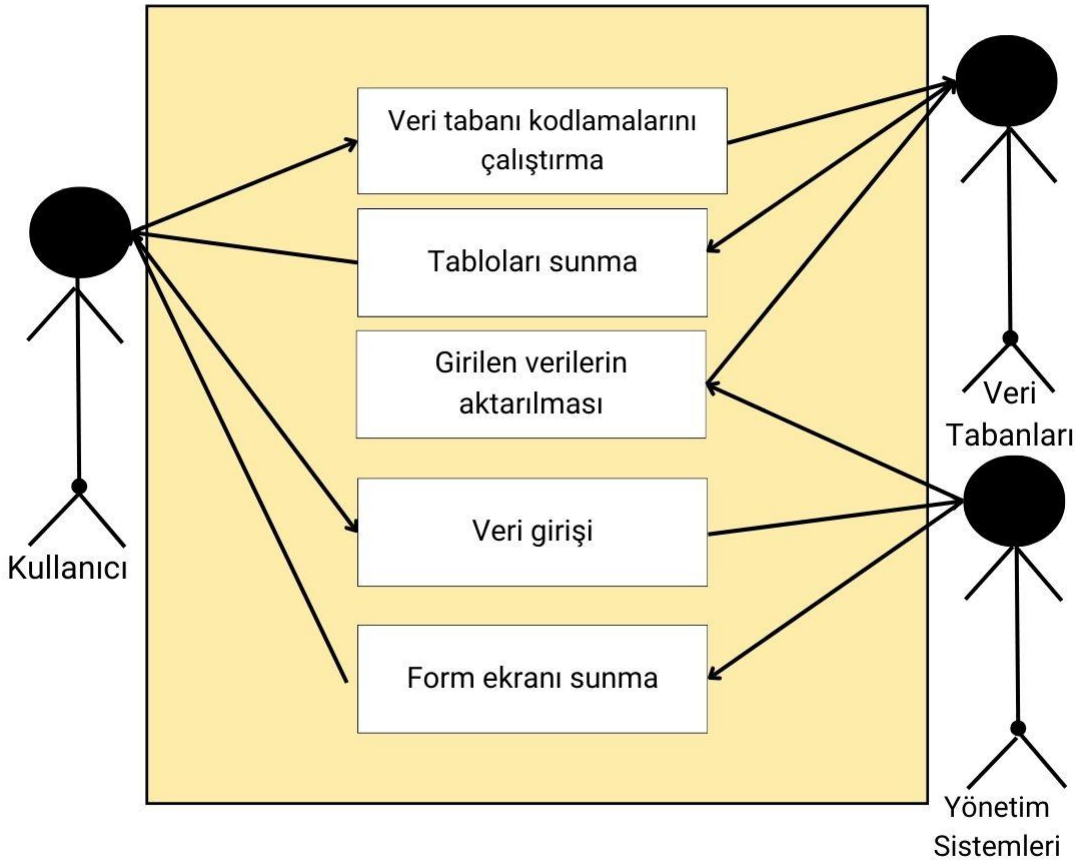
• İlk ekran
Kodlamayı çalıştırdığımızda çıkan ilk ekran terminal olmaktadır. Terminal üzerinden kullanıcı girdisi ile hangi veri tabanıyla işlem yapmak istediğimizi seçeriz ve her koşulda kodu çalıştırdığımız an tüm tablolar arkada çalışmış olur.

2.2 Fonksiyonel gereksinimler

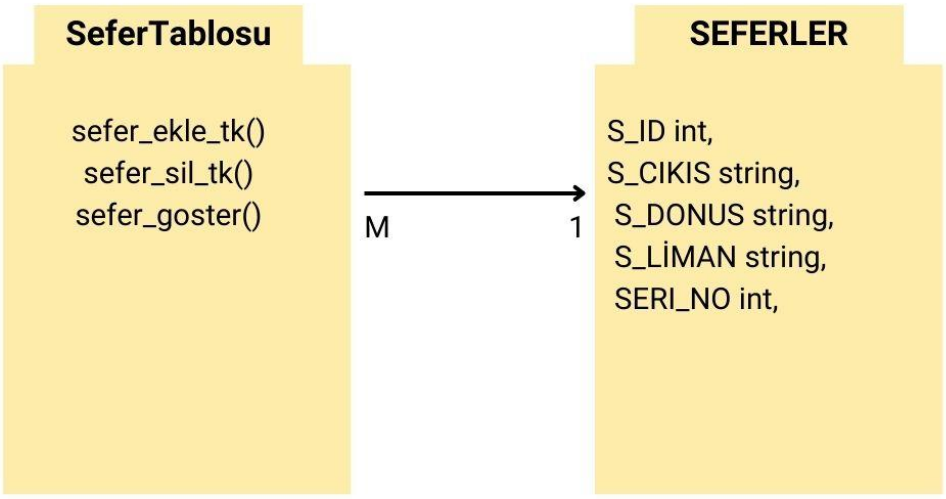
- Veri tabanındaki verilere erişim sağlama, veri ekleme çıkarma gibi işlemleri yapabilme
- Form ekranı oluşturma ve bu form üzerinden veri tabanına veri girişinde bulunma
- Veri tabanı işlemlerinin hatasız ve güvenilir bir şekilde gerçekleştirilmesini sağlama
- 4 adet sistem bulunur. Bunlar için:
  Gemi yönetimi işlevselliği, yeni gemi ekleyebilme, mevcut gemileri görüntüleme, düzenleme ve silme, gemi türüne göre filtreleme yapabilme ve gemi detaylarını görüntüleme yeteneklerini içerir.
  Sefer yönetimi, yeni sefer ekleyebilme, mevcut seferleri görüntüleme, düzenleme ve silme, belirli bir tarihe göre sefer oluşturma ve sefer detaylarını görüntüleme işlevlerini içerir.
  Kaptan ve mürettebat yönetimi, yeni kaptan veya mürettebat ekleyebilme, mevcut kaptan ve mürettebat bilgilerini görüntüleme, düzenleme ve silme, kaptanların lisans bilgilerini görüntüleme ve mürettebatın görevlerini düzenleme yeteneklerini kapsar.
  Liman yönetimi, yeni liman ekleyebilme, mevcut limanları görüntüleme, düzenleme ve silme, liman özelliklerini görüntüleme ve limanları ülke veya nüfusa göre filtreleme işlevlerini içerir.

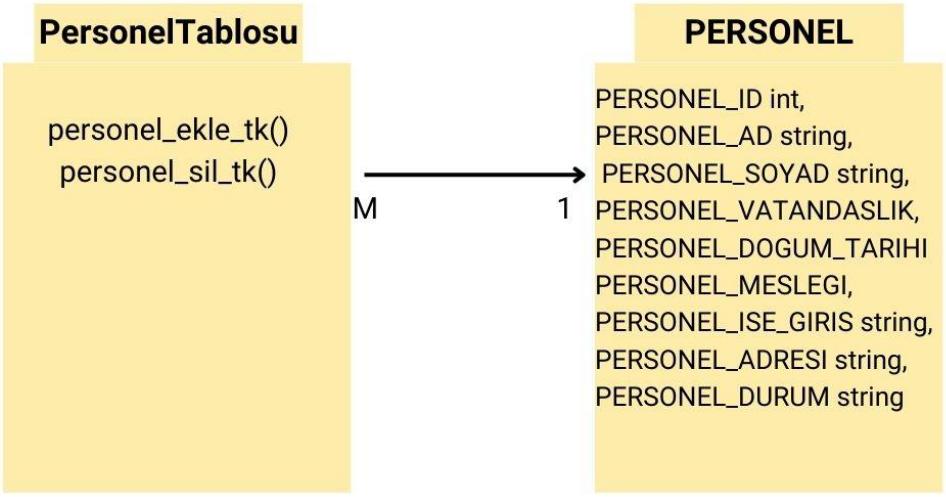
3 TASARIM
3.1 Mimari tasarım

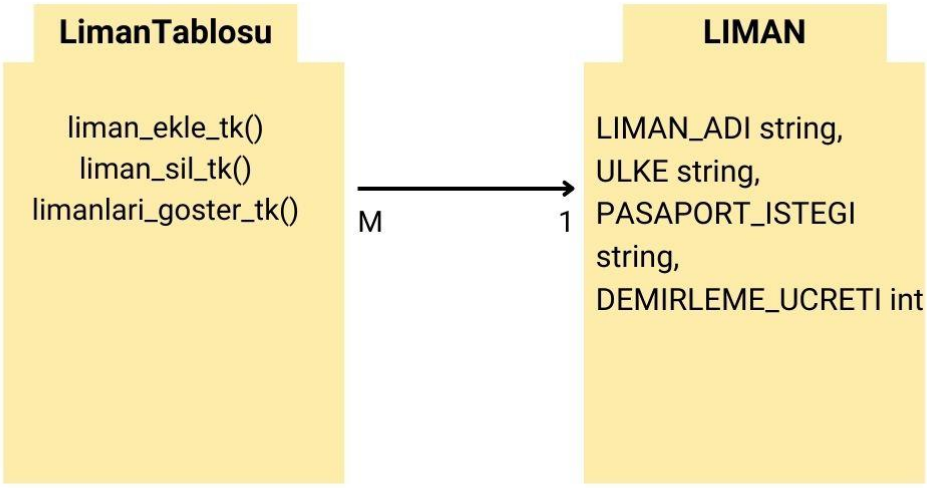
USE CASE DİYAGRAMI:

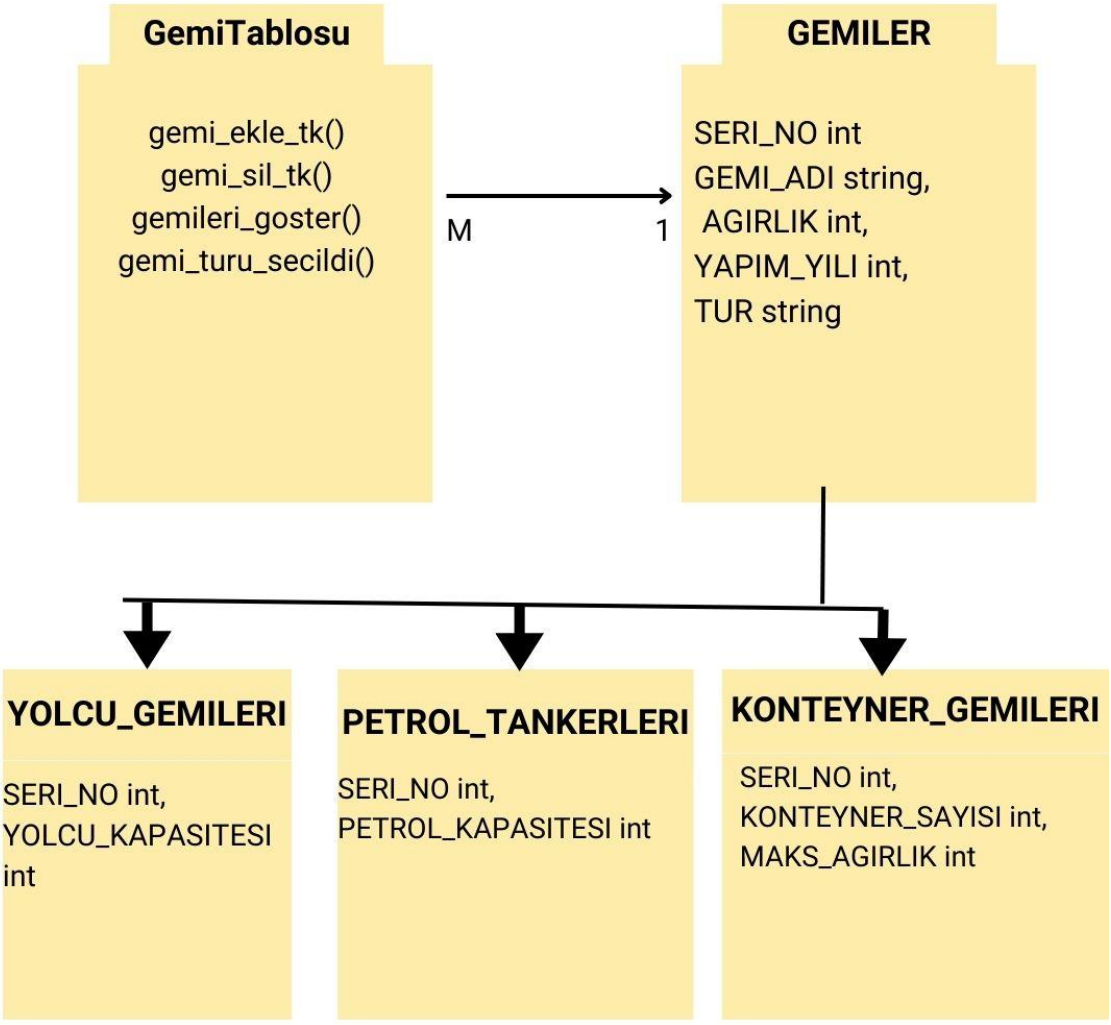


SINIF DİYAGRAMLARI









3.2 Veri tabanı Tasarımı

Bu projenin temel amacı veri tabanları oluşturmaktır. Kod girişli verileri veri tabanına ekleme işlemlerini yaparız. 4 adet veri tabanı bulunur:

A) Gemiler

Bu tablonun verileri ve oluşturulması(kodlama):

Seri numarası
Adı
Ağırlık
Yapım yılı
Tür (yolcu gemisi, petrol tankerleri, konteyner gemileri)
Özel özellikler (yolcu kapasitesi, petrol kapasitesi, konteyner kapasitesi, maksimum ağırlık)

```
# GEMILER tablosu
imlec_gemiler.execute('''
    CREATE TABLE IF NOT EXISTS GEMILER (
        SERI_NO INTEGER PRIMARY KEY,
        GEMI_ADI TEXT,
        AGIRLIK REAL,
        YAPIM_YILI INTEGER,
        TUR TEXT
    )
''')

# Yolcu gemileri için
imlec_gemiler.execute('''
    CREATE TABLE IF NOT EXISTS YOLCU_GEMILERI (
        SERI_NO INTEGER,
        YOLCU_KAPASITESI INTEGER,
        FOREIGN KEY(SERI_NO) REFERENCES GEMILER(SERI_NO)
    )
''')

# Petrol gemileri için
imlec_gemiler.execute('''
    CREATE TABLE IF NOT EXISTS PETROL_TANKERLERI (
        SERI_NO INTEGER,
        PETROL_KAPASITESI REAL,
        FOREIGN KEY(SERI_NO) REFERENCES GEMILER(SERI_NO)
    )
''')

  # Konteyner gemileri için
  imlec_gemiler.execute('''
      CREATE TABLE IF NOT EXISTS KONTEYNER_GEMILERI (
          SERI_NO INTEGER,
          KONTEYNER_SAYISI INTEGER,
          MAKS_AGIRLIK REAL,
          FOREIGN KEY(SERI_NO) REFERENCES GEMILER(SERI_NO)
      )
  ''')
```

B) Seferler

Bu tablonun verileri ve oluşturulması(kodlama):

ID
Gemi (referans)
Yola çıkış tarihi
Dönüş tarihi
Yola çıkış limanı (referans)

```
# SEFERLER tablosu
cursor_seferler.execute('''
        CREATE TABLE IF NOT EXISTS SEFERLER (
            S_ID INTEGER PRIMARY KEY,
            S_CIKIS TEXT,
            S_DONUS TEXT,
            S_LİMAN TEXT,
            SERI_NO INTEGER,
            FOREIGN KEY(SERI_NO) REFERENCES GEMILER(SERI_NO)
        )
    ''')
```

C) Personel

Bu tablonun verileri ve oluşturulması(kodlama):

ID
Ad
Soyad
Adres
Vatandaşlık
Doğum tarihi
İşe giriş tarihi
Lisanslar veya Görevler

```
# PERSONEL tablosu
imlec_personel.execute('''
        CREATE TABLE IF NOT EXISTS PERSONEL (
            PERSONEL_ID INTEGER PRIMARY KEY,
            PERSONEL_AD TEXT,
            PERSONEL_SOYAD TEXT,
            PERSONEL_VATANDASLIK TEXT,
            PERSONEL_DOGUM_TARIHI TEXT,
            PERSONEL_MESLEGI, PERSONEL_ISE_GIRIS TEXT,
            PERSONEL_ADRESI TEXT,
            PERSONEL_DURUM TEXT
        )
    ''')
```

D) Liman

Bu tablonun verileri ve oluşturulması(kodlama):

Adı
Ülkesi
Nüfusu
Pasaport gereksinimi
Demirleme ücreti

```
# LİMAN tablosu
imlec_liman.execute('''
        CREATE TABLE IF NOT EXISTS LIMAN (
        LIMAN_ADI TEXT,
        ULKE TEXT,
        PASAPORT_ISTEGI TEXT CHECK(PASAPORT_ISTEGI IN ('E', 'H')), -- E: Evet, H: Hayır,
        DEMIRLEME_UCRETI REAL,
        PRIMARY KEY (LIMAN_ADI, ULKE)
        )
    ''')
```

3.3 Kullanıcı Arayüzü Tasarımı

Kullanıcı 2 türlü giriş yapabilmektedir: İlki terminal üzerinden hangi veri tabanına ulaşmak istediği sorgusudur.

Bu sorguyu basitçe input ve if-else yapısıyla sağlarız.

```
secim = input("HANGİ TABLO İÇİN VERİ İŞLEMİ YAPMAK İSTERSİNİZ?\n A: GEMİ  B: SEFER  C: PERSONEL  D: LİMAN\n Seçiniz:")

if secim.upper() == "A":
    uygulama = GemiTablosu()
    uygulama.mainloop()
elif secim.upper() == "B":
    uygulama = SeferTablosu()
    uygulama.mainloop()
elif secim.upper() == "C":
    uygulama = PersonelTablosu()
    uygulama.mainloop()
elif secim.upper() == "D":
    uygulama = LimanTablosu()
    uygulama.mainloop()
else:
    print("şıklardan seçim yapınız")
```

Diğer kullanıcı giriş türü ise form ekranıdır. Tkinter kütüphanesiyle oluşturulmuş ve veri tabanıyla bağlantılı olan form yapısı kullanıcının ekranına pencere sunar ve bu pencereden veri girişi sağlayabilmektedir.

Her veri tabanı için ayrı bir form ekranı sunulmaktadır.

Gemi için:

```
class GemiTablosu(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Gemi Yönetim Sistemi")

        self.label_seri_no = tk.Label(self, text="Seri No:")
        self.entry_seri_no = tk.Entry(self)

        self.label_gemi_adi = tk.Label(self, text="Gemi Adı:")
        self.entry_gemi_adi = tk.Entry(self)

        self.label_agirlik = tk.Label(self, text="Ağırlık:")
        self.entry_agirlik = tk.Entry(self)

        self.label_yapim_yili = tk.Label(self, text="Yapım Yılı:")
        self.entry_yapim_yili = tk.Entry(self)

        self.label_tur = tk.Label(self, text="Gemi Türü:")

        self.label_silinecek_seri_no = tk.Label(self, text="Silinecek Seri No:")
        self.label_silinecek_seri_no.grid(row=6, column=0)

        self.entry_silinecek_seri_no = tk.Entry(self)
        self.entry_silinecek_seri_no.grid(row=6, column=1)

        self.sil_button = tk.Button(self, text="Gemi Sil", command=self.gemi_sil)
        self.sil_button.grid(row=7, column=0, columnspan=2)
```

```
self.liste_button = tk.Button(self, text="Gemi Listesi", command=self.gemileri_goster)
self.liste_button.grid(row=8, column=0, columnspan=2)

self.gemi_turu_var = tk.StringVar(value="YOLCU")  # varsayılan tür
self.radio_yolcu = tk.Radiobutton(self, text="Yolcu Gemisi", variable=self.gemi_turu_var, value="YOLCU",
                                  command=self.gemi_turu_secildi)
self.radio_petrol = tk.Radiobutton(self, text="Petrol Tankeri", variable=self.gemi_turu_var,
                                   value="PETROL_TANKERI", command=self.gemi_turu_secildi)
self.radio_konteyner = tk.Radiobutton(self, text="Konteyner Gemisi", variable=self.gemi_turu_var,
                                      value="KONTEYNER", command=self.gemi_turu_secildi)

self.label_yolcu_kapasitesi = tk.Label(self, text="Yolcu Kapasitesi:")
self.entry_yolcu_kapasitesi = tk.Entry(self)

self.label_petrol_kapasitesi = tk.Label(self, text="Petrol Kapasitesi (litre):")
self.entry_petrol_kapasitesi = tk.Entry(self)

self.label_konteyner_sayisi = tk.Label(self, text="Konteyner Sayısı:")
self.entry_konteyner_sayisi = tk.Entry(self)

self.label_maks_agirlik = tk.Label(self, text="Maksimum Ağırlık:")
self.entry_maks_agirlik = tk.Entry(self)

self.ekle_button = tk.Button(self, text="Gemi Ekle", command=self.gemi_ekle)

self.label_seri_no.grid(row=0, column=0, padx=5, pady=5, sticky='w')
self.entry_seri_no.grid(row=0, column=1, padx=5, pady=5, sticky='w')
```

```
self.label_gemi_adi.grid(row=1, column=0, padx=5, pady=5, sticky='w')
self.entry_gemi_adi.grid(row=1, column=1, padx=5, pady=5, sticky='w')

self.label_agirlik.grid(row=2, column=0, padx=5, pady=5, sticky='w')
self.entry_agirlik.grid(row=2, column=1, padx=5, pady=5, sticky='w')

self.label_yapim_yili.grid(row=3, column=0, padx=5, pady=5, sticky='w')
self.entry_yapim_yili.grid(row=3, column=1, padx=5, pady=5, sticky='w')

self.label_tur.grid(row=4, column=0, padx=5, pady=5, sticky='w')
self.radio_yolcu.grid(row=4, column=1, padx=5, pady=5, sticky='w')
self.radio_petrol.grid(row=4, column=2, padx=5, pady=5, sticky='w')
self.radio_konteyner.grid(row=4, column=3, padx=5, pady=5, sticky='w')

self.label_yolcu_kapasitesi.grid(row=5, column=0, padx=5, pady=5, sticky='w')
self.entry_yolcu_kapasitesi.grid(row=5, column=1, padx=5, pady=5, sticky='w')

self.label_petrol_kapasitesi.grid(row=6, column=0, padx=5, pady=5, sticky='w')
self.entry_petrol_kapasitesi.grid(row=6, column=1, padx=5, pady=5, sticky='w')

self.label_konteyner_sayisi.grid(row=7, column=0, padx=5, pady=5, sticky='w')
self.entry_konteyner_sayisi.grid(row=7, column=1, padx=5, pady=5, sticky='w')

self.label_maks_agirlik.grid(row=8, column=0, padx=5, pady=5, sticky='w')
self.entry_maks_agirlik.grid(row=8, column=1, padx=5, pady=5, sticky='w')

self.ekle_button.grid(row=9, column=0, columnspan=2, padx=5, pady=10, sticky='we')
self.label_silinecek_seri_no.grid(row=10, column=0, padx=5, pady=5, sticky='w')
self.entry_silinecek_seri_no.grid(row=10, column=1, padx=5, pady=5, sticky='w')
self.sil_button.grid(row=11, column=0, columnspan=2, padx=5, pady=10, sticky='we')
self.liste_button.grid(row=12, column=0, columnspan=2, padx=5, pady=10, sticky='we')

self.gemi_turu_secildi()

self.ekle_button.grid(row=9, column=0, columnspan=2)
```

Sefer için:

```
class SeferTablosu(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Sefer Yönetim Sistemi")
        self.geometry("500x500")

        self.entry_s_ID = tk.Entry(self)
        self.entry_s_CIKIS = tk.Entry(self)
        self.entry_s_DONUS = tk.Entry(self)
        self.entry_s_LIMAN = tk.Entry(self)
        self.entry_seri_no = tk.Entry(self)

        self.label_sefer_id = tk.Label(self, text="Sefer ID:")
        self.label_sefer_id.grid(row=0, column=0, padx=5, pady=5, sticky='w')
        self.entry_s_ID.grid(row=0, column=1, padx=5, pady=5, sticky='w')

        self.label_sefer_cikis = tk.Label(self, text="Çıkış Tarihi:")
        self.label_sefer_cikis.grid(row=1, column=0, padx=5, pady=5, sticky='w')
        self.entry_s_CIKIS.grid(row=1, column=1, padx=5, pady=5, sticky='w')

        self.label_sefer_donus = tk.Label(self, text="Dönüş Tarihi:")
        self.label_sefer_donus.grid(row=2, column=0, padx=5, pady=5, sticky='w')
        self.entry_s_DONUS.grid(row=2, column=1, padx=5, pady=5, sticky='w')

        self.label_sefer_liman = tk.Label(self, text="Çıkış Limanı:")
        self.label_sefer_liman.grid(row=3, column=0, padx=5, pady=5, sticky='w')
        self.entry_s_LIMAN.grid(row=3, column=1, padx=5, pady=5, sticky='w')
```

```
self.label_seri_no = tk.Label(self, text="Gemi Seri No:")
self.label_seri_no.grid(row=4, column=0, padx=5, pady=5, sticky='w')
self.entry_seri_no.grid(row=4, column=1, padx=5, pady=5, sticky='w')

self.ekle_button = tk.Button(self, text="Sefer Ekle", command=self.sefer_ekle_tk)
self.ekle_button.grid(row=5, column=0, columnspan=2, padx=5, pady=10, sticky='we')

self.label_silinecek_sefer_id = tk.Label(self, text="Silinecek Sefer ID:")
self.label_silinecek_sefer_id.grid(row=6, column=0, padx=5, pady=5, sticky='w')
self.entry_silinecek_sefer_id = tk.Entry(self)
self.entry_silinecek_sefer_id.grid(row=6, column=1, padx=5, pady=5, sticky='w')

self.sil_button = tk.Button(self, text="Sefer Sil", command=self.sefer_sil_tk)
self.sil_button.grid(row=7, column=0, columnspan=2, padx=5, pady=10, sticky='we')

self.liste_button = tk.Button(self, text="Sefer Listesi", command=self.sefer_goster)
self.liste_button.grid(row=8, column=0, columnspan=2, padx=5, pady=10, sticky='we')
```

Personel için:

```
self.label_personel_durum = tk.Label(self, text="Lisans veya Görev:")
self.entry_personel_durum = tk.Entry(self)

self.ekle_button = tk.Button(self, text="Personel Ekle", command=self.personel_ekle_tk)
self.ekle_button.grid(row=9, column=0, columnspan=2, padx=5, pady=10, sticky='we')

self.label_silinecek_personel_id = tk.Label(self, text="Silinecek Personel ID:")
self.entry_silinecek_personel_id = tk.Entry(self)
self.sil_button = tk.Button(self, text="Personel Sil", command=self.personel_sil_tk)

self.liste_button = tk.Button(self, text="Personelleri Göster", command=self.personelleri_goster)

self.label_personel_id.grid(row=0, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_id.grid(row=0, column=1, padx=5, pady=5, sticky='w')

self.label_personel_ad.grid(row=1, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_ad.grid(row=1, column=1, padx=5, pady=5, sticky='w')

self.label_personel_soyad.grid(row=2, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_soyad.grid(row=2, column=1, padx=5, pady=5, sticky='w')

self.label_personel_vatandaslik.grid(row=3, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_vatandaslik.grid(row=3, column=1, padx=5, pady=5, sticky='w')

self.label_personel_dogum_tarihi.grid(row=4, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_dogum_tarihi.grid(row=4, column=1, padx=5, pady=5, sticky='w')

self.label_personel_meslek.grid(row=5, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_meslek.grid(row=5, column=1, padx=5, pady=5, sticky='w')
```

```
class PersonelTablosu(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Personel Yönetim Sistemi")
        self.geometry("500x500")

        self.label_personel_id = tk.Label(self, text="Personel ID:")
        self.entry_personel_id = tk.Entry(self)

        self.label_personel_ad = tk.Label(self, text="Ad:")
        self.entry_personel_ad = tk.Entry(self)

        self.label_personel_soyad = tk.Label(self, text="Soyad:")
        self.entry_personel_soyad = tk.Entry(self)

        self.label_personel_vatandaslik = tk.Label(self, text="Vatandaşlık:")
        self.entry_personel_vatandaslik = tk.Entry(self)

        self.label_personel_dogum_tarihi = tk.Label(self, text="Doğum Tarihi:")
        self.entry_personel_dogum_tarihi = tk.Entry(self)

        self.label_personel_meslek = tk.Label(self, text="Meslek:")
        self.entry_personel_meslek = tk.Entry(self)

        self.label_personel_ise_giris = tk.Label(self, text="İşe Giriş Tarihi:")
        self.entry_personel_ise_giris = tk.Entry(self)

        self.label_personel_adresi = tk.Label(self, text="Adres:")
        self.entry_personel_adresi = tk.Entry(self)
```

```
self.label_personel_ise_giris.grid(row=6, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_ise_giris.grid(row=6, column=1, padx=5, pady=5, sticky='w')

self.label_personel_adresi.grid(row=7, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_adresi.grid(row=7, column=1, padx=5, pady=5, sticky='w')

self.label_personel_durum.grid(row=8, column=0, padx=5, pady=5, sticky='w')
self.entry_personel_durum.grid(row=8, column=1, padx=5, pady=5, sticky='w')

self.ekle_button.grid(row=9, column=0, columnspan=2, padx=5, pady=10, sticky='we')


self.label_silinecek_personel_id.grid(row=10, column=0, padx=5, pady=5, sticky='w')
self.entry_silinecek_personel_id.grid(row=10, column=1, padx=5, pady=5, sticky='w')
self.sil_button.grid(row=11, column=0, columnspan=2, padx=5, pady=10, sticky='we')

self.liste_button.grid(row=12, column=0, columnspan=2, padx=5, pady=10, sticky='we')
```

Liman için:

```
class LimanTablosu(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Liman Yönetim Sistemi")
        self.geometry("400x400")

        self.label_liman_adi = tk.Label(self, text="Liman Adı:")
        self.entry_liman_adi = tk.Entry(self)

        self.label_ulke = tk.Label(self, text="Ülke:")
        self.entry_ulke = tk.Entry(self)

        self.label_pasaport_istegi = tk.Label(self, text="Pasaport Gereksinimi (E/H):")
        self.entry_pasaport_istegi = tk.Entry(self)

        self.label_demirleme_ucreti = tk.Label(self, text="Demirleme Ücreti:")
        self.entry_demirleme_ucreti = tk.Entry(self)

        self.ekle_button = tk.Button(self, text="Liman Ekle", command=self.liman_ekle_tk)

        self.label_silinecek_liman_adi = tk.Label(self, text="Silinecek Liman Adı:")
        self.entry_silinecek_liman_adi = tk.Entry(self)

        self.label_silinecek_ulke = tk.Label(self, text="Silinecek Ülke:")
        self.entry_silinecek_ulke = tk.Entry(self)

        self.sil_button = tk.Button(self, text="Liman Sil", command=self.liman_sil_tk)

        self.liste_button = tk.Button(self, text="Liman Listesi", command=self.limanlari_goster_tk)
```

```
self.label_liman_adi.grid(row=0, column=0, padx=5, pady=5, sticky='w')
self.entry_liman_adi.grid(row=0, column=1, padx=5, pady=5, sticky='w')

self.label_ulke.grid(row=1, column=0, padx=5, pady=5, sticky='w')
self.entry_ulke.grid(row=1, column=1, padx=5, pady=5, sticky='w')

self.label_pasaport_istegi.grid(row=2, column=0, padx=5, pady=5, sticky='w')
self.entry_pasaport_istegi.grid(row=2, column=1, padx=5, pady=5, sticky='w')

self.label_demirleme_ucreti.grid(row=3, column=0, padx=5, pady=5, sticky='w')
self.entry_demirleme_ucreti.grid(row=3, column=1, padx=5, pady=5, sticky='w')

self.ekle_button.grid(row=4, column=0, columnspan=2, padx=5, pady=10, sticky='we')

self.label_silinecek_liman_adi.grid(row=5, column=0, padx=5, pady=5, sticky='w')
self.entry_silinecek_liman_adi.grid(row=5, column=1, padx=5, pady=5, sticky='w')

self.label_silinecek_ulke.grid(row=6, column=0, padx=5, pady=5, sticky='w')
self.entry_silinecek_ulke.grid(row=6, column=1, padx=5, pady=5, sticky='w')

self.sil_button.grid(row=7, column=0, columnspan=2, padx=5, pady=10, sticky='we')

self.liste_button.grid(row=8, column=0, columnspan=2, padx=5, pady=10, sticky='we')
```

3.4 Kullanılan Teknolojiler

Bu ödevde kullanılabilecek çeşitli teknolojiler bulunmaktadır:

Veri tabanı Yönetim Sistemi: Veri tabanı oluşturmak ve yönetmek için bir yönetim sistemi kullanmanız gerekecek. Bunu kullandığımız dile ve platforma uygun olarak belirleyebiliriz. Örneğin, MySQL, PostgreSQL, Microsoft SQL Server gibi birçok seçenek mevcuttur.

Sunucu Tarafı Programlama Dili: Veri tabanına erişim ve iş mantığını yönetmek için sunucu tarafı bir programlama dili kullanabilirsiniz. Biz Python dilini kullandık.

4 UYGULAMA

4.1 Kodlanan bileşenlerin açıklamaları

Veri tabanlarını ve kullanıcı arayüzü için yapılan kodlamaların genel açıklamaları 3.2 ve 3.3 maddesinde bulunmaktadır. Bu veri tabanlarının ve form arayüzünün belirli metotları aşağıdaki gibidir:

Gemi veri tabanı için metotlar:

Gemi satır sütunlara veri ekleme metodu: “gemi_ekle ( )”

Bu metod, gemi için benzersiz bir seri numarası, adı, ağırlığı, yapım yılı ve türü (“YOLCU”, “PETROL_TANKERI”, “KONTEYNER”) parametrelere göre gemi nesnesinin bilgilerini “GEMILER.db” adlı veritabanına ekler. Metod, bu temel bilgileri “GEMILER tablosuna ekler.
Ardından, geminin türüne bağlı olarak; eğer “YOLCU” ise ve yolcu kapasitesi belirtilmişse, bu veriyi “YOLCU_GEMILERI” tablosuna ekler, eğer "PETROL_TANKERI" ise ve petrol kapasitesi belirtilmişse, “PETROL_TANKERLERI” tablosuna ekler; eğer “KONTEYNER” ise ve konteyner sayısı ile maksimum ağırlık belirtilmişse, “KONTEYNER_GEMILERI” tablosuna ekleme yapar.
Son olarak, veritabanı değişikliklerini “baglanti_gemi2.commit()” ile kaydeder ve bağlantıyı “baglanti_gemi2.close()” ile kapatır.

```python
def gemi_ekle(seri_no, gemi_adi, agirlik, yapim_yili, tur, yolcu_kapasitesi=None, petrol_kapasitesi=None,
              konteyner_sayisi=None, maks_agirlik=None):
    baglanti_gemi2 = sqlite3.connect('GEMILER.db')
    imlec_gemi2 = baglanti_gemi2.cursor()
    imlec_gemi2.execute('''
        INSERT INTO GEMILER (SERI_NO, GEMI_ADI, AGIRLIK, YAPIM_YILI, TUR)
        VALUES (?, ?, ?, ?, ?)
    ''', (seri_no, gemi_adi, agirlik, yapim_yili, tur))

    if tur == "YOLCU" and yolcu_kapasitesi is not None:
        imlec_gemi2.execute('''
            INSERT INTO YOLCU_GEMILERI (SERI_NO, YOLCU_KAPASITESI)
            VALUES (?, ?)
        ''', (seri_no, yolcu_kapasitesi))
    elif tur == "PETROL_TANKERI" and petrol_kapasitesi is not None:
        imlec_gemi2.execute('''
            INSERT INTO PETROL_TANKERLERI (SERI_NO, PETROL_KAPASITESI)
            VALUES (?, ?)
        ''', (seri_no, petrol_kapasitesi))
    elif tur == "KONTEYNER" and konteyner_sayisi is not None and maks_agirlik is not None:
        imlec_gemi2.execute('''
            INSERT INTO KONTEYNER_GEMILERI (SERI_NO, KONTEYNER_SAYISI, MAKS_AGIRLIK)
            VALUES (?, ?, ?)
        ''', (seri_no, konteyner_sayisi, maks_agirlik))

    baglanti_gemi2.commit()
    baglanti_gemi2.close()
```

Gemi silme metodu: “ gemi_sil (seri_no)”

Bu metod , “GEMILER.db” adlı veritabanından belirli bir seri numarasına sahip gemiyi silmek için çalışır. “sqlite3.connect(“GEMILER.db”)” ile veritabanına bağlanır ve “imlec_gemi3” adında bir imleç (cursor) oluşturur. İmleç, “DELETE FROM GEMILER WHERE SERI_NO=?” komutuyla paremetre olarak gönderilen seri numarasındaki gemiyi siler.
Ardından, “baglanti_gemi3.commit()” ile bu değişiklikler kaydedilir.
“baglanti_gemi3.close()” ile veritabanı bağlantısı kapatılır.
Kısaca bu kod parçası, yani “gemi_sil ( ) metodu belirli bir seri numarasına sahip gemiyi veritabanından siler, yapılan değişiklikleri kaydeder ve bağlantıyı kapatarak işi tamamlar.

```python
def gemi_sil(seri_no):
    baglanti_gemi3 = sqlite3.connect('GEMILER.db')
    imlec_gemi3 = baglanti_gemi3.cursor()
    imlec_gemi3.execute('''
        DELETE FROM GEMILER WHERE SERI_NO=?
    ''', (seri_no,))
    baglanti_gemi3.commit()
    baglanti_gemi3.close()
```

Gemi alma metodu: “gemileri_al()”

Bu metod, “GEMILER.db” adlı veritabanından gemi kayıtlarını alır. Metod veritabanına bağlanır ve bir imleç oluşturur. İmleç (cursor) aracılığıyla, “GEMILER” tablosundaki tüm kayıtları seçmek için “SELECT * FROM GEMILER” komutu kullanır. fetchall() yöntemi “GEMILER.db” veritabanı sorgusunun tüm sonuçlarını bir kerede almak için kullanılır.

Sonuçlar “gemiler” adlı bir listeye kaydedilir. Alınan gemi kayıtlarını gemiler listesi halinde döndürür.

```
def gemileri_al():
    baglanti_gemi4 = sqlite3.connect('GEMILER.db')
    imlec_gemi4 = baglanti_gemi4.cursor()
    imlec_gemi4.execute('SELECT * FROM GEMILER')
    gemiler = imlec_gemi4.fetchall()
    baglanti_gemi4.close()
    return gemiler
```

Gemi türü belirleme metodu: “gemi_turu_bilgilerini_al(seri_no, tur)”

Bu metod, "GEMILER.db" veritabanından belirli bir seri numarasına sahip gemi hakkında türüne bağlı olarak bilgi almak için kullanılır.
Veritabanına bağlanılır (baglanti_gemi5 = sqlite3.connect(“GEMILER.db”))ve imleç (imlec_gemi5 = baglanti_gemi5.cursor())oluşturulur. Eğer gemi türü “YOLCU” ise, yolcu kapasitesi sorgulanır; “PETROL_TANKERI” ise petrol kapasitesi, “KONTEYNER” ise konteyner sayısı ve maksimum ağırlık bilgisi alınır. Sorgu yapıldıktan sonra, “imlec_gemi5.fetchone()” ile sorgudan ilk sonuç yani ilk satır alınır ve bağlantı kapatılır.

```
def gemi_turu_bilgilerini_al(seri_no, tur):
    baglanti_gemi5 = sqlite3.connect('GEMILER.db')
    imlec_gemi5 = baglanti_gemi5.cursor()
    if tur == "YOLCU":
        imlec_gemi5.execute('SELECT YOLCU_KAPASITESI FROM YOLCU_GEMILERI WHERE SERI_NO = ?', (seri_no,))
    elif tur == "PETROL_TANKERI":
        imlec_gemi5.execute('SELECT PETROL_KAPASITESI FROM PETROL_TANKERLERI WHERE SERI_NO = ?', (seri_no,))
    elif tur == "KONTEYNER":
        imlec_gemi5.execute('SELECT KONTEYNER_SAYISI, MAKS_AGIRLIK FROM KONTEYNER_GEMILERI WHERE SERI_NO = ?', (seri_no,))

    bilgi = imlec_gemi5.fetchone()
    baglanti_gemi5.close()
    return bilgi
```

Gemi form ekranı için metotlar:

Formdan gelen bilgilerle veri tabanına gemi ekleme metodu: “gemi_ekle_tk()”

“Tkinter” kütüphanesi ile oluşturulan ekranda kullanıcının ilgili alanlara girdiği değerler, “self.entry_” yapısıyla alınır ve değişkenlerde tutulur. Türleri farklı olan gemiler için gereken bilgiler de alınır ve diğer bilgiler gibi uygun türlere dönüştürülür. (“int”, “float” gibi)
Değişkenlerde tutulan bilgiler daha önce tanımlanan “gemi_ekle()” metodu çağırılarak veritabanında bulunan tabloların ilgili sütunlarına eklenir.

```
def gemi_ekle(self):
    try:
        seri_no = int(self.entry_seri_no.get())
        gemi_adi = self.entry_gemi_adi.get()
        agirlik = float(self.entry_agirlik.get())
        yapim_yili = int(self.entry_yapim_yili.get())
        tur = self.gemi_turu_var.get()

        # Gemi türüne göre ek bilgileri alın
        if tur == "YOLCU":
            yolcu_kapasitesi = int(self.entry_yolcu_kapasitesi.get())
            gemi_ekle(seri_no, gemi_adi, agirlik, yapim_yili, tur, yolcu_kapasitesi=yolcu_kapasitesi)
        elif tur == "PETROL_TANKERI":
            petrol_kapasitesi = float(self.entry_petrol_kapasitesi.get())
            gemi_ekle(seri_no, gemi_adi, agirlik, yapim_yili, tur, petrol_kapasitesi=petrol_kapasitesi)
        elif tur == "KONTEYNER":
            konteyner_sayisi = int(self.entry_konteyner_sayisi.get())
            maks_agirlik = float(self.entry_maks_agirlik.get())
            gemi_ekle(seri_no, gemi_adi, agirlik, yapim_yili, tur, konteyner_sayisi=konteyner_sayisi,
                      maks_agirlik=maks_agirlik)

        messagebox.showinfo("Başarılı", "Gemi başarıyla eklendi.")

        self.entry_seri_no.delete(0, tk.END)
        self.entry_gemi_adi.delete(0, tk.END)
        self.entry_agirlik.delete(0, tk.END)
        self.entry_yapim_yili.delete(0, tk.END)
        self.gemi_turu_secildi()  # Güncelleme

    except ValueError:
        messagebox.showerror("Hata", "Geçerli değerler giriniz.")
```

Gemi türünü formdan alma metodu: “gemi_turu_secildi()”

Bu metod form ekranında gemi türü seçimi yapılırken kullanır. Kullanıcının seçim yaptığı gemi türüne göre ek bilgiler gerekir. Bu metod sayesinde seçim yapılan geminin ek bilgilerinin de alınmasını sağlanır.

```
def gemi_turu_secildi(self):
    gemi_turu = self.gemi_turu_var.get()

    if gemi_turu == "YOLCU":
        self.label_yolcu_kapasitesi.grid(row=5, column=0)
        self.entry_yolcu_kapasitesi.grid(row=5, column=1)
    # Gemi türleri için seçilen türe göre girilecek değerler farklılık gösterir.
        self.label_petrol_kapasitesi.grid_forget()
        self.entry_petrol_kapasitesi.grid_forget()
        self.label_konteyner_sayisi.grid_forget()
        self.entry_konteyner_sayisi.grid_forget()
        self.label_maks_agirlik.grid_forget()
        self.entry_maks_agirlik.grid_forget()

    elif gemi_turu == "PETROL_TANKERI":
        self.label_petrol_kapasitesi.grid(row=5, column=0)
        self.entry_petrol_kapasitesi.grid(row=5, column=1)

        self.label_yolcu_kapasitesi.grid_forget()
        self.entry_yolcu_kapasitesi.grid_forget()
        self.label_konteyner_sayisi.grid_forget()
        self.entry_konteyner_sayisi.grid_forget()
        self.label_maks_agirlik.grid_forget()
        self.entry_maks_agirlik.grid_forget()

    elif gemi_turu == "KONTEYNER":
        self.label_konteyner_sayisi.grid(row=5, column=0)
        self.entry_konteyner_sayisi.grid(row=5, column=1)

        self.label_maks_agirlik.grid(row=6, column=0)
        self.entry_maks_agirlik.grid(row=6, column=1)
```

Gemi silme metodu: “gemi_sil_tk()”

Bu metod gemiyi seri numarasını silmek için arayüzden “silinecek_seri_no()” bilgisini alır ve daha önce tanımlanmış olan “gemi_sil()” metodu ile veritabanından siler. Girilen seri numarasının veritabanında bulunup bulunmadığını kontrol eder ve hata mesajı gönderir.

```python
def gemi_sil(self):
    silinecek_seri_no = self.entry_silinecek_seri_no.get()

    if silinecek_seri_no:
        try:
            silinecek_seri_no = int(silinecek_seri_no)
            gemi_sil(silinecek_seri_no)
            messagebox.showinfo("Başarılı", "Gemi başarıyla silindi.")
            self.entry_silinecek_seri_no.delete(0, tk.END)
        except ValueError:
            messagebox.showerror("Hata", "Geçerli bir seri no giriniz.")
    else:
        messagebox.showerror("Hata", "Lütfen silinecek seri no'yu giriniz.")
```

Gemileri başka bir tablo penceresinde gösterme metodu: “gemileri_göster()”

“tk.Toplevel(self)” ile mevcut pencerenin üstünde yeni bir pencere oluşturulur.
Bu yeni pencereye "Gemi Listesi" başlığı verilir ve boyutu 600x400 olarak ayarlanır.
Yeni pencere de bilgileri listelemek için satır ve sütunlardan oluşan “Treeview” kullanılır.
Daha sonra sütunlarının ismi ve genişlikleri ayarlanır.

“gemileri_al()” fonksiyonu çağrılarak, “GEMILER” tablosundaki tüm gemi kayıtları alınır.
Bu veriler, gemiler listesinde tutulur.Eğer gemiler boş değilse, her gemi için özel bilgileri almak için “gemi_turu_bilgilerini_al(seri_no, tur)” fonksiyonu çağrılır. Sonra “gemi_treeview.insert()” ile her gemi için yeni bir satır eklenir. Sütunlar, geminin seri numarası, adı, ağırlığı, yapım yılı, türü ve özel bilgileri içerir.

```
 def gemileri_goster(self):
     yeni_pencere = tk.Toplevel(self)
     yeni_pencere.title("Gemi Listesi")
     yeni_pencere.geometry("600x400")  # Pencere boyutunu

     gemi_treeview = ttk.Treeview(
         yeni_pencere,
         columns=("serino", "gemiadi", "agirlik", "yapimyili", "tur", "ozel_bilgi"),
         show='headings'
     )

     gemi_treeview.heading("serino", text="Seri No")
     gemi_treeview.heading("gemiadi", text="Gemi Adı")
     gemi_treeview.heading("agirlik", text="Ağırlık")
     gemi_treeview.heading("yapimyili", text="Yapım Yılı")
     gemi_treeview.heading("tur", text="Tür")
     gemi_treeview.heading("ozel_bilgi", text="Özel Bilgi")

     # Treeview sütun genişliklerini ayarla
     gemi_treeview.column("serino", width=50)
     gemi_treeview.column("gemiadi", width=50)
     gemi_treeview.column("agirlik", width=50)
     gemi_treeview.column("yapimyili", width=50)
     gemi_treeview.column("tur", width=50)
     gemi_treeview.column("ozel_bilgi", width=150)

     gemi_treeview.pack(fill='both', expand=True)

     gemiler = gemileri_al()
gemi_treeview.column("yapimyili", width=50)
gemi_treeview.column("tur", width=50)
gemi_treeview.column("ozel_bilgi", width=150)

gemi_treeview.pack(fill='both', expand=True)

gemiler = gemileri_al()

if gemiler:
    for gemi in gemiler:
        seri_no = gemi[0]
        tur = gemi[4]
        ozel_bilgi = ""

        # Gemi türüne göre özel bilgiyi alın
        bilgi = gemi_turu_bilgilerini_al(seri_no, tur)

        if tur == "YOLCU" and bilgi:
            ozel_bilgi = f"Yolcu Kapasitesi: {bilgi[0]}"
        elif tur == "PETROL_TANKERI" and bilgi:
            ozel_bilgi = f"Petrol Kapasitesi: {bilgi[0]} Litre"
        elif tur == "KONTEYNER" and bilgi:
            ozel_bilgi = f"Konteyner Sayısı: {bilgi[0]}, Maksimum Ağırlık: {bilgi[1]} Ton"

        gemi_treeview.insert("", "end", values=(seri_no, gemi[1], gemi[2], gemi[3], tur, ozel_bilgi))
else:
    messagebox.showinfo("Bilgi", "Veritabanında hiç gemi bulunamadı.")
```

Sefer veri tabanı için metotlar:

Veritabanına sefer eklemek için kullanılan metod: “sefer_ekle()”

Bu metod, veritabanına sefer kaydı eklemek için kullanılır. İlk adımda, “sqlite3.connect(“SEFERLER.db”)” ifadesi ile “SEFERLER.db” adlı veritabanına bağlanır. Bir veritabanı imleci (cursor) oluşturulduktan sonra, “imlec_sefer2.execute()” komutuyla “SEFERLER” tablosuna yeni bir sefer kaydı eklenir. Veritabanı değişikliklerini kalıcı hale getirmek için “baglanti_sefer2.commit()” çağrısı kullanılır. Son olarak, “baglanti_sefer2.close()” ile veritabanı bağlantısı kapatılarak işlem tamamlanır.

Kısaca bu metod veritabanına sefer bilgisi ekler ve işlemi tamamlamak için veritabanı bağlantısını kapatır.

```
1 usage
def sefer_ekle(s_ID, s_CIKIS, s_DONUS, s_LIMAN, seri_no):
    baglanti_sefer2 = sqlite3.connect('SEFERLER.db')
    imlec_sefer2 = baglanti_sefer2.cursor()
    imlec_sefer2.execute( _sql: '''
            INSERT INTO SEFERLER (S_ID,S_CIKIS,S_DONUS,S_LİMAN,SERI_NO)
            VALUES (?, ?, ?, ?, ?)
        ''', _parameters: (s_ID, s_CIKIS, s_DONUS, s_LIMAN, seri_no))
    baglanti_sefer2.commit()
    baglanti_sefer2.close()
```

Veritabanından sefer silme metodu: “sefer_sil()”

Bu metod, “SEFERLER.db” adlı veritabanında belirli bir sefer kimliğine (s_ID) göre silmek için kullanılır.
İlk olarak, “sqlite3.connect(“SEFERLER.db”)” ifadesiyle veritabanına bağlantı kurulur. Ardından “imlec = baglanti.cursor() “ile bir veritabanı imleci (cursor) oluşturulur. Metod, “SELECT * FROM SEFERLER WHERE S_ID = ?” sorgusunu çalıştırarak, verilen sefer kimliğine sahip bir kayıt olup olmadığını kontrol eder.

Eğer sefer kaydı bulunursa, “DELETE FROM SEFERLER WHERE S_ID = ?” komutuyla seferi siler ve değişiklikleri “baglanti.commit()” ile kalıcı hale getirir.
Bu durumda, “silindi = True” olarak işaretlenir. Ancak sefer bulunmazsa, silme işlemi gerçekleşmez ve “silindi = False” olur.

Fonksiyon sonunda, “baglanti.close()” ile veritabanı bağlantısı kapatılır.

```
1 usage
def sefer_sil(s_ID):
    baglanti = sqlite3.connect('SEFERLER.db')
    imlec = baglanti.cursor()
    imlec.execute( _sql: 'SELECT * FROM SEFERLER WHERE S_ID = ?', _parameters: (s_ID,))
    sefer = imlec.fetchone()  # Girilen s_ID'de sefer olup olmadığını kontrol edip buna
    if sefer:
        imlec.execute( _sql: 'DELETE FROM SEFERLER WHERE S_ID = ?', _parameters: (s_ID,))
        baglanti.commit()
        silindi = True
    else:
        silindi = False
    baglanti.close()
    return silindi
```

Sefer alma metodu: “sefer_belirle()”

İlk olarak, sqlite3.connect(“SEFERLER.db”) ifadesi ile “SEFERLER.db” adlı veritabanına bağlanılır. Daha sonra, “baglanti_sefer4.cursor()” ile bir veritabanı imleci (cursor) oluşturulur. Metod, “imlec_sefer4.execute(“SELECT * FROM SEFERLER”)” ifadesiyle “SEFERLER” tablosundaki tüm kayıtları seçmek için sorgusu çalıştırır.
“imlec_sefer4.fetchall()” ile sorgunun döndürdüğü tüm sonuçlar alınır ve seferler adlı bir listeye aktarılır.

```
1 usage
def sefer_belirle():
    baglanti_sefer4 = sqlite3.connect('SEFERLER.db')
    imlec_sefer4 = baglanti_sefer4.cursor()
    imlec_sefer4.execute('SELECT * FROM SEFERLER')
    seferler = imlec_sefer4.fetchall()
    baglanti_sefer4.close()
    return seferler
```

Sefer form ekranı için metotlar:

Formdan gelen bilgilerle veri tabanına sefer ekleme metodu: “sefer_ekle_tk()”

“Tkinter” kütüphanesi ile oluşturulan ekranda kullanıcının ilgili alanlara girdiği değerler, “self.entry_” yapısıyla alınır ve değişkenlerde tutulur.
Değişkenlerde tutulan bilgiler daha önce tanımlanan “sefer_ekle()” metodu çağırılarak veritabanında bulunan tabloların ilgili sütunlarına eklenir.

```
def sefer_ekle_tk(self):
    try:
        s_ID = int(self.entry_s_ID.get())
        s_CIKIS = self.entry_s_CIKIS.get()
        s_DONUS = self.entry_s_DONUS.get()
        s_LIMAN = self.entry_s_LIMAN.get()
        seri_no = int(self.entry_seri_no.get())

        if all([s_ID, s_CIKIS, s_DONUS, s_LIMAN, seri_no]):
            sefer_ekle(s_ID, s_CIKIS, s_DONUS, s_LIMAN, seri_no)
            messagebox.showinfo( title: "Başarılı",  message: "Sefer başarıyla eklendi.")

            self.entry_s_ID.delete( first: 0, tk.END)
            self.entry_s_CIKIS.delete( first: 0, tk.END)
            self.entry_s_DONUS.delete( first: 0, tk.END)
            self.entry_s_LIMAN.delete( first: 0, tk.END)
            self.entry_seri_no.delete( first: 0, tk.END)
        else:
            raise ValueError("Eksik bilgiler")
    except ValueError:
        messagebox.showerror( title: "Hata",  message: "Tüm alanları doldurun ve geçerli değerler girin.")
```

Formdan gelen bilgilerde sefer silme metodu: “sefer_sil_tk()”

Bu metod seferin seri numarasını silmek için arayüzden “silinecek_s_ID()” bilgisini alır ve daha önce tanımlanmış olan “sefer_sil()” metodu ile veritabanından siler. Girilen sefer ID nin veritabanında bulunup bulunmadığını kontrol eder ve hata mesajı gönderir.

```
def sefer_sil_tk(self):
    try:
        silinecek_s_ID = int(self.entry_silinecek_sefer_id.get())
        sefer_sil(silinecek_s_ID)
        messagebox.showinfo( title: "Başarılı",  message: "Sefer başarıyla silindi.")

        self.entry_silinecek_sefer_id.delete( first: 0, tk.END)
    except ValueError:
        messagebox.showerror( title: "Hata",  message: "Geçerli bir Sefer ID girin.")
```

Seferleri başka bir tablo penceresinde gösterme metodu: “sefer_göster()”

“tk.Toplevel(self)” ile mevcut pencerenin üstünde yeni bir pencere oluşturulur.
Bu yeni pencereye "Sefer Listesi" başlığı verilir ve boyutu 400x400 olarak ayarlanır.
Yeni pencere de bilgileri listelemek için satır ve sütunlardan oluşan “Treeview” kullanılır.
Daha sonra sütunlarının ismi ve genişlikleri ayarlanır.
“sefer_belirle()” fonksiyonu çağrılarak, “SEFERLER” tablosundaki tüm sefer kayıtları alınır. Bu veriler, seferler listesinde tutulur. Bu bilgiler tabloda gösterilir.

```
1 usage
def sefer_goster(self):
    seferler = sefer_belirle()

    yeni_pencere = tk.Toplevel(self)
    yeni_pencere.title("Sefer Listesi")
    yeni_pencere.geometry("400x400")

    sefer_treeview = ttk.Treeview(
        yeni_pencere,
        columns=("s_id", "s_cikis", "s_donus", "s_liman", "seri_no"),
        show='headings'
    )
```

```
sefer_treeview.heading("s_id", text="Sefer ID")
sefer_treeview.heading("s_cikis", text="Çıkış Tarihi")
sefer_treeview.heading("s_donus", text="Dönüş Tarihi")
sefer_treeview.heading("s_liman", text="Liman")
sefer_treeview.heading("seri_no", text="Gemi Seri No")

sefer_treeview.column("s_id", width=70)
sefer_treeview.column("s_cikis", width=100)
sefer_treeview.column("s_donus", width=100)
sefer_treeview.column("s_liman", width=100)
sefer_treeview.column("seri_no", width=100)

sefer_treeview.pack(fill='both', expand=True)

if seferler:
    for sefer in seferler:
        sefer_treeview.insert( parent: "",  index: "end", values=(
            sefer[0], sefer[1], sefer[2], sefer[3], sefer[4]
        ))
else:
    messagebox.showinfo( title: "Bilgi",  message: "Hiç sefer bulunamadı.")
```

Personel veri tabanı için metotlar:

Veritabanına personel ekleme metodu: “personel_ekle()”

Bu metod, "PERSONEL.db" veritabanına yeni bir personel kaydı eklemek için kullanılır. Metod, benzersiz personelin kimliği, adı, soyadı, vatandaşlığı, doğum tarihi, mesleği, işe giriş tarihi, adresi ve durumu gibi bilgileri alır ve bunları veritabanına ekler.
Önce, “sqlite3.connect('PERSONEL.db')” ile veritabanına bağlanılır, işlem tamamlandıktan sonra “baglanti.commit()” ile değişiklikler kaydedilir son olarak, “baglanti.close()” ile veritabanı bağlantısı kapatılır.

```
1 usage
def personel_ekle(personel_id, ad, soyad, vatandaslik, dogum_tarihi, meslek, ise_giris, adres,durum):
    baglanti = sqlite3.connect('PERSONEL.db')
    imlec = baglanti.cursor()
    imlec.execute( _sql: '''
        INSERT INTO PERSONEL (PERSONEL_ID, PERSONEL_AD, PERSONEL_SOYAD, PERSONEL_VATANDASLIK,
                              PERSONEL_DOGUM_TARIHI, PERSONEL_MESLEGI, PERSONEL_ISE_GIRIS, PERSONEL_ADRESI,PERSONEL_DURUM)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?,?)
    ''', _parameters: (personel_id, ad, soyad, vatandaslik, dogum_tarihi, meslek, ise_giris, adres,durum))
    baglanti.commit()
    baglanti.close()
```

Veritabanından personel silme metodu: “personel_sil()”

Bu metod, SQLite veritabanında belirli bir kimliğe sahip personeli silmek için kullanılır. Metod, önce “sqlite3.connect('PERSONEL.db')” ile "PERSONEL.db" adlı veritabanına bağlanır ve bir veritabanı imleci oluşturur. İmleç, SQL komutlarını çalıştırmak için kullanılır.
“DELETE FROM PERSONEL WHERE PERSONEL_ID=?” sorgusu ile belirli bir personel kimliği (personel_id) ile eşleşen kaydı silmek için bir komut çalıştırılır. Değişiklikler “baglanti.commit()”kaydedilir ve “baglanti.close()” ile veritabanı bağlantısı kapatılır.

```
1 usage
def personel_sil(personel_id):
    baglanti = sqlite3.connect('PERSONEL.db')
    imlec = baglanti.cursor()
    imlec.execute( _sql: '''
        DELETE FROM PERSONEL WHERE PERSONEL_ID=?
    ''', _parameters: (personel_id,))
    baglanti.commit()
    baglanti.close()
```

Personel alma metodu: “personel_al()”

Bu Python fonksiyonu, "PERSONEL.db" adlı bir SQLite veritabanından tüm personel kayıtlarını almak için kullanılır. Fonksiyon, sqlite3.connect('PERSONEL.db') ifadesiyle

veritabanına bağlanır ve bir imleç (cursor) oluşturur. İmleç, SQL sorguları çalıştırmak için kullanılır.

Fonksiyon, imlec.execute('SELECT * FROM PERSONEL') ile "PERSONEL" tablosundaki tüm kayıtları sorgular ve imlec.fetchall() ile bu sorgunun tüm sonuçlarını alır. Bu sonuçlar, personeller adlı bir listede saklanır. Ardından, baglanti.close() ile veritabanı bağlantısı kapatılarak kaynaklar serbest bırakılır. Son olarak, fonksiyon personeller listesini döndürerek, tüm personel kayıtlarını sağlar.

```
1 usage
def personelleri_al():
    baglanti = sqlite3.connect('PERSONEL.db')
    imlec = baglanti.cursor()
    imlec.execute('SELECT * FROM PERSONEL')
    personeller = imlec.fetchall()
    baglanti.close()
    return personeller
```

Personel form ekranı için metotlar:

Formdan gelen bilgilerle veri tabanına personel ekleme metodu: “personel_ekle_tk()”

“Tkinter” kütüphanesi ile oluşturulan ekranda kullanıcının ilgili alanlara girdiği değerler, “self.entry_” yapısıyla alınır ve değişkenlerde tutulur. Türleri farklı olan personel bilgiler uygun türlere dönüştürülür. (“int”, “float” gibi)
Değişkenlerde tutulan bilgiler daha önce tanımlanan “personel_ekle()” metodu çağırılarak veri tabanında bulunan tabloların ilgili sütunlarına eklenir.

```
def personel_ekle_tk(self):
    try:
        personel_id = int(self.entry_personel_id.get())
        ad = self.entry_personel_ad.get()
        soyad = self.entry_personel_soyad.get()
        vatandaslik = self.entry_personel_vatandaslik.get()
        dogum_tarihi = self.entry_personel_dogum_tarihi.get()
        meslek = self.entry_personel_meslek.get()
        ise_giris = self.entry_personel_ise_giris.get()
        adres = self.entry_personel_adresi.get()
        durum = self.entry_personel_durum.get()

        personel_ekle(personel_id, ad, soyad, vatandaslik, dogum_tarihi, meslek, ise_giris, adre

        messagebox.showinfo( title: "Başarılı",  message: "Personel başarıyla eklendi.")
```

```
        self.entry_personel_id.delete( first: 0, tk.END)
        self.entry_personel_ad.delete( first: 0, tk.END)
        self.entry_personel_soyad.delete( first: 0, tk.END)
        self.entry_personel_vatandaslik.delete( first: 0, tk.END)
        self.entry_personel_dogum_tarihi.delete( first: 0, tk.END)
        self.entry_personel_meslek.delete( first: 0, tk.END)
        self.entry_personel_ise_giris.delete( first: 0, tk.END)
        self.entry_personel_adresi.delete( first: 0, tk.END)
        self.entry_personel_durum.delete( first: 0, tk.END)
    except ValueError:
        messagebox.showerror( title: "Hata",  message: "Geçerli değerler giriniz.")
```

Personel silme metodu: “personel_sil_tk()”

```
1 usage
def personel_sil_tk(self):
    try:
        personel_id = int(self.entry_silinecek_personel_id.get())
        personel_sil(personel_id)

        messagebox.showinfo( title: "Başarılı",  message: "Personel başarıyla silindi.")
        self.entry_silinecek_personel_id.delete( first: 0, tk.END)
    except ValueError:
        messagebox.showerror( title: "Hata",  message: "Geçerli bir Personel ID giriniz.")
```

Bu metod personeli ID numarasından silmek için arayüzden “silinecek_personel_id()” bilgisini alır ve daha önce tanımlanmış olan “personel ()” metodu ile veritabanından siler. Girilen ID numarasının veritabanında bulunup bulunmadığını kontrol eder ve hata mesajı gönderir.

Personelleri başka bir tablo penceresinde gösterme metodu: “personelleri_göster()”

“tk.Toplevel(self)” ile mevcut pencerenin üstünde yeni bir pencere oluşturulur.
Bu yeni pencereye "Personel Listesi" başlığı verilir ve boyutu 400x400 olarak ayarlanır.
Yeni pencere de bilgileri listelemek için satır ve sütunlardan oluşan “Treeview” kullanılır.
Daha sonra sütunlarının ismi ve genişlikleri ayarlanır.

“personelleri_al()” fonksiyonu çağrılarak, “PERSONELLER” tablosundaki tüm personel kayıtları alınır. Bu veriler, personeller listesinde tutulur.. Sonra “personel_treeview.insert()” ile her personel için yeni bir satır eklenir. Sütunlar, personellerin bilgilerini içerir.

```
1 usage
def personelleri_goster(self):
    personeller = personelleri_al()

    yeni_pencere = tk.Toplevel(self)
    yeni_pencere.title("Personel Listesi")
    yeni_pencere.geometry("400x400")

    personel_treeview = ttk.Treeview(
        yeni_pencere,
        columns=("personel_id", "ad", "soyad", "vatandaslik", "dogum_tarihi", "meslek", "ise_giris", "adres",
        show='headings'
    )
```

```
personel_treeview.heading("personel_id", text="Personel ID")
personel_treeview.heading("ad", text="Ad")
personel_treeview.heading("soyad", text="Soyad")
personel_treeview.heading("vatandaslik", text="Vatandaşlık")
personel_treeview.heading("dogum_tarihi", text="Doğum Tarihi")
personel_treeview.heading("meslek", text="Meslek")
personel_treeview.heading("ise_giris", text="İşe Giriş Tarihi")
personel_treeview.heading("adres", text="Adres")
personel_treeview.heading("durum", text="Lisans veya Görev")

personel_treeview.column("personel_id", width=70)
personel_treeview.column("ad", width=100)
personel_treeview.column("soyad", width=100)
personel_treeview.column("vatandaslik", width=100)
personel_treeview.column("dogum_tarihi", width=100)
personel_treeview.column("meslek", width=100)
personel_treeview.column("ise_giris", width=100)
personel_treeview.column("adres", width=150)
personel_treeview.column("durum", width=100)

personel_treeview.pack(fill='both', expand=True)
```

```
if personeller:
    for personel in personeller:
        personel_treeview.insert( parent: "", index: "end", values=(
            personel[0], personel[1], personel[2], personel[3], personel[4], personel[5], personel[6], personel[7],
        ))
else:
    messagebox.showerror( title: "Hata", message: "Hiç personel bulunamadı.")
```

Liman veri tabanı için metotlar:

Veritabanına liman eklemek için metod: “liman_ekle()”

Bu metod, veritabanına yeni bir liman kaydı eklemek için kullanılır. liman_adi, ulke, pasaport_istegi, ve demirleme_ucreti parametrelerini alır. Eğer liman adı ve ülke kombinasyonu zaten mevcutsa, Error yakalanır ve kullanıcıya "Bu liman adı ve ülke kombinasyonu zaten mevcut." mesajı gösterilir. Son olarak, baglanti_liman.close() ile veritabanı bağlantısı kapatılarak işlem tamamlanır. Bu metod, liman ekleme işlemini yapar, hata durumunda mesaj gösterir ve veritabanı bağlantısını kapatır.

```
1 usage
def liman_ekle(liman_adi, ulke, pasaport_istegi, demirleme_ucreti):
    baglanti_liman = sqlite3.connect('LIMAN.db')
    imlec_liman = baglanti_liman.cursor()
    try:
        imlec_liman.execute( _sql: '''
            INSERT INTO LIMAN (LIMAN_ADI, ULKE, PASAPORT_ISTEGI, DEMIRLEME_UCRETI)
            VALUES (?, ?, ?, ?)
        ''', _parameters: (liman_adi, ulke, pasaport_istegi, demirleme_ucreti))
        baglanti_liman.commit()
    except sqlite3.IntegrityError:
        messagebox.showerror( title: "Hata", message: "Bu liman adı ve ülke kombinasyonu zaten m
    finally:
        baglanti_liman.close()
```

Veri tabanından liman silme metodu: “liman_sil()”

Bu metod, veritabanından belirli bir liman adını ve ülkesini kullanarak liman kaydını silmek için tasarlanmıştır. Veritabanına bağlanıldıktan sonra bir imleç (cursor) oluşturulur. Metod, “DELETE FROM LIMAN WHERE LIMAN_ADI = ? AND ULKE = ?” komutunu çalıştırarak, belirli bir liman adı ve ülkeye uyan kaydı siler.

“baglanti_liman.commit()” ile değişiklikler kaydedilir. “imlec_liman.rowcount”, silinen satır sayısını verir; bu, silme işleminin başarılı olup olmadığını kontrol etmek için kullanılır. Fonksiyon, “baglanti_liman.close()” ile veritabanı bağlantısını kapatarak işlemi tamamlar.

```
def liman_sil(liman_adi, ulke):
    baglanti_liman = sqlite3.connect('LIMAN.db')
    imlec_liman = baglanti_liman.cursor()
    imlec_liman.execute( _sql: '''
        DELETE FROM LIMAN WHERE LIMAN_ADI = ? AND ULKE = ?
    ''', _parameters: (liman_adi, ulke))
    baglanti_liman.commit()
    silinen_sayisi = imlec_liman.rowcount
    baglanti_liman.close()
    return silinen_sayisi > 0
```

Veri tabanından veri alma metodu: “limanlari_al()”

Bu metod, "LIMAN.db" adlı veritabanından tüm liman kayıtlarını almak için kullanılır. Fonksiyonun başlangıcında, “sqlite3.connect('LIMAN.db')” ifadesiyle veritabanına bağlanılır ve “baglanti_liman.cursor()” ile bir imleç (cursor) oluşturulur.
Fonksiyon, “SELECT * FROM LIMAN” komutunu kullanarak "LIMAN" tablosundaki tüm kayıtları sorgular. “imlec_liman.fetchall()” ile sorgudan dönen tüm sonuçlar alınır ve limanlar adlı bir listeye aktarılır. Son olarak, baglanti_liman.close() ifadesiyle veritabanı bağlantısı kapatılır.

```
1 usage
def limanlari_al():
    baglanti_liman = sqlite3.connect('LIMAN.db')
    imlec_liman = baglanti_liman.cursor()
    imlec_liman.execute('SELECT * FROM LIMAN')
    limanlar = imlec_liman.fetchall()
    baglanti_liman.close()
    return limanlar
```

Liman form ekranı için metotlar:

Formdan gelen bilgilerle veri tabanına liman ekleme metodu: “liman_ekle_tk()”

“Tkinter” kütüphanesi ile oluşturulan ekranda kullanıcının ilgili alanlara girdiği değerler, “self.entry_” yapısıyla alınır ve değişkenlerde tutulur. Türleri farklı olan limanlar için gereken bilgiler de alınır ve diğer bilgiler gibi uygun türlere dönüştürülür. (“int”, “float” gibi)
Değişkenlerde tutulan bilgiler daha önce tanımlanan “liman_ekle()” metodu çağırılarak veri tabanında bulunan tabloların ilgili sütunlarına eklenir.

```
def liman_ekle_tk(self):
    try:
        liman_adi = self.entry_liman_adi.get()
        ulke = self.entry_ulke.get()
        pasaport_istegi = self.entry_pasaport_istegi.get().upper()
        demirleme_ucreti = float(self.entry_demirleme_ucreti.get())

        if pasaport_istegi not in ['E', 'H']:
            raise ValueError("Pasaport gereksinimi yalnızca 'E' veya 'H' olmalıdır.")

        if all([liman_adi, ulke, pasaport_istegi, demirleme_ucreti]):
            liman_ekle(liman_adi, ulke, pasaport_istegi, demirleme_ucreti)
            messagebox.showinfo( title: "Başarılı",  message: "Liman başarıyla eklendi.")

            self.entry_liman_adi.delete( first: 0, tk.END)
            self.entry_ulke.delete( first: 0, tk.END)
            self.entry_pasaport_istegi.delete( first: 0, tk.END)
            self.entry_demirleme_ucreti.delete( first: 0, tk.END)

        else:
            raise ValueError("Eksik bilgiler.")

    except ValueError:
        messagebox.showerror( title: "Hata",  message: "Geçerli ve eksiksiz bilgiler girin.")
```

Liman silme metodu: “liman_sil_tk()”

Bu metod gemiyi ad ve ülke kombinasyonuna göre silmek için arayüzden “silinecek_liman_adi()” ve “silinecek_ulke()” bilgisini alır ve daha önce tanımlanmış olan “liman_sil()” metodu ile veri tabanından siler.

```
1 usage
def liman_sil_tk(self):
    try:
        liman_adi = self.entry_silinecek_liman_adi.get()
        ulke = self.entry_silinecek_ulke.get()

        if all([liman_adi, ulke]):
            silindi = liman_sil(liman_adi, ulke)
            if silindi:
                messagebox.showinfo( title: "Başarılı",  message: "Liman başarıyla silindi.")
                self.entry_silinecek_liman_adi.delete( first: 0, tk.END)
                self.entry_silinecek_ulke.delete( first: 0, tk.END)
            else:
                messagebox.showerror( title: "Hata",  message: "Liman adı ve ülke kombinasyonu bulunamadı.")
        else:
            raise ValueError("Eksik bilgiler.")

    except ValueError:
        messagebox.showerror( title: "Hata",  message: "Geçerli Liman Adı ve Ülke girin.")
```

Gemileri başka bir tablo penceresinde gösterme metodu: “limanları_göster_tk()”

“tk.Toplevel(self)” ile mevcut pencerenin üstünde yeni bir pencere oluşturulur.
Bu yeni pencereye "Liman Listesi" başlığı verilir ve boyutu 400x400 olarak ayarlanır.
Yeni pencere de bilgileri listelemek için satır ve sütunlardan oluşan “Treeview” kullanılır.
Daha sonra sütunlarının ismi ve genişlikleri ayarlanır.

“limanları_al()” fonksiyonu çağrılarak, “LİMANLAR” tablosundaki tüm liman kayıtları alınır. Bu veriler, limanlar listesinde tutulur. Sonra “liman_treeview.insert()” ile her liman için yeni bir satır eklenir. Sütunlar, geminin seri numarası, adı, ağırlığı, yapım yılı, türü ve özel bilgileri içerir.

```python
def limanlari_goster_tk(self):
    limanlar = limanlari_al()

    yeni_pencere = tk.Toplevel(self)
    yeni_pencere.title("Liman Listesi")
    yeni_pencere.geometry("400x400")

    liman_treeview = ttk.Treeview(
        yeni_pencere,
        columns=("liman_adi", "ulke", "pasaport_istegi", "demirleme_ucreti"),
        show='headings'
    )

    liman_treeview.heading("liman_adi", text="Liman Adı")
    liman_treeview.heading("ulke", text="Ülke")
    liman_treeview.heading("pasaport_istegi", text="Pasaport Gereksinimi (Y/N)")
    liman_treeview.heading("demirleme_ucreti", text="Demirleme Ücreti")

    liman_treeview.column("liman_adi", width=100)
    liman_treeview.column("ulke", width=100)
    liman_treeview.column("pasaport_istegi", width=80)
    liman_treeview.column("demirleme_ucreti", width=100)
```

4.2 Görev dağılımı

Kodlama aşaması:

Aybüke Türidi: Veri tabanlarını oluşturma, Sütunları ve özelliklerini belirleme, kullanıcı girişi sağlama

Elif Beyza Beyaz: Form ekranı yapımı, veri geçişini sağlama

Rapor hazırlanma aşaması:

Ortak

4.3 Karşılaşılan zorluklar ve çözüm yöntemleri

Zorluklar:

- Gemi türlerini için ayrı bilgiler alma.

Çözüm Yöntemleri:

- Her gemi türü için aynı veri tabanında farklı tablolar oluşturacak şekilde ayarlanması. (Yabancı anahtar kullanımı ile)

4.4 Proje isterlerine göre eksik yönler

Personel tablosunda lisans ve görevi girilen meslek tipine göre alamıyor. Kullanıcının Kaptan mı mürettebat mı girdiğini bilmesi gerekiyor.

5 TEST VE DOĞRULAMA

5.1 Yazılımın test süreci

Yazılım için ayrı bir test kodu yazılmadı fakat konsoldan olabilecek durumlar için girişler yapıldı.

5.2 Yazılımın doğrulanması

- Çeşitli girişler uygulanarak temel veri tabanı durumlarına uygunluğu test edildi.