CODEWAR WEB APP

WEB PROGRAMLAMA ÖDEV-1 RAPORU

HAZIRLAYANLAR
220229034 Zeynep Özçelik
220229030 İrem Bengü Bal
220229047 Oya Ilgın Akyıldız
220229029 Rukiye Berna Turan

Blog.py daki fonksiyon ve class incelenmesi

Routing: Web uygulamasında URL'lere göre çalışacak kod bloklarını belirlemeye yarar. Gelen URL isteğine göre ilgili Python fonksiyonunu çalıştırmayı sağlar.

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/secim', methods=['GET','POST'])
```

Şablonlama: Dinamik web sayfalarında programlama dilinin çıktılarını belli bir tasarım ile göstermeyi sağlar. Flask Jinja2 şablonlama motorunu kullanmaktadır.

```
from flask import Flask, render_template, request
```

```
            return render_template('result.html', data = secilen)
        return render_template('secim.html', data1=secilen, data2=diger, round = tur)
    else:
        return render_template('secim.html', data1=data[0], data2=data[1], round = tur)
```

Request Nesnesi: İstemcinin sunucuya gönderdiği bilgileri içerir. Bu bilgiler URL parametreleri veya form verileri olabilir.

```
from flask import Flask, render_template, request
```

```
@app.route('/secim', methods=['GET','POST'])
def secim():
    global tur
    if request.method == 'POST':
        secilen = request.form['selected']
```

Kayıt Olma ve Login Formu :

```
from flask import Flask, render_template,request ,url_for,redirect,flash,session
from flask_mysqldb import MySQL
from wtforms import Form, StringField, PasswordField, validators
from passlib.hash import sha256_crypt
import email_validator
import random
```

```
# Kullanıcı kayıt formu
class RegisterForm(Form):
    name = StringField("İsim - Soyisim", validators=[validators.Length(min=4, max=25)])
    username = StringField("Kullanıcı Adı", validators=[validators.Length(min=5, max=35)])
    email = StringField("E-mail", validators=[validators.Email(message="Lütfen geçerli bir email adresi girin")])
    password = PasswordField("Parola:", validators=[
        validators.DataRequired(message="Lütfen bir parola belirleyin"),
        validators.EqualTo(fieldname="confirm", message="Parolanız Uyuşmuyor")
        ])
    confirm = PasswordField("Parola doğrula ")
```

```python
class LoginForm(Form):
    username = StringField("Kullanıcı Adı")
    password = StringField("Parola")
```

WTForms, Flask uygulamalarında form işlemlerini kolaylaştıran ve doğrulama gibi işlemleri sağlayan bir Python kütüphanesidir. WTForms, Flask ile birlikte kullanılarak, kullanıcı girişi, kayıt formu, profil düzenleme formu gibi çeşitli web formu işlemleri için kullanılabilir.

WTForm'sun kullanılabilmesi için

- İlk önce form uygulamasının import edilip tanınması gerekmektedir.
- Form sınıfları aşağıdaki şekilde tanımlanmıştır.

```python
class LoginForm(Form):
```

```python
class RegisterForm(Form):
```

- Forma eklenecek veriler belirlenir ve onlar için wtform özellikleri entegrelenir.

```python
# Kullanıcı kayıt formu
class RegisterForm(Form):
    name = StringField("İsim - Soyisim", validators=[validators.Length(min=4, max=25)])
    username = StringField("Kullanıcı Adı", validators=[validators.Length(min=5, max=35)])
    email = StringField("E-mail", validators=[validators.Email(message="Lütfen geçerli bir email adresi girin")])
    password = PasswordField("Parola:", validators=[
        validators.DataRequired(message="Lütfen bir parola belirleyin"),
        validators.EqualTo(fieldname="confirm", message="Parolanız Uyuşmuyor")
        ])
    confirm = PasswordField("Parola doğrula ")
```

Yukarıda da olduğu gibi RegisterForm sınıfımıza Form özelliğini wtformsun eklentisi olan "Form" kütüphanesinden almaktayız.

Değişkenler =

'name': StringField tipinde bir form alanıdır. Kullanıcıdan isim ve soyisim bilgisini almak için kullanılır. Bu alana uygulanan doğrulama kuralları şunlardır:
validators.Length(min=4, max=25): İsim ve soyismin en az 4 karakter uzunluğunda olması ve en fazla 25 karakter uzunluğunda olması gerektiğini belirtir. Bu doğrulama kuralı, ismin minimum ve maksimum uzunluklarını belirler.

'username': StringField tipinde bir form alanıdır. Kullanıcı adı bilgisini almak için kullanılır. Bu alana uygulanan doğrulama kuralları şunlardır:
validators.Length(min=5, max=35): Kullanıcı adının en az 5 karakter uzunluğunda olması ve en fazla 35 karakter uzunluğunda olması gerektiğini belirtir. Bu doğrulama kuralı, kullanıcı adının minimum ve maksimum uzunluklarını belirler.

‘email’: StringField tipinde bir form alanıdır. E-posta adresi bilgisini almak için kullanılır. Bu alana uygulanan doğrulama kuralları şunlardır:
validators.Email(message="Lütfen geçerli bir email adresi girin"): Girilen değerin geçerli bir e-posta adresi olup olmadığını kontrol eder. Eğer geçerli değilse, belirtilen hata mesajını kullanıcıya gösterir.

‘**password**’: PasswordField tipinde bir form alanıdır. Kullanıcıdan parola bilgisini almak için kullanılır. Bu alana uygulanan doğrulama kuralları şunlardır:

1. **validators.DataRequired(message="Lütfen bir parola belirleyin"):** Parola alanının boş olmamasını sağlar. Eğer boşsa, belirtilen hata mesajını kullanıcıya gösterir.
2. **validators.EqualTo(fieldname="confirm", message="Parolanız Uyuşmuyor"):** Parola alanı ile parola doğrulama alanı (confirm) arasında eşleşme olup olmadığını kontrol eder. Eğer eşleşmiyorsa, belirtilen hata mesajını kullanıcıya gösterir.

‘confirm’: PasswordField tipinde bir form alanıdır. Kullanıcının parolayı doğrulamak için tekrar girmesi gereken alandır. Bu alanın doğrulama kuralları password alanında zaten tanımlanmıştır ve doğrulama işlemi password alanında gerçekleştirilir.

```
class LoginForm(Form):
    username = StringField("Kullanıcı Adı")
    password = StringField("Parola")
```

Yukarıda da wtform özelliklerin Form eklentisini ve özelliklerini LoginForm classımıza entegre etmekteyiz. LoginForm sınıfı, kullanıcı giriş formunda kullanılacak form alanlarını tanımlamaktadır.

“username”: StringField tipinde bir form alanıdır. Kullanıcı adı bilgisini almak için kullanılır. Doğrulama kuralları burada tanımlanmamıştır, çünkü kullanıcı adı alanının boş olup olmadığı kontrol edilse bile, kullanıcı adı ve parolanın doğruluğu veritabanında kontrol edilecektir.

“password”: StringField tipinde bir form alanıdır. Kullanıcıdan parola bilgisini almak için kullanılır. Doğrulama kuralları burada tanımlanmamıştır, çünkü parolanın doğruluğu veritabanında kontrol edilecektir.

FLASK-MYSQL BAĞLANTISI

```
app = Flask(__name__)
app.secret_key = "test"
app.config["MYSQL_HOST"] = "localhost" #sunucu adresi yazılır
app.config["MYSQL_USER"] = "root"
app.config["MYSQL_PASSWORD"] = ""
app.config["MYSQL_DB"] = "kullanıcılar"
app.config["MYSQL_CURSORCLASS"] = "DictCursor"

mysql = MySQL(app)
```

- **app.secret_key = "test":** Flask uygulamasının kullanacağı gizli anahtarı belirtir. Gizli anahtar, kullanıcı oturumlarının güvenliğini sağlamak için kullanılır. Özellikle oturum

verilerini şifrelemek için kullanılır. Bu anahtarın rastgele ve güvenli bir değer olması önemlidir.

- **app.config["MYSQL_HOST"] = "localhost":**
  MySQL veritabanına bağlanmak için kullanılacak sunucunun adresini belirtir. "localhost" değeri, MySQL sunucusunun yerel makinede çalıştığını ve aynı makinede çalışan Flask uygulamasıyla iletişim kuracağını gösterir. Eğer MySQL sunucusu farklı bir adreste çalışıyorsa, bu adrese uygun bir IP adresi veya alan adı sağlanmalıdır.

- **app.config["MYSQL_USER"] = "root":** MySQL veritabanına bağlanmak için kullanılacak kullanıcı adını belirtir. Bu örnekte, root kullanıcı adı kullanılmıştır. Gerçek bir uygulamada, güvenlik nedenleriyle, veritabanı erişimi için daha kısıtlı bir kullanıcı hesabı kullanılmalıdır.

- **app.config["MYSQL_PASSWORD"] = "":** MySQL veritabanına bağlanmak için kullanılacak parolayı belirtir. Boş bir değer verildiği durumda, parola kullanılmayacağı anlamına gelir. Gerçek bir uygulamada, güvenlik nedenleriyle, güçlü bir parola kullanılmalıdır.

- **app.config["MYSQL_DB"] = "kullanıcılar":** Bağlanılacak MySQL veritabanının adını belirtir. Bu örnekte, kullanıcılar adında bir veritabanı kullanılacağı belirtilmiştir. Gerçek bir uygulamada, bu değer projenin gereksinimlerine ve veritabanı yapısına uygun olarak belirlenmelidir.

- **app.config["MYSQL_CURSORCLASS"] = "DictCursor":** MySQL veritabanı üzerinde yapılan sorguların sonuçlarını almak için kullanılacak cursor sınıfını belirtir. **"DictCursor"** değeri, sorgu sonuçlarının bir Python sözlüğü olarak alınmasını sağlar. Bu, veritabanı kayıtlarının sözlükler şeklinde erişilebilir ve daha okunabilir bir şekilde elde edilmesini sağlar.

- **mysql = MySQL(app):** Flask-MySQL bağlantısını oluşturur.MYSQL nesnesi, Flask uygulamasına MySQL veritabanı işlemleri yapmak için gerekli olan araçları sağlar. Bu nesne, Flask uygulamasının yapılandırmasını (app nesnesini) parametre olarak alır ve MySQL bağlantısını başlatır. Artık bu nesne aracılığıyla MySQL veritabanı ile etkileşim sağlanabilir.

Kayıt Ol fonksiyonu

```
from flask import Flask, render_template,request ,url_for,redirect,flash,session
from flask_mysqldb import MySQL
from wtforms import Form, StringField, PasswordField, validators
from passlib.hash import sha256_crypt
import email_validator
import random
```

Yukarıda Kayıt olma işleminin gerçekleşeceği alanda kullanılan özellikler altı çizili olarak belirtilmiştir.

```
# register page
@app.route("/register",methods = ["GET","POST"])
```

Register işlemimizin gerçekleşebilmesi için ilk önce işlemlerini yapacağımız URL bloğunu route eklentisi sayesinde belirlemekteyiz.

“methods” parametresi, bir Flask route'unun hangi HTTP yöntemlerini kabul edeceğini belirten bir parametredir. Bu parametre, bir HTTP isteğinin hangi metodlarla işlenebileceğini belirtir. Bir route hem GET hemde POST isteklerini işleyebilmektedir.

GET: Bu metod, bir kaynağın okunması için kullanılır. Kullanıcı tarayıcısına bir URL girip sayfayı çağırdığında, tarayıcı aslında bir GET isteği gönderir. Bu yöntem, sunucudan bilgi almak için kullanılır ve genellikle sayfa görüntülemesi, veri okuması gibi işlemlerde kullanılır.

POST: Bu metod, bir kaynağa veri göndermek için kullanılır. Kullanıcı bir form doldurup gönderdiğinde veya bir komut gönderdiğinde, tarayıcı bir POST isteği gönderir. Bu yöntem, sunucuya veri göndermek için kullanılır ve genellikle form gönderimi, veri oluşturma veya güncelleme gibi işlemlerde kullanılır.

Sonra register fonksiyonu oluşturulmuştur, Kullanıcı kayıt sayfasını oluşturur ve kullanıcıların kayıt işlemlerini gerçekleştirmesini sağlar.

```
def register():
    form = RegisterForm(request.form)
```

form = RegisterForm(request.form): HTTP isteği içinden gelen form verileri, RegisterForm sınıfından bir form nesnesine dönüştürülür. Böylece, HTML formundan gelen verilerin doğrulanması ve işlenmesi sağlanır. HTTP’den form verileri Flask’ın request eklentisi ile alınır ve daha sonra formda username ve password şeklinde iki alan varsa , bu form gönderildiğinde, route içindeki POST isteğiyle gönderilen form alanlarının değerleri içerilir.

```
    if request.method == "POST" and form.validate():
        name = form.name.data
        username = form.username.data
        email = form.email.data
        password = sha256_crypt.encrypt(form.password.data)

        cursor = mysql.connection.cursor()

        sorgu = "Insert into users(name,email,username,password) VALUES(%s,%s,%s,%s)"

        cursor.execute(sorgu,(name,email,username,password))
        mysql.connection.commit()

        cursor.close()
        flash("Başarıyla kayıt oldunuz...","success")

        return redirect(url_for("login"))
```

- **if request.method == "POST" and form.validate():** İstek metodu, HTTP POST isteği ise ve form verileri doğrulama kurallarını geçiyorsa, yani form verileri geçerliyse, bu koşul sağlanır. Kullanıcı formu doldurduktan sonra "Kayıt Ol" butonuna bastığında ve formdaki veriler doğrulama kurallarını karşılıyorsa, bu blok çalışır.

- **name = form.name.data, username = form.username.data, email = form.email.data, password = sha256_crypt.encrypt(form.password.data):**

RegisterForm nesnesinden alınan veriler, ilgili değişkenlere atanır. Bu değişkenler, kullanıcının kayıt formunda girdiği isim, kullanıcı adı, e-posta ve şifre bilgilerini temsil eder. Şifre, sha256_crypt kullanılarak şifrelenir.

- **cursor = mysql.connection.cursor():**

  Veritabanına bağlantı kurmak için bir cursor nesnesi oluşturulur. Bu nesne, veritabanı işlemlerini gerçekleştirmek için kullanılacak bir araçtır.

- **'sorgu' = "Insert into users(name,email,username,password) VALUES(%s,%s,%s,%s)":**

  SQL sorgusu oluşturulur. Bu sorgu, kullanıcının veritabanına eklenmesini sağlar. %s placeholder'ları, sorguya dışarıdan gelecek olan değerlerin yerini belirtir.

- **cursor.execute(sorgu,(name,email,username,password)):**

  Oluşturulan SQL sorgusu, execute() metodu aracılığıyla veritabanında çalıştırılır. Placeholder'lara karşılık gelen değerler, tuple içinde sağlanır.

- **mysql.connection.commit():**

  Yapılan değişikliklerin veritabanına kalıcı olarak kaydedilmesi için commit() metodu çağrılır.

- **cursor.close():**

  Cursor nesnesi kapatılır. Veritabanı işlemleri bittikten sonra, cursor nesnesinin kapatılması iyi bir uygulama pratiğidir.

- **flash("Başarıyla kayıt oldunuz...","success"):**

  Flask uygulamasında mesaj göstermek için kullanılan flash() fonksiyonu aracılığıyla, kullanıcıya başarılı bir şekilde kayıt olduğu bildirilir.

- **return redirect(url_for("login")):**

  Kayıt işlemi başarılı olduğunda, kullanıcı otomatik olarak giriş sayfasına yönlendirilir. Bu, login fonksiyonunu çağırarak gerçekleşir.

- **else: return render_template("register.html", form= form):** Eğer HTTP isteği POST değilse veya form doğrulama kurallarını geçmiyorsa, yani kullanıcı sayfayı ilk defa açtığında veya formdaki veriler hatalıysa, kayıt formu tekrar gösterilir. Bu durumda, kullanıcıya hatalı girişleri hakkında bilgi verilir ve formda girdiği bilgilerle tekrar doldurulmuş form sunulur.

GİRİŞ FONKSİYONU

```
from flask import Flask, render_template,request ,url_for,redirect,flash,session
from flask_mysqldb import MySQL
from wtforms import Form, StringField, PasswordField, validators
from passlib.hash import sha256_crypt
import email_validator
import random
```

Giriş işlemleri için kullanılacak entegreler yukarıda belirtilmiştir.

```python
@app.route("/login", methods =["GET","POST"])

def login():
    form= LoginForm(request.form)
    if request.method =="POST":
     username = form.username.data
     password_entered = form.password.data

     cursor = mysql.connection.cursor()
     sorgu = "Select * From users where username = %s"
     result = cursor.execute(sorgu,(username,))

     if result > 0 :

         data = cursor.fetchone()
         real_password = data["password"]
         if sha256_crypt.verify(password_entered,real_password):
             flash("Başarıyla giriş yapıldı ...","success")
             session["logged_in"] = True #giriş anahtar değeri
             session["username"] = username
             return redirect(url_for("ana_sayfa"))
         else:
             flash("Parolanızı Yanlış Girdiniz...","danger")
             return redirect(url_for("login"))
     else:
          flash("Böyle bir kullanıcı bulunamadı","danger")
          return redirect(url_for("login"))
    return render_template("login.html",form= form)
```

- **@app.route("/login", methods=["GET","POST"])**: **/login** URL'sine yapılan hem GET hem de POST isteklerini kabul eden bir Flask route'u tanımlar. Bu route, kullanıcı giriş sayfasını gösterir ve kullanıcı girişi formunun işlenmesini sağlar.

- **def login()**:: **login** fonksiyonu, kullanıcı girişi işlemlerini gerçekleştirir. Fonksiyon, kullanıcı giriş formunu gösterir ve form gönderildiğinde form verilerini işler.

- **form = LoginForm(request.form)**: **LoginForm** sınıfından bir form nesnesi oluşturulur. Bu form nesnesi, kullanıcı giriş formunda kullanılacak form alanlarını ve doğrulama kurallarını içerir.

- **if request.method == "POST"**:: HTTP isteği bir POST isteği ise, yani kullanıcı giriş formunu doldurup gönderdiyse, bu koşul sağlanır.

- **username = form.username.data**: Form nesnesinden kullanıcı adı bilgisini alır.

- **password_entered = form.password.data**: Form nesnesinden girilen parola bilgisini alır.

- **cursor = mysql.connection.cursor()**: MySQL veritabanına bağlantı kurmak için bir cursor nesnesi oluşturulur.
- **sorgu = "Select * From users where username = %s"**: Veritabanında kullanıcı adına göre kullanıcıyı sorgulamak için SQL sorgusu oluşturulur.

- **result = cursor.execute(sorgu, (username,))**: SQL sorgusu veritabanında çalıştırılır ve sonuç alınır. Eğer kullanıcı adına sahip bir kullanıcı varsa, **result** değişkeni 1'den büyük olacaktır.

- **if result > 0:**: Kullanıcı adına sahip bir kullanıcı varsa, bu koşul sağlanır ve içeri girilir.

- **data = cursor.fetchone()**: Veritabanından alınan kullanıcı bilgileri, bir sözlük olarak **data** değişkenine atanır.

- **real_password = data["password"]**: Veritabanındaki kayıttaki şifre bilgisi **real_password** değişkenine atanır.

- **if sha256_crypt.verify(password_entered, real_password):**: Girilen parola, veritabanındaki şifre ile karşılaştırılır. Eğer doğrulanırsa, kullanıcı başarılı bir şekilde giriş yapmıştır ve bu durumda ilgili mesaj kullanıcıya gösterilir, oturum bilgileri ayarlanır ve anasayfaya yönlendirilir.

- **(1)else:**: Parola doğrulanamazsa, kullanıcıya hatalı giriş bildirimi gösterilir ve giriş sayfasına yeniden yönlendirilir.

- **(2)else:**: Eğer kullanıcı adına sahip bir kullanıcı bulunamazsa, yani sorgu sonucu boşsa, bu durumda da kullanıcıya uygun bir hata mesajı gösterilir ve giriş sayfasına yönlendirilir.

- **return render_template("login.html", form=form)**: HTTP isteği bir GET isteği ise veya kullanıcı giriş formu doğrulanamazsa, giriş sayfası tekrar gösterilir. Bu, **render_template** fonksiyonu ile gerçekleştirilir ve kullanıcı giriş formu verileriyle birlikte gönderilir.

ÇIKIŞ FONKSİYONU

```python
#logout
@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for("ana_sayfa"))
```

- **@app.route("/logout"):** /logout URL'sine yapılan GET isteklerini kabul eden bir Flask route'u tanımlar. Bu route, kullanıcıların oturumlarını sonlandırmak için kullanılır.

- **def logout()::** logout fonksiyonu, kullanıcı oturumlarını sonlandırır.
- **session.clear():** session nesnesinin clear() metodu çağrılarak, kullanıcının oturumu temizlenir. Bu, kullanıcının oturumuna ait tüm bilgilerin silinmesini sağlar.
- **return redirect(url_for("ana_sayfa")):** Kullanıcının oturumu sonlandırıldıktan sonra, redirect() fonksiyonu aracılığıyla kullanıcıyı anasayfaya (ana_sayfa route'u) yönlendirir.

SEÇİM FONKSİYONU(TEST İÇİN)

```
from flask import Flask, render_template,request ,url_for,redirect,flash,session
from flask_mysqldb import MySQL
from wtforms import Form, StringField, PasswordField, validators
from passlib.hash import sha256_crypt
import email_validator
import random
```

```
data = ["Python", "C", "C++", "Java", "JavaScript", "R", "Kotlin", "Swift", "PHP", "Pascal", "Ruby", "Objective-C", "Go", "SQL",
"Typescript",]


tur = 0

@app.route('/secim', methods=['GET', 'POST'])
def secim():
    if not session.get('logged_in'):
        flash("Lütfen giriş yapınız!", "danger")
        return redirect(url_for('login'))  # Redirect to the login page if not logged in
```

- **data = ["Python", "C", "C++", "Java", "JavaScript", "R", "Kotlin", "Swift", "PHP", "Pascal", "Ruby", "Objective-C", "Go", "SQL", "Typescript",]:** Kullanıcıya sunulacak seçeneklerin listesi oluşturulur. Bu liste, bir dizi programlama dilini içerir.

- **tur = 0:** Kullanıcının kaç turda olduğunu takip etmek için bir sayaç oluşturulur ve başlangıç değeri 0 olarak ayarlanır.

- **@app.route('/secim', methods=['GET', 'POST']):** /secim URL'sine yapılan hem GET hem de POST isteklerini kabul eden bir Flask route'u tanımlanır. Bu route, kullanıcıya seçim yapma işlemini sunar.

- **def secim()::** secim fonksiyonu, kullanıcıya seçim yapma işlemini gerçekleştirir.

- **if not session.get('logged_in'): ... else::** Eğer kullanıcı oturumu açık değilse, yani kullanıcı giriş yapmamışsa, kullanıcıya "Lütfen giriş yapınız!" uyarısı gösterilir ve giriş sayfasına yönlendirilir.

```
    else:
        global tur
        if request.method == 'POST':
            secilen = request.form['selected']
            diger = random.choice(data)
```

- **global tur:** tur değişkeni, bu fonksiyon içinde kullanılacağı için global olarak tanımlanır.

- **if request.method == 'POST':**: Eğer HTTP isteği bir POST isteği ise, yani kullanıcı bir seçim yapmışsa, bu koşul sağlanır.
- **secilen = request.form['selected']:** Kullanıcının seçtiği dil bilgisini formdan alır.
- **diger = random.choice(data):** Kullanıcının seçtiği dil dışında rastgele bir dil seçer.
- **data.remove(diger):** Seçilen dil, data listesinden kaldırılır, böylece bir dil birden fazla kez seçilemez.
- **tur += 1:** Tur sayacı artırılır.

```
        if not data:
            return render_template('result.html', data=secilen)
        return render_template('secim.html', data1=secilen, data2=diger, round=tur, resim1="static/" + secilen.lower() + ".png",
        resim2="static/" + diger.lower() + ".png")
    else:
        if data:
            return render_template('secim.html', data1=data[0], data2=data[1], round=tur, resim1="static/" + data[0].lower() + ".
            png", resim2="static/" + data[1].lower() + ".png")
        else:
            flash("No choices available!", "danger")
            return render_template('result.html')
```


- **if not data::** Eğer data listesi boşsa, yani seçilecek dil kalmamışsa, bu koşul sağlanır ve sonuç sayfasına yönlendirilir. Sonuç sayfasında, kullanıcının son seçtiği dil gösterilir.

- **return render_template('secim.html', ...):** Eğer data listesi boş değilse, yani seçilecek dil daha varsa, seçim ekranı tekrar gösterilir. Bu ekranda, kullanıcının son seçtiği dil ve bir sonraki seçenek arasında rastgele bir seçim yapması istenir.

- **(1)else::** Eğer HTTP isteği bir GET isteği ise, yani kullanıcı daha önce herhangi bir dil seçmemişse, bu koşul sağlanır ve kullanıcıya seçim ekranı gösterilir.

- **if data: ... else: ...:** Eğer data listesi boş değilse, yani seçilecek dil daha varsa, seçim ekranı gösterilir. Eğer data listesi boş ise, yani seçilecek dil kalmamışsa, kullanıcıya "No choices available!" uyarısı gösterilir ve sonuç sayfasına yönlendirilir.

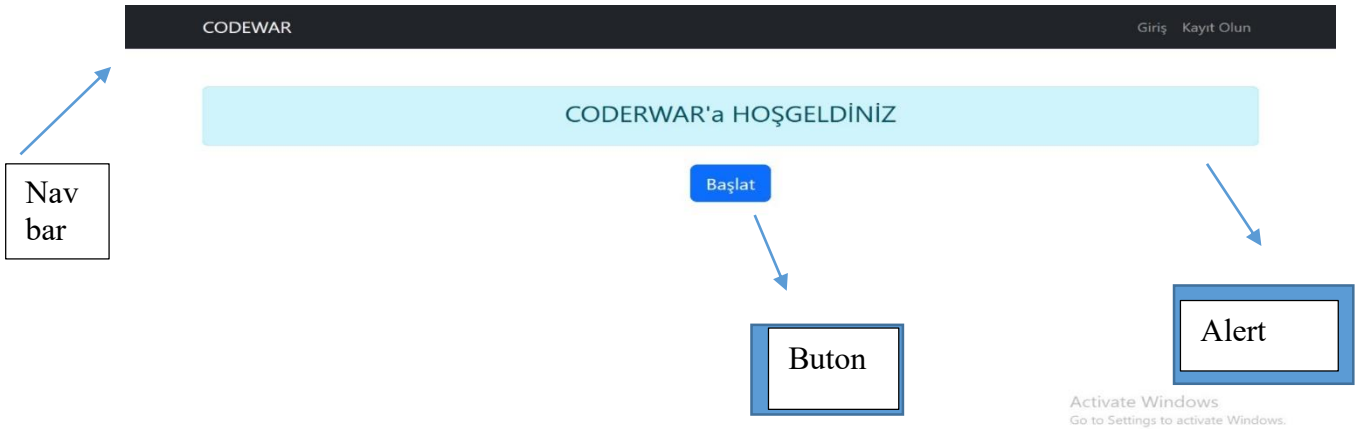
```
if __name__ == "__main__":
    app.run(debug=True)
```

- **if __name__ == "__main__"::** Bu koşul, Python betiğinin doğrudan çalıştırıldığı zamanı kontrol eder. Eğer betik doğrudan çalıştırılıyorsa (__name__ değişkeninin değeri "__main__" ise), içindeki kod bloğu çalıştırılır.

- **app.run(debug=True):** Bu kod, Flask uygulamasını çalıştırmak için kullanılır. run() metodu, Flask uygulamasını başlatır ve belirtilen parametrelere göre ayarlarını yapar. debug=True parametresi, hata ayıklama modunu açar. Bu modda, uygulama hataları daha ayrıntılı bir şekilde gösterilir ve değişiklikler otomatik olarak algılanır ve uygulanır.

Bootstrap5 Entegrasyonu ve Template kısımları

Kullanılar entegrasyonların web arayüzündeki görünümü aşağıdaki gibidir.



navbar.html

```
oject > Flask_E > templates > includes > navbar.html > nav.navbar.navbar-expand-sm.navbar-dark.bg-dark
<nav class="navbar navbar-expand-sm navbar-dark bg-dark">
  <div class="container">
    <a class="navbar-brand" href="#">CODEWAR</a>

    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" ari
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse justify-content-end" id="navbarNav">
      <ul class="navbar-nav">
        {% if session["logged_in"] %}
        <li class="nav-item">
          <a class="nav-link" href="/logout">Çıkış</a>
        </li>
        {% else %}
        <li class="nav-item">
          <a class="nav-link" href="/login">Giriş</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/register">Kayıt Olun</a>
        </li>
        {% endif %}
      </ul>
    </div>
  </div>
</nav>
```

<nav class="navbar navbar-expand-sm navbar-dark bg-dark">: Bu kod parçası bir navigasyon çubuğu oluşturur. Bootstrap sınıflarıyla birlikte, bu navigasyon çubuğu koyu renkli (bg-dark) ve açıldığında genişleyebilen (navbar-expand-sm) bir yapıya sahiptir.

: Bu div içerisinde navbar içeriği yer alır. container sınıfı, içeriğin genişliğini sınırlar ve içeriğin ortalanmasını sağlar.
CODEWAR: Navbar'ın sol tarafında bulunan marka adını temsil eden bir bağlantıdır (navbar-brand). href="#" boş bir bağlantı olduğunu belirtir.
<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">: Bu buton, küçük ekranlarda navbar'ın açılıp kapanmasını sağlar. Bootstrap tarafından sağlanan navbar-toggler sınıfı ile stilize edilmiştir.
: Butonun içerisinde bulunan simge, kullanıcıya navbar'ın açılıp kapanabilir olduğunu gösterir.
: Bu div, navbar içeriğini küçük ekranlarda gizlemek ve açılıp kapanabilmesini sağlamak için kullanılır. collapse ve navbar-collapse sınıfları, içeriğin küçük ekranlarda gizlenebilir olmasını sağlar. justify-content-end sınıfı içeriği sağa hizalar.
<ul class="navbar-nav">: Navbar içeriğini liste halinde tutmak için kullanılan bir <ul> etiketi.
{% if session["logged_in"] %} ve {% endif %}: Bu Django şablon etiketleri, oturum durumuna bağlı olarak farklı navbar öğelerinin gösterilmesini sağlar.
<li class="nav-item">: Navbar içeriğindeki her bir öğeyi temsil eden bir liste öğesi. nav-item sınıfı, Bootstrap'te bir navbar öğesi olduğunu belirtir.
Çıkış ve Giriş: Navbar'da bulunan bağlantılar. nav-link sınıfı, bağlantının navbar içeriğiyle uyumlu olmasını sağlar.
Kayıt Olun: Kullanıcı kaydı oluşturmak için bir bağlantı sağlar.
Bu şekilde, verilen HTML kod parçası Bootstrap ile stilize edilmiş bir navbar oluşturur ve kullanıcının oturum durumuna göre farklı bağlantıları görüntüler.
Bu kısımda arayüzde bulunan navigation bar entegrasyonu yer almaktadır.

layout.html:

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIds1K1eN7N6jIeHz" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js" integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js" integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bglidoEyl4H8zUF0QKbrJ0EcQF" crossorigin="anonymous"></script>

</body>
```

Scriptler:

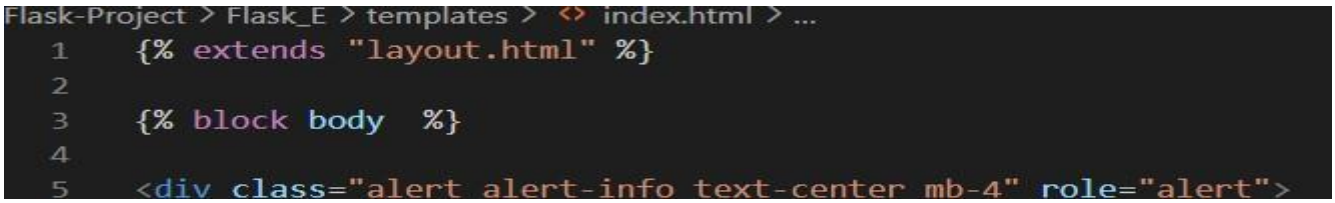
Bootstrap JavaScript: Bootstrap JavaScript dosyalarını ekler. bootstrap.bundle.min.js, popper.min.js, ve bootstrap.min.js dosyaları eklendi.

Bu scriptler, Bootstrap bileşenlerinin işlevselliğini sağlar ve bazı ek özellikler için gereklidir.

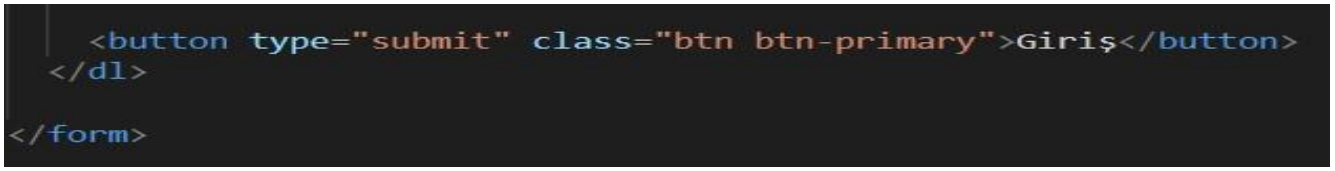
index.html:

```
<div class="text-center mt-4"> <!-- Bu div içinde butonu ortalamak için text-center sınıfını kullanıyoruz -->
    <a href="{{ url_for('secim') }}">
        <button type="button" class="btn btn-primary btn-lg">Başlat</button>
    </a>
</div>
```

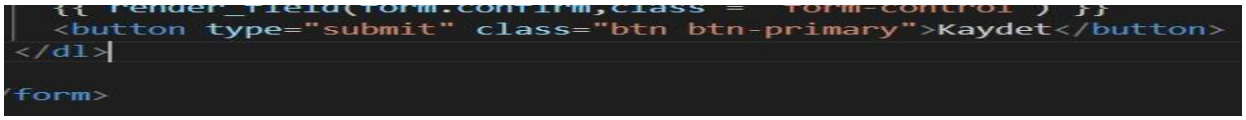
: Bootstrap text-center sınıfı ile içeriği yatay olarak ortalar. mt-4 sınıfı ise üstten 4 birim boşluk bırakır.
<button type="button" class="btn btn-primary btn-lg">Başlat</button>: Bootstrap btn ve btn-primary sınıflarıyla bir buton oluşturur. btn-lg sınıfı butonun boyutunu büyük yapar. Bootstrap ile stilize edilmiş bir buton oluşturur.



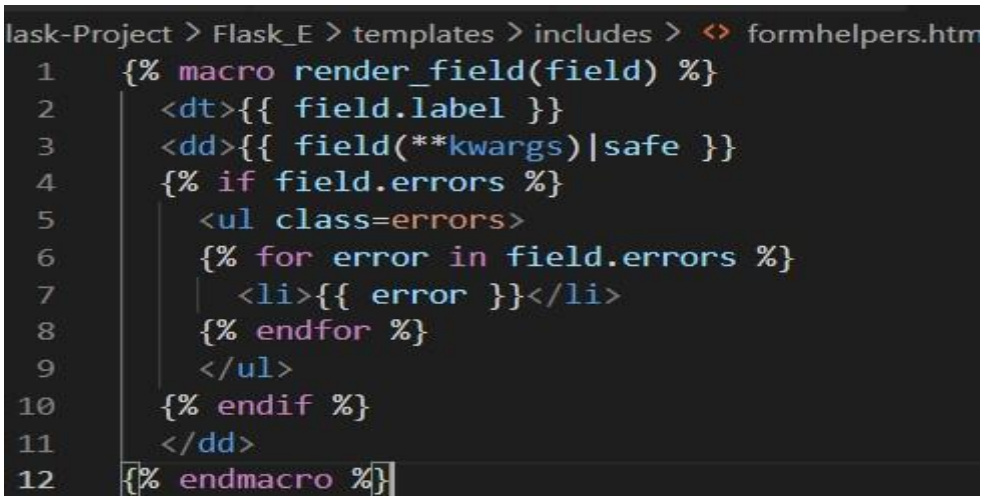
Buradaki alert kısmı da bootstrapden alınmıştır. Ayrıca html kodlarının birçok yerinde yukarıdaki {% extends "layout.html" %}{% block body %} kısmında olduğu gibi jinja templateler kullanılmıştır.
login.html:



<button type="submit" class="btn btn-primary">Giriş</button>: Bu kod parçası, HTML <button> etiketiyle bir buton oluşturur. type="submit" özelliği, bu düğmenin bir form gönderme butonu olduğunu belirtir. Yani, bu butona basıldığında, bir formun gönderilmesini tetikler.
register.html:



formhelpers.html:



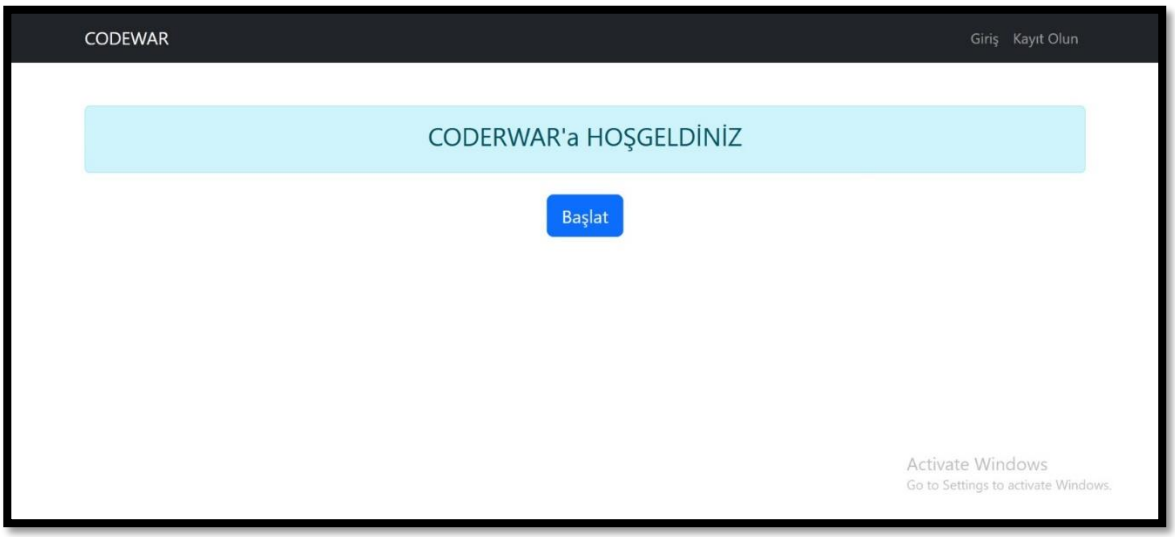
Bu kısımda flask in hazır formhelpers template'i kullanılmıştır.

messages.html

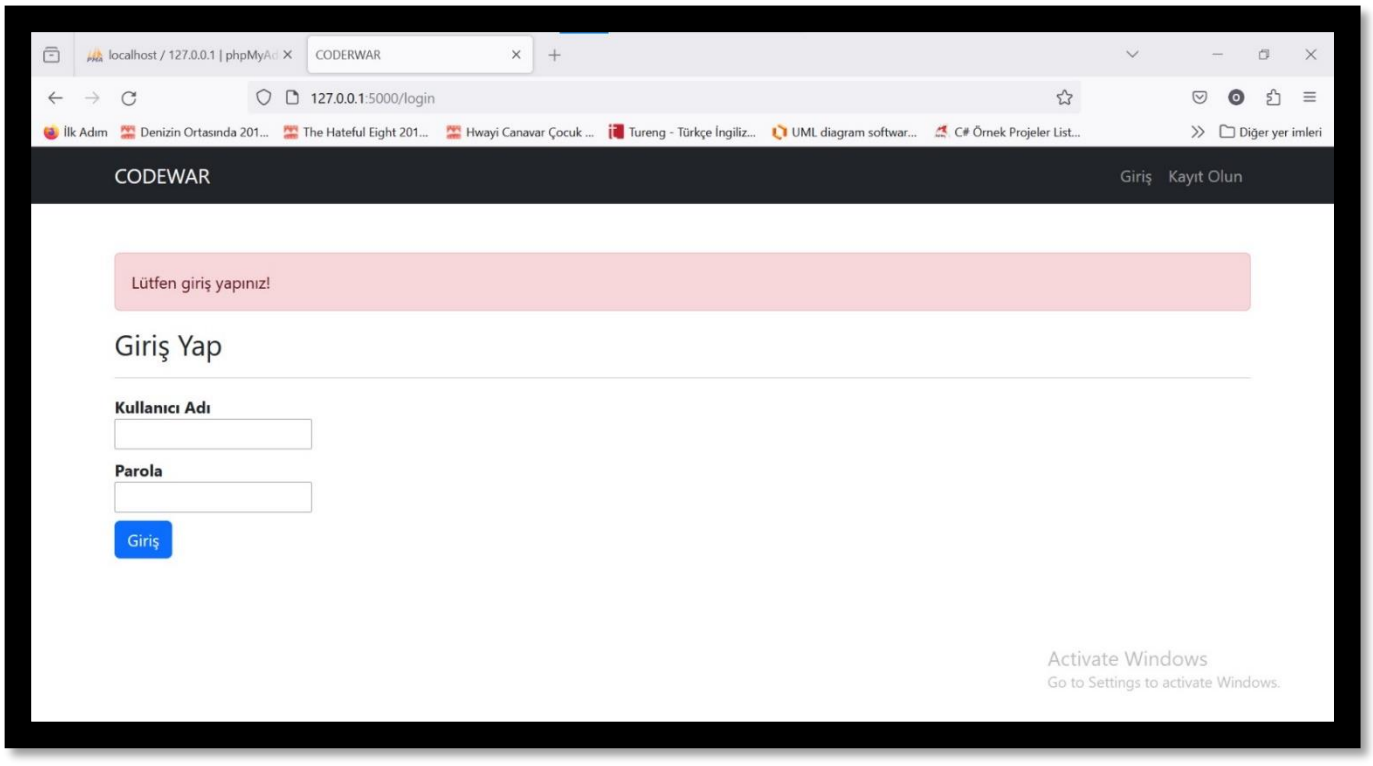
```
{% with messages = get_flashed_messages(with_categories=true) %}
  {% if messages %}

    {% for category, message in messages %}
    <div class="alert alert-{{category}}" role="alert">
        {{message}}
      </div>
    {% endfor %}

  {% endif %}
{% endwith %}
```

Burada da bootstrap alert özelliği kullanılmıştır. Arayüzdeki mavi penceredir.

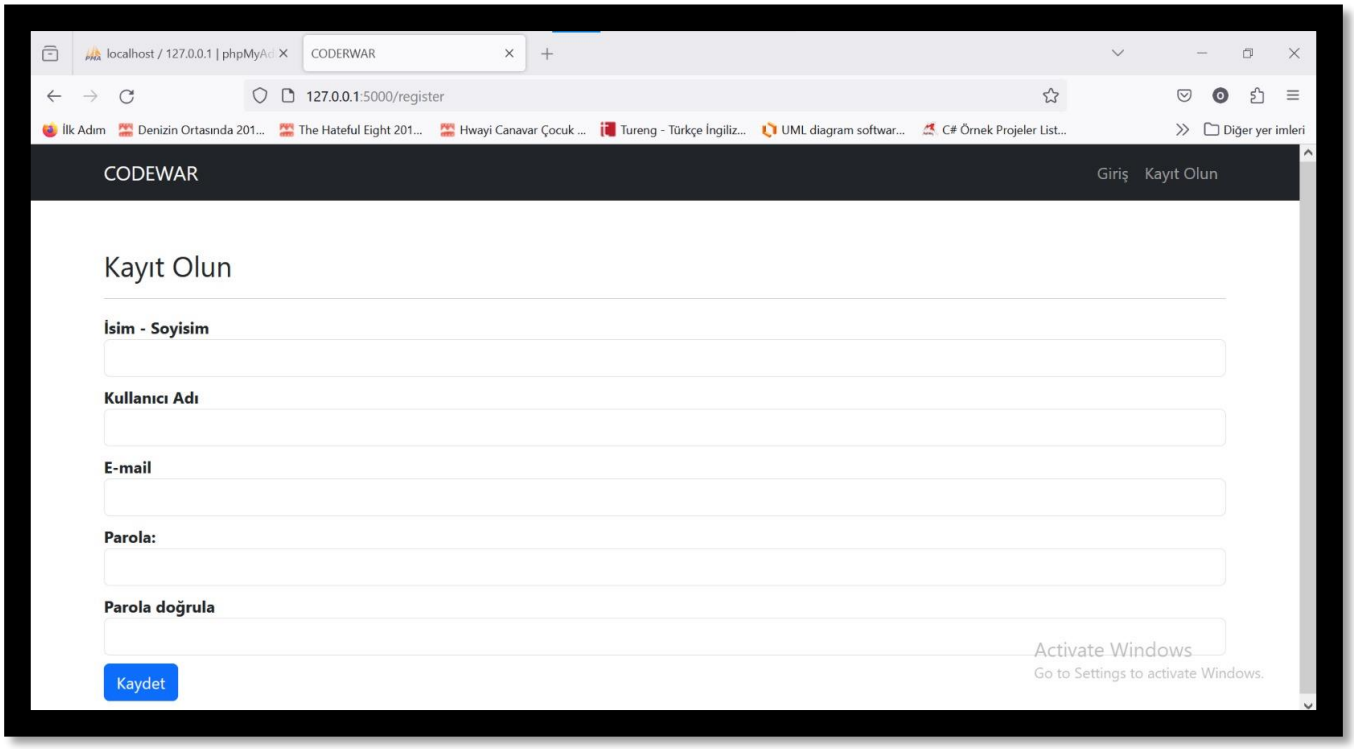
ANA EKRAN



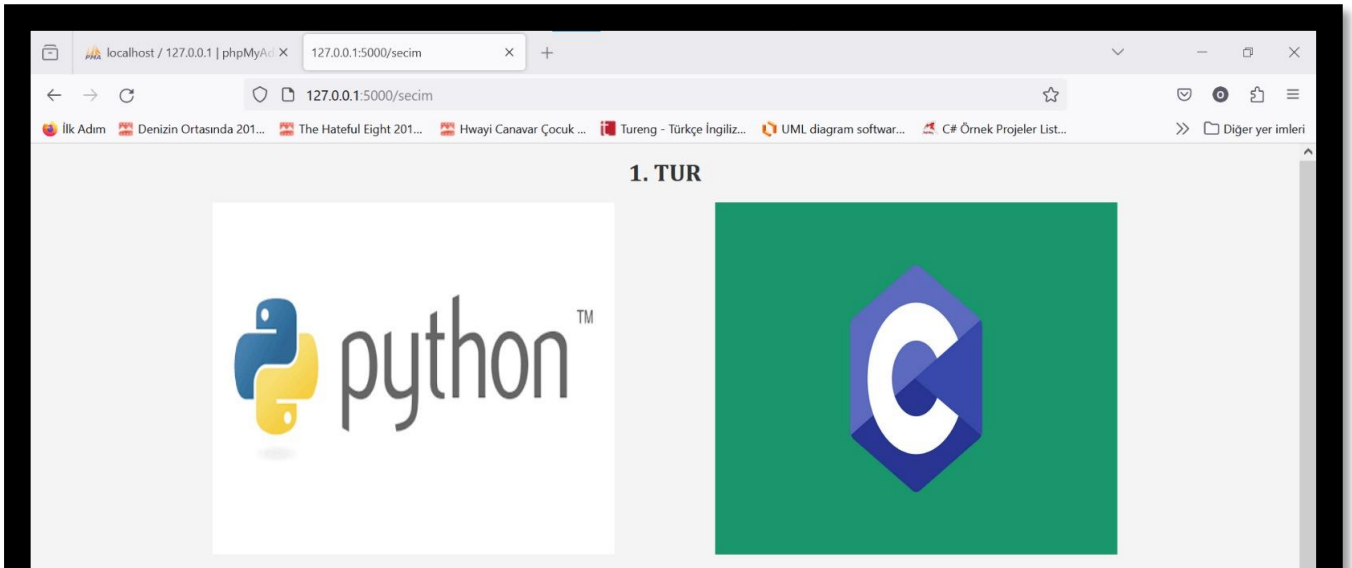
GİRİŞ EKRANI



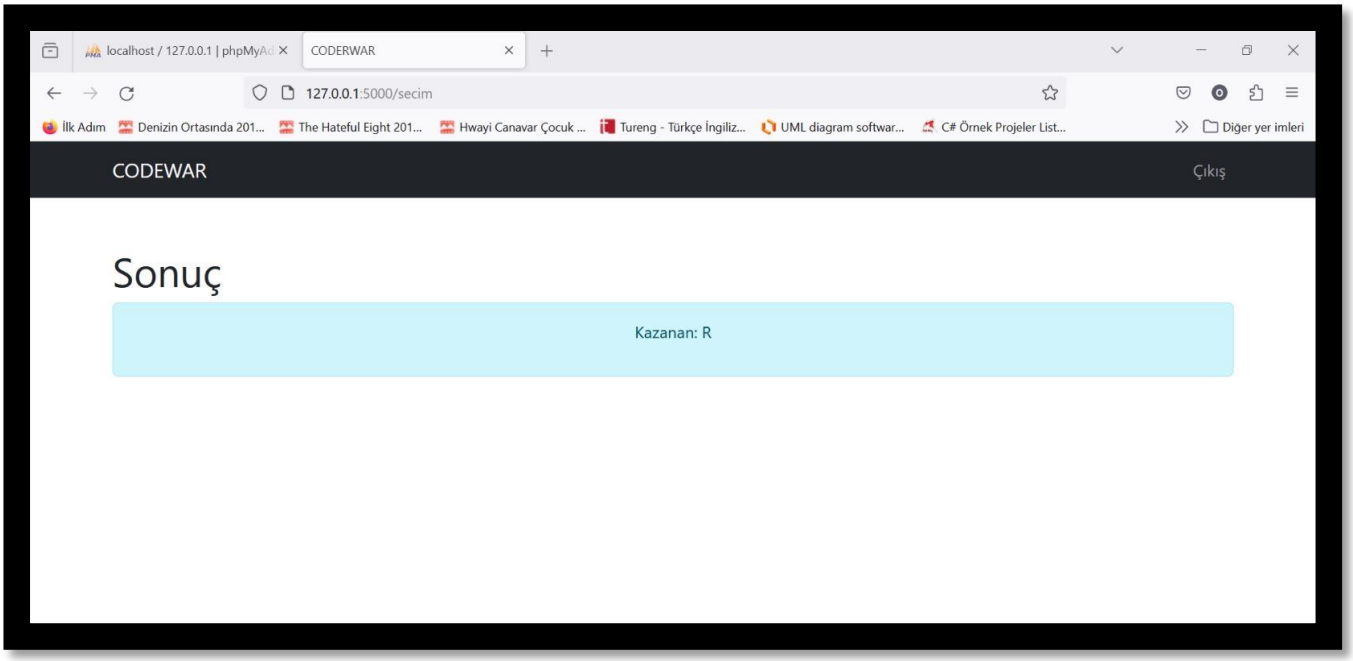
KAYIT EKRANI



TEST EKRANI



SONUÇ EKRANI



KULLANILAN VERİTABANI

Fazladan seçenekler

	id	name	username	email	password
Düzenle Kopyala Sil	1	zeynep	zeynep	ozcelikzeynep03@gmail.com	5rounds=535000$vPRhmWYBZjuH32gw$RbjEXPEbfqvLTWDp...
Düzenle Kopyala Sil	2	berna	berna	berna@gmail.com	5rounds=535000$p59ZcfCEbwYYcxiR$yoVgpsB135lAkpbv...
Düzenle Kopyala Sil	3	İrembengü	irembengü	iremb3@gmail.com	5rounds=535000$BCW3NTBJCNLbH8oY$uwgDhOKYyonizfO....
Düzenle Kopyala Sil	4	oyaılgın	oyaılgın	oya@gmail.com	5rounds=535000$8g.qB4JUtncnkHA1$a5dpAVFKjihZCk.r...

Tümünü işaretle Seçilileri: Düzenle Kopyala Sil Dışa aktar