

به نام خدا

استاد : امیر کیوان شفیعی

نویسنده : مهسا نادری پور

۹۹۱۲۸۱۴۰۵۲

تمرین ۳ درس مهندسی نرم افزار :

طراحی یک وب سایت کاربر محور با استفاده از سیستم احراز هویت گوگل و اپل ، استفاده از API های سایت خبری برای نشان دادن اخبار درون وب سایت؛ اشتراک و دسترسی به شبکه های اجتماعی.

عنوان ها

۳	مقدمه :
۴	نصب و راه اندازی ماژول های مورد نیاز برنامه :
۶	توضیحات برنامه نویسی :
۶	__init.py__
۷	راه اندازی پایگاه داده:
۷	تابع بررسی احراز هویت:
۸	تابع create_app:
۸	تنظیمات احراز هویت:
۸	تابع create_database:
۹	App.py
۹	از ماژول __init.py__ وارد کردن تابع create_app:
۱۰	ساخت اپلیکیشن با استفاده از تابع create_app:
۱۰	بخش اصلی برای اجرای اپلیکیشن:
۱۱	Models.py
۱۲	views.py
۱۲	@login_required:
۱۳	auth.py
۱۷	ورود به سیستم از طریق ایمیل و رمز عبور:
۱۷	ورود به سیستم با گوگل:
۱۸	ورود با اپل:
۲۰	خروج از سیستم:
۲۰	استفاده از API سایت خبری :
۲۰	استفاده از موتور قالب بندی jinja2:
۲۱	طراحی رابط سمت کاربر:
۲۳	Log in
۲۳	Sign up
۲۴	Log in by Google Account
۲۴	Log in by Apple id
۲۵	Home and News
۲۸	About Us

مقدمه :

در این روزگار پیشرفته از نظر فناوری اطلاعات، وبسایتها و اپلیکیشنها تنها پلتفرمهایی نیستند که محتوا را ارائه می‌دهند؛ آنها دروازه‌هایی به دنیای دیجیتالی هستند که تعامل کاربران با اطلاعات و یکدیگر را شکل می‌دهند. به دنبال همین اهداف، برای طراحی یک وبسایت کاربرمحور با استفاده از سیستم‌های احراز هویت گوگل و اپل و همچنین API های سایت‌های خبری را برای نمایش اخبار به‌روز برای طراحی وبسایت استفاده کردیم، ادغام این قابلیت‌ها با قابلیت اشتراک‌گذاری محتوا در شبکه‌های اجتماعی، بستری مناسب برای کسب‌وکارها، رسانه‌ها و سرویس‌دهندگان محتوا می‌تواند فراهم کند .

برای طراحی وبسایت از زبان python و ماژول Flask در قسمت سمت سرور (بک‌اند) استفاده گردید و برای طراحی سمت کاربر (فرانت‌اند) از زبان‌های نشانه‌گذاری HTML و CSS و برای پویا سازی و تعامل با کاربر از JAVA SCRIPTS استفاده شده است .

همچنین از موتور قالب بندی jinja2 برای ارث‌بری از الگوها استفاده گردیده است .

در این تمرین سعی براین بوده است که وبسایت طراحی شده برای تمامی دستگاه‌ها و افراد که از آن استفاده می‌کنند جذاب باشد و رابط کاربری پویا و جذابی داشته باشد. در قسمت‌های بعدی به برنامه و روش نوشتن آن می‌پردازیم.

نصب و راه اندازی ماژول های مورد نیاز برنامه :

جهت برنامه نویسی قسمت بک اند این وب سایت از ماژول های متفاوتی استفاده گردیده است که مهمترین های آن ها ماژول Flask ، flask-sqlalchemy ، flask_login ، google.oauth و ... می باشد که در پایین لیست و روش وارد کردن آن ها آمده است .

در **auth.py** که مسئول وارد شدن و احراز هویت می باشد و توابع login دستی ، login با گوگل ، login به وسیله اپل آیدی و ثبت نام و خروج در آن قرار دارد این ماژول ها شامل :

```
import os
import pathlib
import requests
from flask import Blueprint, render_template, flash, url_for, session, abort,
redirect, request, Flask
from models import User
from werkzeug.security import generate_password_hash, check_password_hash
from __init__ import db
from flask_login import login_user, login_required, logout_user, current_user
from google.oauth2 import id_token
from google_auth_oauthlib.flow import Flow
import google.auth.transport.requests
import random
import string
from pip._vendor import cachecontrol
from flask_migrate import Migrate
import jwt
from time import time
import finnhub
```

می باشد .

در **views.py** که **route** ها و توابع که برای کاربر به نمایش می آید را نشان می دهد
ماژول های مورد استفاده شامل :

```
from flask import Blueprint, render_template, request, flash, jsonify
from flask_login import login_required, current_user
from __init__ import db
import json
import finnhub
```

می باشد.

در **__init__.py** که دیتابیس ها و توابع در آن قرار دارد ماژول های مورد استفاده
این ماژول ها می باشد:

```
from flask import Flask, session, abort
from flask_sqlalchemy import SQLAlchemy
from os import path
from flask_login import LoginManager
import os
```

در **models.py** که کلاس های مدل در آن تعریف می شود شامل این ها می باشد:

```
from __init__ import db
from flask_login import UserMixin
```

در فایل **app.py** که مسئول اجرای این برنامه و راه اندازی وب می باشد ماژولی
فراخوانی نشده است زیرا توسط دیگر بخش های برنامه که در قبل ذکر شده است این
عملیات صورت می گیرد.

توضیحات برنامه نویسی :

در این قسمت درمورد بخش های مختلف برنامه صحبت می کنیم و درباره کد های آن توضیح می دهیم .

لازم به ذکر است که محیط توسعه استفاده شده برای طراحی وب سایت در قسمت های فرانت اند و بک اند ، pycharm بوده است.

__init.py__

ماژول `__init.py__` به عنوان نقطه آغازین مورد استفاده قرار می گیرد که به مدیریت پایگاه داده، کنترل ورود کاربران و بررسی وضعیت احراز هویت کاربران می پردازد. این قسمت از کد بعد از وارد کردن ماژول های برنامه در زیر قرار داده شده است که به تنظیم و مدیریت دیتابیس ، وارد شدن کاربران و تابع `create_app` می پردازد.

```
db = SQLAlchemy()
DB_NAME = "mahsa.db"

def login_is_required(function):
    def wrapper(*args, **kwargs):
        if "google_id" not in session:
            return abort(401)
        else:
            return function()

    return wrapper

def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = 'hjshjhdjah kjshkjdhjs'
    app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:/// {DB_NAME}'
    os.environ["OAUTHLIB_INSECURE_TRANSPORT"] = "1"
    db.init_app(app)

    from views import views
```

```

from auth import auth

app.register_blueprint(views, url_prefix='/')
app.register_blueprint(auth, url_prefix='/')

from models import User

with app.app_context():
    db.create_all()

login_manager = LoginManager()
login_manager.login_view = 'auth.login'
login_manager.init_app(app)

@login_manager.user_loader
def load_user(id):
    return User.query.get(int(id))

return app

def create_database(app):
    if not path.exists('website/' + DB_NAME):
        db.create_all(app=app)
        print('Created Database!')

```

راهاندازی پایگاه داده:

`db = SQLAlchemy()` یک نمونه از کلاس `SQLAlchemy` ساخته شده که برای کار با پایگاه داده استفاده می‌شود.

`DB_NAME = "mahsa.db"`: نام پایگاه داده تعیین شده است.

تابع بررسی احراز هویت:

`login_is_required`: یک دکوراتور است که برای تعیین اینکه آیا کاربر وارد شده (logged-in) است یا خیر قبل از اجازه دسترب به بعضی مسیرها بکار می‌رود.

تابع `create_app`:

این تابع یک نمونه از کلاس Flask می‌سازد، تنظیمات مرتبط با اپلیکیشن و پایگاه داده را پیاده‌سازی می‌کند، و چارچوب‌های blueprints وارد شده را ثبت می‌کند که برای تقسیم پروژه به چندین بخش در فلسک استفاده می‌شوند.

استفاده از `app.app_context` اجازه می‌دهد که دستور `db.create_all` پایگاه داده‌ها و جداول لازم برای اپلیکیشن را بسازد حتی اگر در زمان اجرای تابع، اپلیکیشن هنوز شروع به کار نکرده باشد.

تنظیمات احراز هویت:

`login_manager = LoginManager()` اینستنس `LoginManager` برای مدیریت فرآیندهای ورود و خروج کاربران ساخته می‌شود.

`login_manager.user_loader`: تابع مورد استفاده برای بارگذاری اطلاعات کاربر از پایگاه داده وقتی که کاربر وارد می‌شود.

تابع `create_database`:

این تابع برای بررسی وجود پایگاه داده و در صورت نیاز، ساخت آن استفاده می‌شود.

App.py

این فایل اصلی برنامه است و وظیفه اجرای برنامه را دارد که در زیر برنامه این فایل قرار داده شده است .

```
from __init__ import create_app  
  
app = create_app()  
  
if __name__ == '__main__':  
    app.run(debug=True)
```

این کد یک نمونه اساسی برای شروع یک برنامه وب با استفاده از فریم‌ورک فلسک (Flask) است. در اینجا، به تفسیر بخش‌های مختلف این کد می‌پردازیم :

از ماژول `__init__.py` وارد کردن تابع `create_app`:

`from __init__ import create_app`

این خط از کد، تابع `create_app` را از ماژول `__init__.py` وارد (import) می‌کند. در پروژه‌های فلسک، ماژول `__init__.py` به عنوان نقطه آغازین مورد استفاده قرار می‌گیرد که تنظیمات اولیه اپلیکیشن و ایجاد نمونه‌های اپلیکیشن فلسک را مدیریت می‌کند. تابع `create_app` برای تنظیم کردن تنظیمات اپلیکیشن و اولیه‌سازی

اکستنشن‌ها (extensions)، مدل‌های دیتابیس، مسیرها (routes)، و سایر منابع مورد نیاز اپلیکیشن است. که در قسمت قبل درمورد فایل `__init__.py` توضیحات را ارائه دادیم و در مورد کارکرد آن نیز صحبت کردیم.

ساخت اپلیکیشن با استفاده از تابع `create_app`:

```
app = create_app()
```

این خط یک نمونه جدید از یک اپلیکیشن فلسک می‌سازد با استفاده از تابع `create_app`، این به معنای این است که تمام تنظیمات و تعریف‌های لازم برای اپلیکیشن ما قبل از شروع سرور اعمال شده‌اند. که این تنظیمات در فایل‌های دیگر برنامه قرار دارد.

بخش اصلی برای اجرای اپلیکیشن:

در این بخش، `if __name__ == '__main__':` اطمینان می‌دهد که بلوک کد زیر تنها زمانی اجرا می‌شود که اسکریپت مستقیماً اجرا گردد به عنوان مثال، به وسیله اجرای

python app.py در ترمینال بجای وارد شدن به عنوان یک ماژول .
app.run(debug=True) سرور توسعه‌ای فلسک را راه‌اندازی می‌کند. .
debug=True گزینه‌ای است که به فلسک اجازه می‌دهد تغییرات کد را دنبال کرده
و در صورت وقوع خطا، پیغام خطا نمایش دهد.

مجموعه این تکه‌های کد، زیربنای ساخت وب اپلیکیشن با فلسک را فراهم می‌کند و به
عنوان سنگ بنایی برای توسعه بیشتر اپلیکیشن می‌باشد.

Models.py

این بخش از کد تعریف مدل User در یک برنامه وب فلسک است که از پایگاه داده
SQL و flask_login برای مدیریت جلسات کاربری استفاده می‌کند.

```
class User(db.Model, UserMixin):  
    id = db.Column(db.Integer, primary_key=True)  
    email = db.Column(db.String(150), unique=True)  
    password = db.Column(db.String(150))  
    first name = db.Column(db.String(150))
```

این کلاس یک پایگاه داده کاربر را تعریف می‌کند که از هر دو که نشان‌دهنده یک
جدول پایگاه داده در SQLAlchemy است و UserMixin ارث‌بری می‌کند تا از
قابلیت‌هایی که flask_login برای مدیریت جلسه کاربری فراهم می‌کند، استفاده
نماید.

views.py

این قسمت از کد به ساخت و عملکرد بخش نمایش‌ها (views) در فلسک می‌پردازد. این نمایش‌ها مسیرهای مختلفی را برای کاربران تعریف می‌کنند که از طریق آن‌ها می‌توانند به صفحات مختلف وبسایت دسترسی پیدا کنند.

```
views = Blueprint('views', __name__)

finnhub_client =
finnhub.Client(api_key="cmd9ra1r01qip5t7i7o0cmd9ra1r01qip5t7i7og")

@views.route("/about")
def about():
    return render_template("about.html", user=current_user)

@views.route('/', methods=['GET', 'POST'])
@login_required
def home():
    try:
        news = finnhub_client.general_news('general', min_id=0)
        return render_template('home.html', news=news, user=current_user)
    except Exception as e:
        return render_template('home.html', user=current_user)
```

@login_required: این دکوراتور قبل از تابع home استفاده شده و نشان می‌دهد که کاربر باید قبل از دسترسی به مسیر اصلی وارد سیستم شده باشد. اگر کاربر وارد نشده بود، کاربر به صفحه ورود به سیستم ری‌دایرکت می‌شود.

auth.py

این کد یک بخش از سیستم احراز هویت برای اپلیکیشن فلسک که نوشته ایم می باشد که قابلیت های ورود به سیستم از طریق ایمیل/رمز عبور معمولی، گوگل و اپل را فراهم می آورد. این کد شامل موارد زیر است:

```
app = Flask("__name__")

auth = Blueprint('auth', __name__)
app.secret_key = 'ijhuljhu hukhouhku'
os.environ["OAUTHLIB_INSECURE_TRANSPORT"] = "1"
migrate = Migrate(app, db)

client_secrets_file = os.path.join(pathlib.Path(__file__).parent,
"client_secret.json")
finnhub_client =
finnhub.Client(api_key="cmd9ra1r01qip5t7i7o0cmd9ra1r01qip5t7i7og")

GOOGLE_CLIENT_ID = "724949503986-
fo1k68hojv53c88rprpk0f1fcgkq098p.apps.googleusercontent.com"
GOOGLE_CLIENT_SECRET = "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"

flow = Flow.from_client_secrets_file(
    client_secrets_file=client_secrets_file,
    scopes=["https://www.googleapis.com/auth/userinfo.profile",
"https://www.googleapis.com/auth/userinfo.email", "openid"],
    redirect_uri="http://127.0.0.1:5000/callback",
)

def login_is_required(function):
    def wrapper(*args, **kwargs):
        if "google_id" not in session:
            return abort(401) # Authorization required
        else:
            return function()

    return wrapper

@auth.route('/login', methods=['GET', 'POST'])
def login():

    if request.method == 'POST':
```

```

email = request.form.get('email')
password = request.form.get('password')

user = User.query.filter_by(email=email).first()
if user:
    if check_password_hash(user.password, password):
        flash('Logged in successfully!', category='success')
        login_user(user, remember=True)
        return redirect(url_for('views.home'))
    else:
        flash('Incorrect password, try again.', category='error')
else:
    flash('Email does not exist.', category='error')

return render_template("login.html", user=current_user)

@auth.route('/sign-up', methods=['GET', 'POST'])
def sign_up():
    if request.method == 'POST':
        email = request.form.get('email')
        first_name = request.form.get('firstName')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')
        user = User.query.filter_by(email=email).first()
        if user:
            flash('Email already exists.', category='error')
        elif len(email) < 4:
            flash('Email must be greater than 3 characters.',
category='error')
        elif len(first_name) < 2:
            flash('First name must be greater than 1 character.',
category='error')
        elif password1 != password2:
            flash('Passwords don\'t match.', category='error')
        elif len(password1) < 7:
            flash('Password must be at least 7 characters.',
category='error')
        else:
            new_user = User(email=email, first_name=first_name,
password=generate_password_hash(
password1, method='pbkdf2:sha256'))
            db.session.add(new_user)
            db.session.commit()
            login_user(new_user, remember=True)
            flash('Account created!', category='success')
            return redirect(url_for('views.home'))

    return render_template("sign_up.html", user=current_user)

@auth.route('/login-by-google')
def login_bygoogle():

    authorization_url, state = flow.authorization_url()
    session["state"] = state

```

```

        return redirect(authorization_url)

@auth.route('/callback')
def callback():

    flow.fetch_token(authorization_response=request.url)

    if not session["state"] == request.args["state"]:
        abort(500)

    credentials = flow.credentials
    request_session = requests.session()
    cached_session = cachecontrol.CacheControl(request_session)
    token_request =
google.auth.transport.requests.Request(session=cached_session)

    id_info = id_token.verify_oauth2_token(
        id_token=credentials._id_token,
        request=token_request,
        audience=GOOGLE_CLIENT_ID

    )
    session["google_id"] = id_info.get("sub")
    session["name"] = id_info.get("name")
    session["email"] = id_info.get("email")
    user = User.query.filter_by(email=session["email"]).first()
    if not user:
        plain_password = ''.join(random.choice(string.ascii_letters) for i in
range(10))
        hashed_password = generate_password_hash(plain_password,
method='pbkdf2:sha256')

        new_user = User(
            email=session["email"],
            first_name=session["name"],
            password=hashed_password
        )
        db.session.add(new_user)
        db.session.commit()
        user = new_user
    flash('Logged in successfully!', category='success')
    login_user(user, remember=True)
    return redirect(url_for('views.home'))

```

```

@auth.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('auth.login'))

@auth.route('/login-by-apple')
def login_by_apple():
    authorize_url = 'https://appleid.apple.com/auth/authorize'
    token_url = 'https://appleid.apple.com/auth/token'

    client_id = 'YOUR.APPLE.CLIENT.ID'
    redirect_uri = 'YOUR REDIRECT URI AFTER SUCCESSFUL LOGIN'

    state = ''.join(random.choice(string.ascii_letters) for i in range(10))
    session['state'] = state

    request_uri =
f'{authorize_url}?response_type=code&client_id={client_id}&redirect_uri={redi
rect_uri}&state={state}'

    return redirect(request_uri)

@auth.route('/callback-from-apple')
def callback_from_apple():
    code = request.args.get('code')
    state = request.args.get('state')

    if state != session.pop('state', None):
        abort(403)

    token_payload = {
        'iss': 'YOUR APPLE TEAM ID',
        'iat': time(),
        'exp': time() + 3600,
        'aud': 'https://appleid.apple.com',
        'sub': 12554884565654,
    }

    client_secret = jwt.encode(
        token_payload,
        'YOUR PRIVATE KEY FROM APPLE',
        algorithm='ES256',
        headers={
            'kid': 'edsgwr44wtwfcxxaerer-cafdeff',
        }
    ).decode('utf-8')

    return redirect(url_for('views.home'))

```


ورود به سیستم از طریق ایمیل و رمز عبور:

روت‌های `/login` و `/sign-up` به کاربران اجازه می‌دهند که به وب اپلیکیشن ورود کرده و ثبت نام کنند. این توابع از `form` های `HTML` برای گرفتن اطلاعات کاربر استفاده می‌کنند و با استفاده از `User.query.filter_by` که از `ORM` `SQLAlchemy` استفاده می‌کند، کاربران را در بانک اطلاعات جستجو می‌کنند.

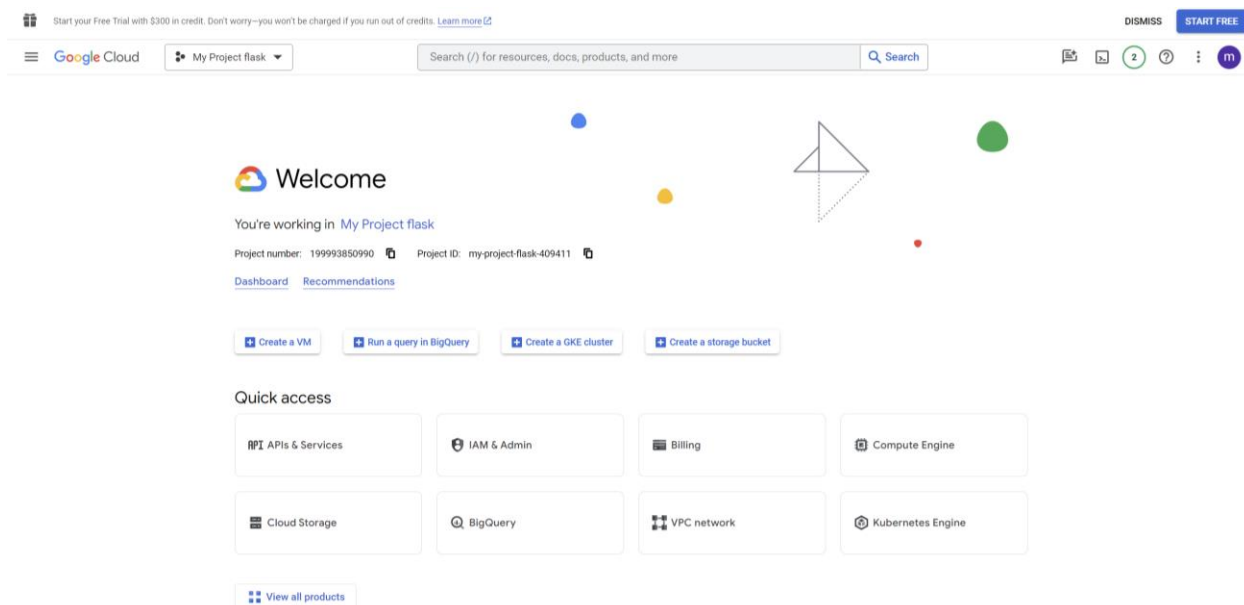
این رمزهای عبور با قسمت های مختلفی مانند `check_password_hash` و `generate_password_hash` مواجه می‌شوند تا اطمینان حاصل شود که ذخیره‌سازی و تأیید به شیوه‌ای امن انجام می‌شود.

ورود به سیستم با گوگل:

روت‌های `/login-by-google` و `/callback` امکان ورود از طریق حساب کاربری گوگل را فراهم می‌آورند.

`Flow.from_client_secrets_file` برای ایجاد و مدیریت یک جریان `OAuth2` برای احراز هویت گوگل استفاده شده است. کاربران به صفحه ورود گوگل هدایت شده و پس از ورود، درخواست حاوی یک `callback` به سرور فلکس برگردانده می‌شود، که توکن‌ها را تبادل می‌کند و اطلاعات کاربر را تأیید می‌کند.

در زمینه ورود و اعتبار سنجی توسط گوگل نیاز داریم تا ابتدا یک اکانت در گوگل کلود داشته باشیم. پس از ثبت نام یک توکن و کد اعتبار سنجی برای برقراری ارتباط با گوگل و استفاده از آن در پروژه بدست می‌آوریم که در قسمتی که قبلاً ذکر کردیم قرار داده می‌شود.



ورود با اپل:

روت‌های `/login-by-apple` و `/callback-from-apple` سناریوی ورود به سیستم از طریق حساب اپل را شروع می‌کنند. این بخش نمونه‌ای است و نیاز به کامل‌تر کردن اطلاعات مثل `'YOUR.APPLE.CLIENT.ID'` و دیگر متغیرهایی که مربوط به اپل هستند دارد. همچنین برای ایجاد `client_secret` از کد رمزنگاری JWT استفاده می‌شود.

بدست آوردن توکن و API اپل نیز تقریباً مانند گوگل می‌باشد با این تفاوت که در اپل نیاز داریم به اپل ایدی و اپل دولوپر.

Apple ID

Sign InCreate Your Apple IDFAQ

Create Your Apple ID

One Apple ID is all you need to access all Apple services.

First name

Last name

COUNTRY / REGION

United States

Birthday

name@example.com

This will be your new Apple ID.

Password

Confirm password

+1 (United States)

نکته مهم :

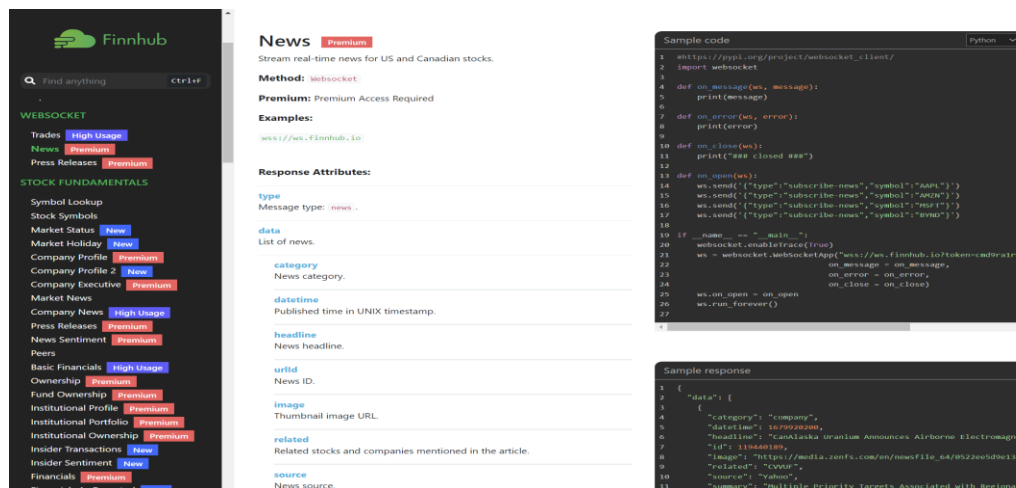
باید توجه داشته باشیم که برخی از داده‌های حساس مانند 'SECRET_KEY' ، 'GOOGLE_CLIENT_ID' ، 'GOOGLE_CLIENT_SECRET' و API کلیدها بایستی به صورت محافظت شده نگهداری شوند. اما چون این یک تمرین می باشد به متغیر های محیطی virtualenviorment اکتفا کردم.

خروج از سیستم:

روت `/logout` امکان خروج از حساب کاربری را می‌دهد و تابع `logout_user` را فراخوانی می‌کند که از کتابخانه `flask_login` استفاده می‌کند.

استفاده از API سایت خبری :

برای بهره بردن از `api news` سایت `finnhub` استفاده کردیم و یک توکن رایگان دریافت کردیم و `api` مورد نظر را سرچ و در قسمتی که قبلاً توضیح دادیم و مربوط به `API` بود وارد کردیم.



The screenshot displays the Finnhub website interface. On the left, there's a sidebar with navigation links like 'Trades', 'News', 'Press Releases', 'STOCK FUNDAMENTALS', 'Symbol Lookup', 'Market Status', 'Company Profile', 'Company Executive', 'Market News', 'Company News', 'Press Releases', 'News Sentiment', 'Peers', 'Basic Financials', 'Ownership', 'Fund Ownership', 'Institutional Profile', 'Institutional Portfolio', 'Institutional Ownership', 'Insider Transactions', 'Insider Sentiment', 'Financials', and 'Financials As Reported'. The main content area is titled 'News' and includes a search bar, a 'Method: websocket' dropdown, a 'Premium: Premium Access Required' notice, and an 'Examples' section with a URL `ws://ws.finnhub.io`. Below this, 'Response Attributes' are listed, including 'type', 'data', 'category', 'datetime', 'headline', 'urlid', 'image', 'related', and 'source'. To the right, a 'Sample code' block shows a Python script using the `websocket` library to connect to the Finnhub API and subscribe to news. Below the code, a 'Sample response' block shows a JSON object representing the API response.

استفاده از موتور قالب بندی jinja2:

از موتور قالب بندی `JINJA2` برای قالب بندی و تعامل با سرور استفاده کردیم تا داده های پویا داشته باشیم و بتوانیم صفحات را برای هر کاربر متفاوت شخصی سازی کنیم. در ادامه یک مثال از استفاده `JINJA2` در وبسایتمان را قرار میدهیم.

```
<p>welcome, {{ user.first_name }} </p>
</div>
```

```

<div class="nav-items">
  <a href="/" class="nav-item">Home</a>
  <a href="/about" class="nav-item">About</a>
  <a href="/logout" class="nav-item">logout</a>
</div>
</nav>

<section class="landing">
  <h1></h1>
  <p></p>

  <h1> News</h1>
<ul>
  {% for article in news_articles %}
  <li>
    <h2>{{ article['headline'] }}</h2>
    
    <p>{{ article['summary'] }}</p>
    <a href="{{ article['url'] }}">Read more</a>
  </li>
  {% endfor %}
</ul>
</section>
</div>

```

در این مثال با استفاده از موتور قالب بندی JINJA2 در ابتدا بعد از welcome نام کاربر نمایش داده می شود ، همچنین در ادامه برای نمایش اخبار با استفاده از API آن ها را فراخوانی می کند و محل قرار گیری و متن های مربوطه را تنظیم می کند.

طراحی رابط سمت کاربر :

وب سایتی که طراحی کردیم از چند صفحه شامل صفحه ورود (login)، صفحه ثبت نام (signup) و صفحه درباره ما (about us) می شود.

هرکدام از این صفحات توسط زبان نشانه گذاری HTML طراحی گردیده و توسط CSS به آن ها استایل داده شده است تا توسط کاربران هنگام دیده شدن دارای جذابیت بصری بالاتری باشند و همچنین برای پویا سازی و تعامل بیشتر با کاربر از زبان برنامه نویسی javascripts استفاده شده است.

برای ریسپانسیو سازی وب سایت نیز سعی شد تا از وایر فریم هایی استفاده شود که مناسب استفاده در دستگاه های مختلف باشد و همچنین از media screen در CSS برای حالت دهی متفاوت در سایز های مختلف دستگاه ها در این طراحی استفاده گردید.


برای راحتی بیشتر و استفاده توسط تمام کاربران در سند های HTML ، alt تصاویر و type هر بخش و سکشن ها مشخص گردیده است.


در زیر صفحات طراحی شده قرار گرفته است .

Log in

G

Log in to Flask App

 Sign in with Google

 Sign in with Apple

or


Email

password

Next

Don't have an account? [Sign up](#)

Sign up



Sign Up Flask App

Email

First Name

Please fill out this field.

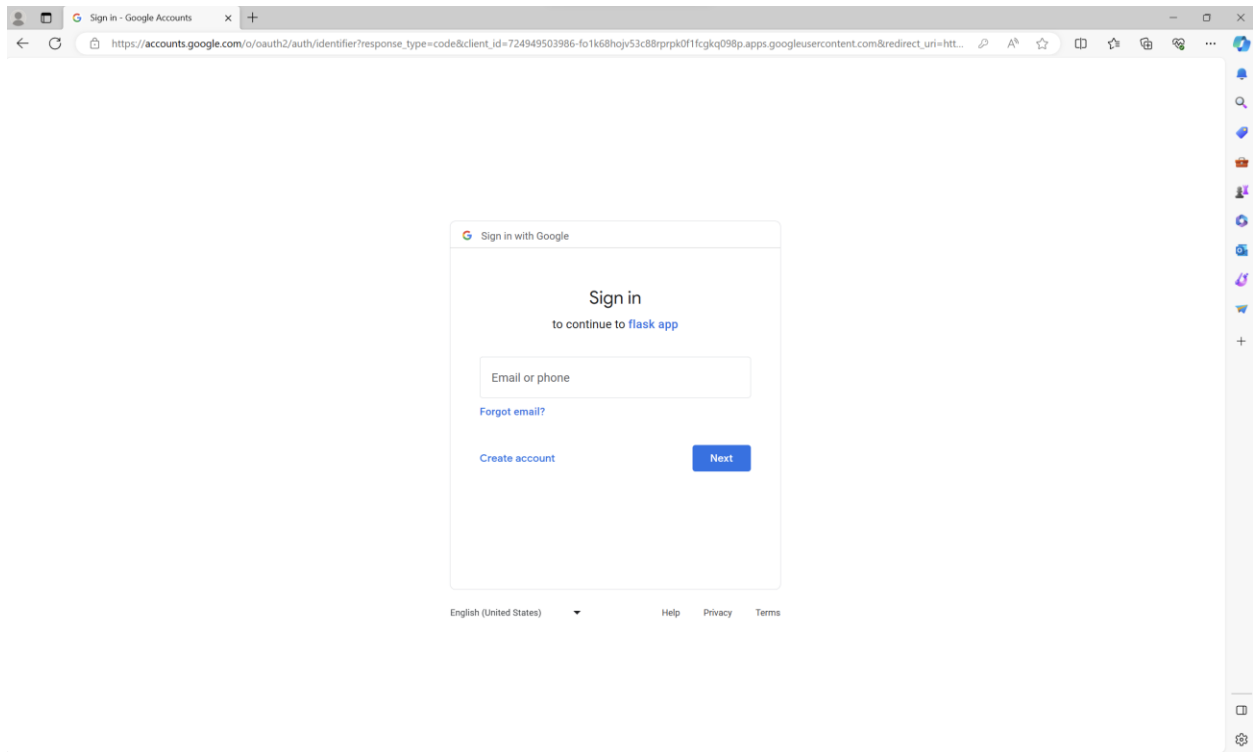
Password

Confirm Password

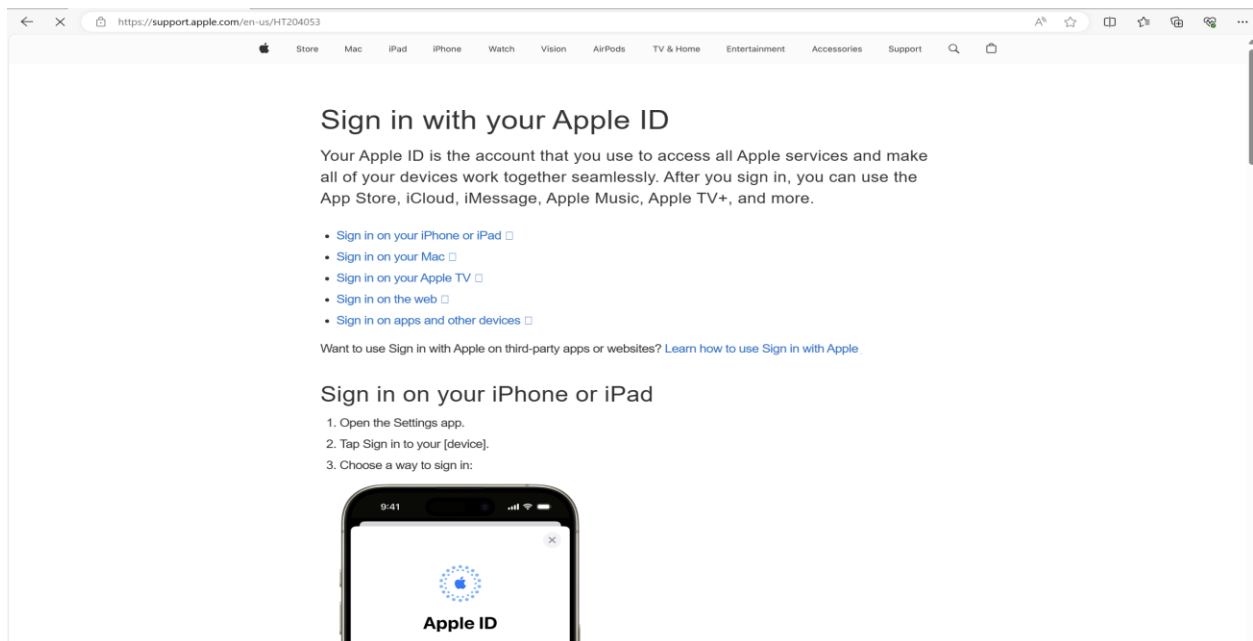
Sign Up

have an account? [log in](#)

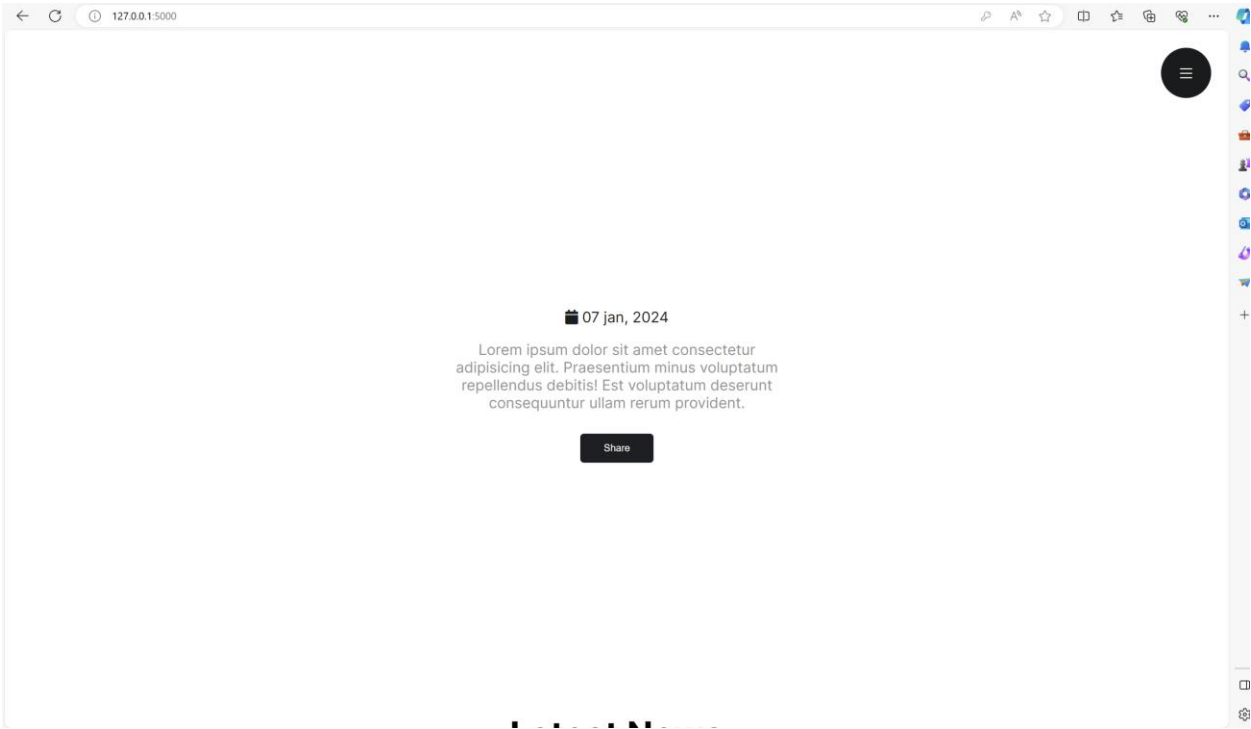
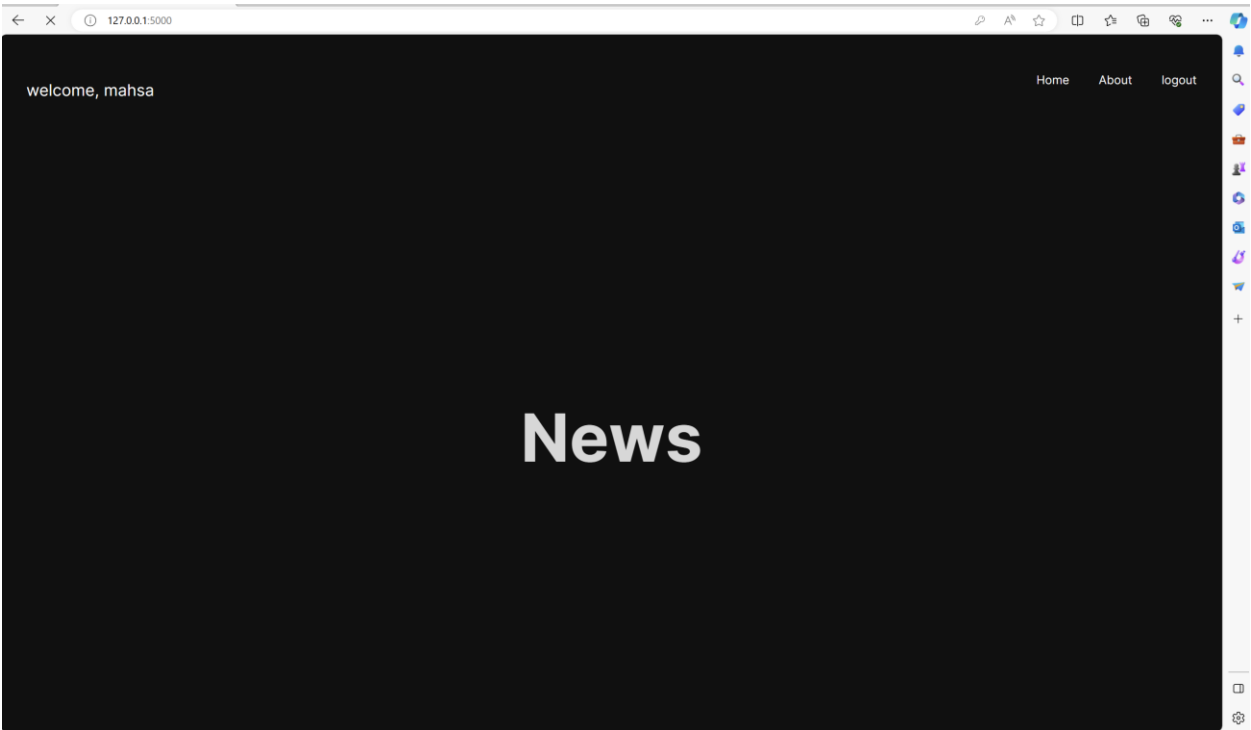
Log in by Google Account

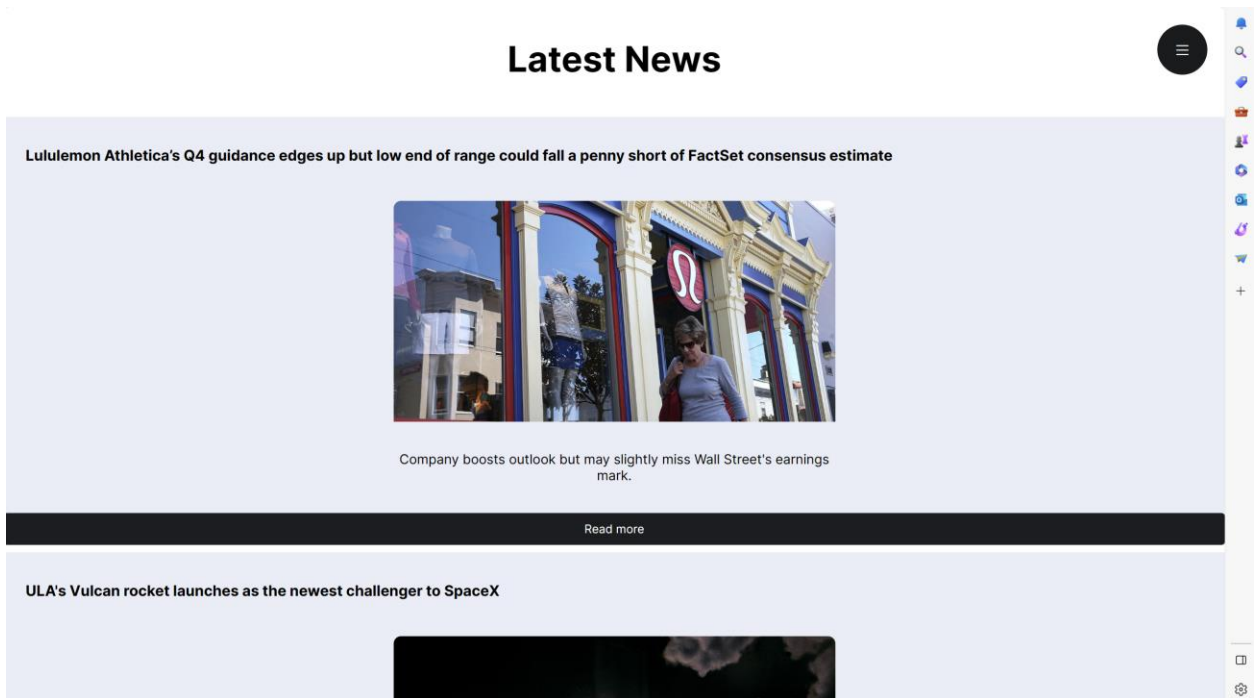
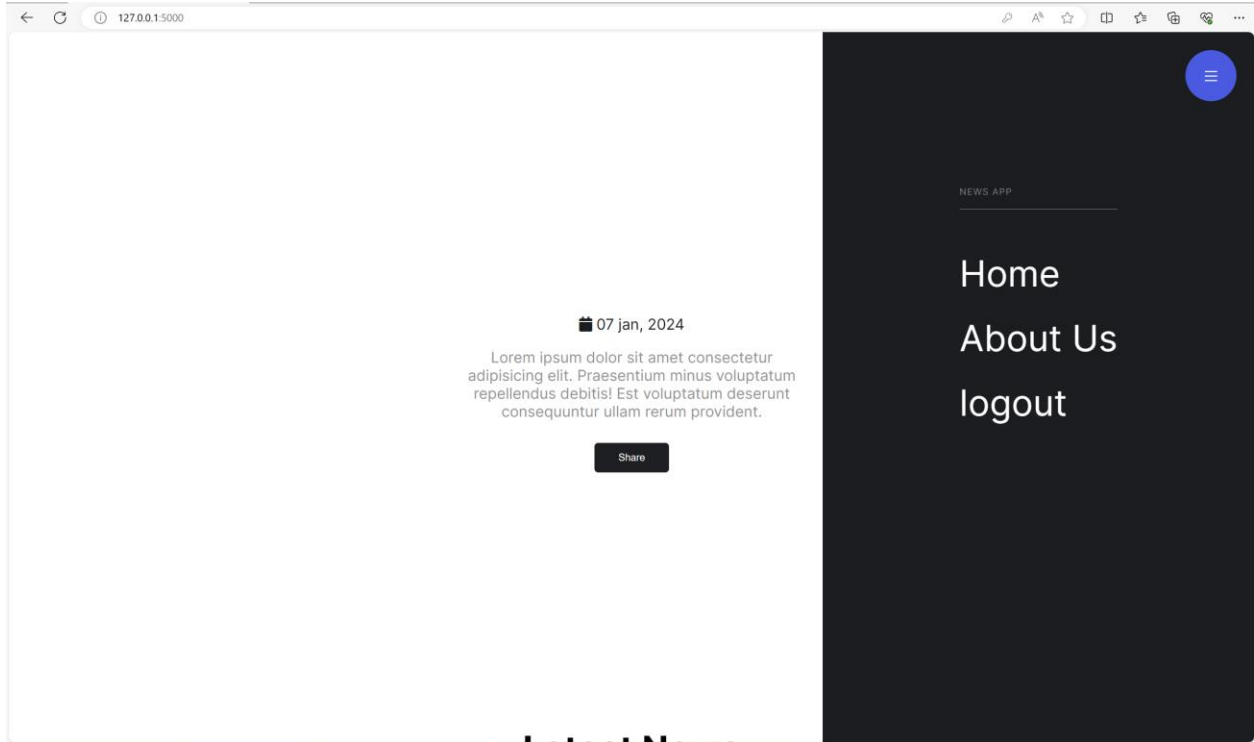


Log in by Apple id



Home and News







Retail investors are falling back into a bad habit of following a herd.

Read more

Oil prices slump as Saudis cut prices



Oil futures start the week with a slump.

← ⓘ 127.0.0.1:5000 🔍 🔊 ☆ 📄 ⋮

again.



Retail investors are falling back into a bad habit of following a herd.

Read more

About Us

