FLASK-FORMS AND USER INPUT

במסמך הבא נלמד איך לבנות מבנים וקלטים בעזרת flask. ניצור מבנה התחברות למערכת (log in) והרשמה (sign up).
אם אנחנו רוצים לבנות מבנים מהבסיס נראה שבקלות הדברים יכולים להסתבך, למשל נצטרך לוודא שמשתמש הכניס סיסמא נכונה, לבצע בדיקת תקפות לסוג הקלט של המשתמש וכו'.
למזלנו הנושא כל כך נפוץ שאין לנו צורך להמציא את הגלגל. ל-flask יש כמה הרחבות שיכולות לבצע לנו את כל העבודה הקשה בכמה שורות קוד.
אחד ההרחבות המפורסמות ביותר ב-flask נקראת wtforms ובה נשתמש.
לפני הכל נצטרך להתקין את הספרייה:

```
pip install flask-wtf
```

שם קליט סה"כ.

לאחר ההתקנה ניצור מסמך פייתון חדש בתיקייה flask_example ונקרא לו forms.py .

כשמדברים על form ב-html מתכוונים למבנה מיוחד עם כפתורים, תיבת טקסט וכו' , למשל מבנה של הרשמה לאתר או שאלון וכדו'. ב-flask בונים את המבנה בצורה דומה לבניית מסד נתונים עם sqlalchemy , כלומר נבנה מחלקה ייחודית עבור כל מבנה שיורשת מאובייקט שנקרא FlaskForm של הספרייה flask_wtf, ואח"כ המבנה יוושם על קוד ה-html.

למבנים יש כמה שדות שניתן ליישם דרך משתנים של הספרייה wtforms, למשל בשביל תיבת טקסט פשוטה נשתמש ב-wtforms.StringField() , שאחד הארגומנטים שלו הם שם השדה. למשל עבור תיבת טקסט לשם המשתמש נשתמש באובייקט StringField('username') וזה גם השם שייצג את השדה בקובץ ה-html ; חוץ מזה נרצה גם שיהיה לנו הגבלות לאותו שדה, לדוגמא לא נרצה שיהיה ניתן להכניס שם ארוך מידי, ולא נרצה לאפשר להכניס null. בשביל למנוע מצב כזה נשתמש ב-validators שהם ארגומנט נוסף שניתן להוסיף ליצירת האובייקט. הארגומנט אמור להגיע כרשימה של validators אפשריים, וניתן למצוא אותם ב- wtforms.validators . אם נמשיך את הדוגמא הקודמת נרצה למנוע מצב שהמשתמש יכניס שם ארוך מידי או שיכניס null נוכל להשתמש באובייקטים DataRequiered() ו-Length(min=..,max =...) מתוך ה-validators.

עוד validator שימושי הוא Email() שבודק שאכן הוכנסה כתובת מייל תקנית.

בשביל סיסמאות נשתמש בשדה מיוחד של wtforms שנקרא PasswordField() שעובד בדיוק כמו StringField רק במצב נסתר. כמו כן יכול להיות שנרצה ליצור אימות סיסמא, ובמקרה כזה נשתמש ב-validator מיוחד שנקרא EqualTo שמקבל את שם השדה שהוא אמור להיות דומה לו.

ולבסוף נרצה להוסיף גם כפתור submit, וגם לזה יש שדה מיוחד של wtforms שנקרא SubmitField('...') ומקבל את הטקסט שיוצג על הכפתור כארגומנט.

כמו כן יש שדות נוספים שיכול להיות שנצטרך בהמשך למשל BooleanField שיכול לשמש אותנו כדי לבדוק אם המשתמש כבר נרשם למערכת או שהוא מחובר עדיין וכו'.

דוגמא למבנה: ניצור את המבנה ההרשמה הבא שיכיל שם משתמש, אי-מייל, סיסמא, אימות סיסמא, והרשמה:

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, BooleanField
from wtforms.validators import DataRequired, Length, Email, EqualTo


class RegistrationForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired(), Length(min=2, max=20)])
```

```
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')
```

בצורה דומה ניצור מבנה כניסה למערכת (login) , והוא יהיה קצת שונה שכן לא נצטרך אימות סיסמא או שם משתמש, אבל נוסיף לו שדה שיגדיר האם לזכור את המשתמש בפעם הבאה שהוא נכנס או לא:

```
class LoginForm(FlaskForm):
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember = BooleanField('Remember Me')
    submit = SubmitField('Login')
```

כשמשתמשים במבנים צריך להוסיף לרוב secret key שנועד להגן על התוכן מפני התקפות חיצוניות.
לא כל-כך מסובך להגדיר secret_key נחזור לקובץ __init__.py ונוסיף עוד קינפיגורציה מעל או מתחת לקונפיגורציה SQLALCHEMY_DATABASE_URI שקוראים לה SECRET_KEY והערך שלה יהיה איזושהי מחרוזת של תווים לא מוגדרים.
כדי להשיג את הערך של המחרוזת הזאת ניכנס למצב האינטרקטיבי של פייתון ונשתמש במתודה token_hex() של הספרייה secrets עם הפרמטר 16 ונקבל מחרוזת של תווים הקסדסימלים אקראית באורך 32 תווים:

```
>>> import secrets
>>> secrets.token_hex(16)
'ecf6e975838a2f7bf3c5dbe7d55ebe5b'
```

ונוסיף את הקוד ל-__init__.py כך:

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SECRET_KEY'] = 'ecf6e975838a2f7bf3c5dbe7d55ebe5b'
app.config['SQLALCHEMY_DATABASE_URI']= 'sqlite:///site.db'
db = SQLAlchemy(app)

from flask_example import routes
```

משום שהניתוב מתרחש בקובץ routes.py ונצטרך להשתמש במבנים בניתוב לדפים, נייבא את המחלקות של המודול במסמך routes.py:

```
from flask import render_template
from flask_example import app
from flask_example.forms import RegistrationForm, LoginForm
from flask_example.models import User , Post
...
```

עכשיו צריך להוסיף ניתוב לדף הרשמה ולהעביר לו אינסטנס של המחלקה- RegistrationForm נעשה את זה ע"י יצירת פונקציית ניתוב חדשה במסמך routes.py ונעביר את האובייקט כערך חזרה מהפונקציה, וכנ"ל נעשה לדף הכניסה למערכת :

```
@app.route("/register")
def register():
    form = RegistrationForm()
    return render_template('register.html', title='Register', form=form)


@app.route("/login")
def login():
    form = LoginForm()
    return render_template('login.html', title='Login', form=form)
```

נשאר להגדיר את ה-templates של עמוד הרישום ועמוד התחברות.

אבל לפני זה בואו נעשה קצת סדר בקוד ונשנה כמה דברים, דבר ראשון בואו נוסיף סטייל נוסף לקוד כדי לשפר את הנראות שלו. אנחנו נשתמש בספרייה מהאתר bootstrap שמאפשרת להשתמש בספריות css שהם מציעים אונליין מבלי שנצטרך להוריד אותם , כל מה שצריך הוא להוסיף קוד לתחילת מסמך ה-layout.html: ובתוך התג <body>.

כמו כן הוספנו קובץ css קטן כדי להתאים את המראה של התוכן של הפוסטים להמשך. עוד שינוי שביצענו הוא ב-home.html כדי להתאים אותו לעיצוב החדש.

השינויים שהוספנו הם בעיקר קוסמטיים כדי לשפר את ניראות האתר. לנו חשוב יותר להציג את החלקים הקשורים לצד שרת ופחות לחלקים שקשורים ב-frontend. עבור אלה שחשוב להם לעבוד על הנראות של האתר נמליץ שוב להסתכל ב-w3schools בנושא css, ב-bootstrap או בכל מקום שמספק מידע שימושי על frontend.

ועכשיו החלק היותר חשוב בתהליך- כשאנחנו רוצים לנתב לקובץ מסויים דרך flask ,ולאו דווקא מסמך html, אנחנו יכולים להשתמש בפונקציה url_for() שמקבלת כארגומנט את ה-url אליו צריך לנתב ,או במקרה שלנו את שם התיקייה אליה צריך לנתב. משום שאנחנו מנתבים לקובץ main.css נכניס אותו לפונקציית הניתוב ע"י הפרמטר filename :

```
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='main.css') }}">
```

הקובץ אליו אנחנו מנתבים הוא main.css והוא בתיקייה statis.
את השינויים שביצענו ניתן לראות בתיקייה 5.forms <-code

נחזור לענייננו- עמוד רישום ועמוד התחברות. הטמפלייט של העמודים אמור להמשיך את הקובץ layout, ובתוך ה-block content אנחנו אמורים ליצור div חדש שהשם של המחלקה שלו הוא content-section כדי שיוכל לקבל את העיצוב של main.css. בתוך ה-div נצטרך ליצור מבנה form שהמתודה שלו היא POST, ואת ה-action נגדיר אח"כ:

```
{% extends "layout.html" %}
{% block content %}
    <div class="content-section">
        <form method="POST" action="">
            ...
```

יש כמה מתודות בקשה של http. אם נשים לב כשמקימים את השרת ונכנסים לכתובת האתר, נוצרת בקשת get של הלקוח לשרת. בקשת get היא קצרה, לא שוקלת הרבה ולרוב לא מוסתרת , לעומתה בקשת post היא בקשה לשרת שאין לה הגבלת כמות כלשהי לכאורה. אנחנו נשתמש ב-post כדי להעביר לשרת, בצורה בטוחה, מידע של הלקוח.

לtemplate אנחנו מעבירים אובייקט form שהוא אובייקט מטיפוס המחלקות שהגדרנו קודם, לכן יש לו גם מתודות של המחלקה, ומשום שהמחלקה יורשת מ-FlaskForm יש להן גם את המתודות שלה, למשל המתודה hidden_tag() שמאבטחת את תוכן המחלקה עם ה-SECRET_KEY שהגדרנו קודם לכן. ומעכשיו כל שדה שנשתמש בו ב-form יהיה בנוי בתוך div נפרד.

במסמך ההרשמה נצטרך בהתחלה תיבת טקסט למשתנה username של form, ובאמת כבר יש לנו תיבה כזאת שמגיעה עם האובייקט. כדי להציג את השדה username נשתמש במתודה label() של StringField (המשתנה username הוא מטיפוס StringField), שיכולה גם לקבל איזשהי הגדרת html, למשל אם נרצה לשמור על העיצוב של bootstrap נקרא למחלקה של המשתנה באיזשהו שם מוגדר מראש של אחד מקבצי ה-css של הספרייה, לדוגמא (' form-control-label = class) form-control-label . כמו כן נרצה שתוצג התיבת טקסט של ה-username, לכן נוסיף עוד פעם משתנה form.username וגם לו נגדיר איזושהי מחלקה מ-bootstrap, אבל הפעם לא נשתמש במתודה label.

```
{% extends "layout.html" %}
{% block content %}
    <div class="content-section">
        <form method="POST" action="">
          <div class="form-group">
           {{ form.username.label(class="form-control-label") }}
           {{ form.username(class="form-control form-control-lg") }}

         </div>
```

אותו דבר נעשה גם עבור שאר הפריטים של הפורום- email ו-password, ואישור סיסמא, רק שנשנה מ-username לשמות השדות שלהם, ולכפתור ה-submit בצורה דומה רק בלי ה-label:

```
<div class="form-group">
     {{ form.submit(class="btn btn-outline-info") }}
</div>
```

אם נריץ את השרת העמוד אמור לעבוד כמו שצריך, רק שעדיין לא ניתן להגיש את בקשת ה-post, וזה הדבר הבא שעליו נעבוד.
כדי לאפשר בקשת post נצטרך לחזור למסמך routes.py ולהוסיף לקשטן גם אילו מתודות ניתן לקבל חזרה מהעמוד. המתודות אמורות להגיע כרשימה של מחרוזות עם שם המתודה, אנחנו הולכים לאפשר מתודות GET ומתודות POST :

```
@app.route("/register" , methods = ['GET', 'POST'])
def register():
    form = RegistrationForm()
    return render_template('register.html', title='Register', form=form)
```

עכשיו לא נקבל שגיאה כשנלחץ עם sign up, אבל אנחנו לא יודעים אם השדות נשלחו כמו שצריך, לשם כך נשתמש במתודה validate_on_submit() שכפי ששמה רומז היא אומרת לנו אם לא קיבלנו שגיאה כשביצענו את שליחת הנתונים.

אם אכן הנתונים נשלחו כמו שצריך נרצה שתוצג הודעה חד פעמית למסך שאומרת שהפעולה הצליחה. נוכל להשתמש בפונקציה של flask שנקראת flash, בשביל להשתמש בה נצטרך כמובן לייבא אותה. הפונקציה יכולה לשלוח הודעה שתוצג על המסך באופן חד פעמי וניתן להגדיר לה את סוג ההודעה, אנחנו נשתמש בהודעה מסוג success , וכדי להציג את שם המשתמש שנבחר נשתמש במשתנה form.username.data . וכמובן שנרצה

לעבור לעמוד הבית שוב לאחר השליחה של ההודעה. גם בשביל זה נשתמש במתודה של flask שנקראת redirect והיא משתמשת בפונקציה **url_for** כדי להפעיל פונקציית ניתוב:

```
@app.route("/register" , methods = ['GET', 'POST'])
def register():
    form = RegistrationForm()
    if form.validate_on_submit():

        flash(f'Account created for {form.username.data}!', 'success')
        return redirect(url_for(home.__name__))
    return render_template('register.html', title='Register', form=form)
```

שימו לב שהפונקציה url_for מקבלת את השם של **הפונקציה** home, ולא של הדף. כך, אם נחליט בעתיד להעביר את הדף לכתובת אחרת, הכתובת בטופס תתעדכן אוטומטית. יכול להיות שבמסמכים הקודמים השתמשנו בשם hello_world() בכל מקרה הפונקציה שמנתבת לעמוד הראשי עכשיו היא הפונקציה home() .

כדי שנוכל לראות את השינויים בעמוד הראשי, נצטרך להוסיף כמה שורות קוד מעל המשתנה block content במסמך layout.html.

כאן נשתמש ב-context manager כדי להציג את השינוי. ב-flask על עמודי html הקוד נכתב קצת אחרת, ובמקום להגדיר as ... with ... כפי שאנחנו מכירים, ישר נגדיר משתנה עם אופרטור השמה לפונקציה שעליה מבצעים את ה-context manager (ראו דוגמא למטה).

כדי להציג את ההודעות שהועברו עם הפונקציה flash צריך לקרוא לפונקציה get_flashed_messages עם הפרמטר with_categories שמוגדר כ-true אם נרצה לקחת את הקטגוריה של הפונקציה, במקרה שלנו הקטגוריה היא 'success'. ועכשיו רק במקרה שקיבלנו הודעות flash (ולא None) נדפיס אותן עם הקטגוריות שלהן:

```
{% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
          {% for category, message in messages %}
            <div class="alert alert-{{ category }}">
              {{ message }}
            </div>
          {% endfor %}
        {% endif %}
      {% endwith %}
```

ושוב ה-class שבתוך ה-div הוא בסה"כ סוג של עיצוב מהאתר bootstrap, לא משתנה שהגדרנו קודם לכן.

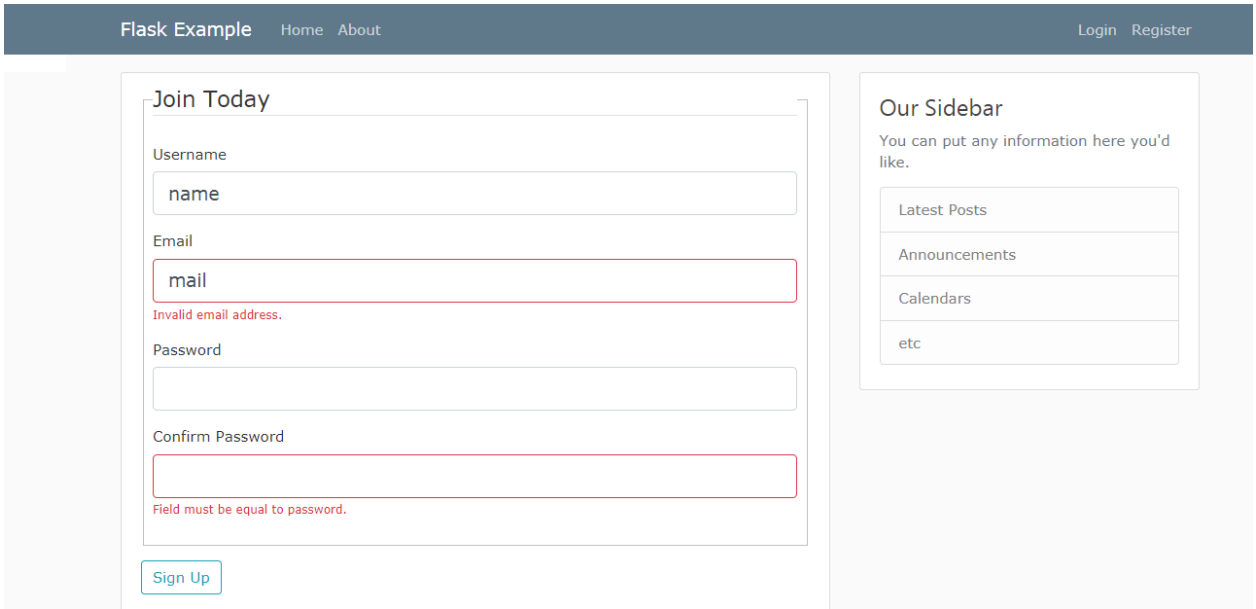
עדיין לא סיימנו עם עמוד הרישום, נשאר להציג פידבק אם היו שגיאות במהלך ההרשמה, למשל שמייל לא היה תקין, או שהסיסמא לא מספיק ארוכה, או שאחד השדות ריק וכו'. נחזור לדף register.html (דף ההרשמה). צריך לבדוק אם במהלך הכנסת הקלט למשתנה התקבלו שגיאות, השגיאות נשמרות בכל שדה במשתנה error שלו. במידה והיו כאלה נרצה להדפיס אותן ולשנות את תיבת הטקסט שתראה אדומה (כאילו קרתה שגיאה בתיבה). בשביל זה נוכל להשתמש ב-is-invalid של bootstrap כתוספת ל-class, ועבור הצגת השגיאות בכל שדה נוכל להשתמש ב- לדוגמא את השדה username נשנה לזה:

```
<div class="form-group">
     {{ form.username.label(class="form-control-label") }}
     {% if form.username.errors %}
```

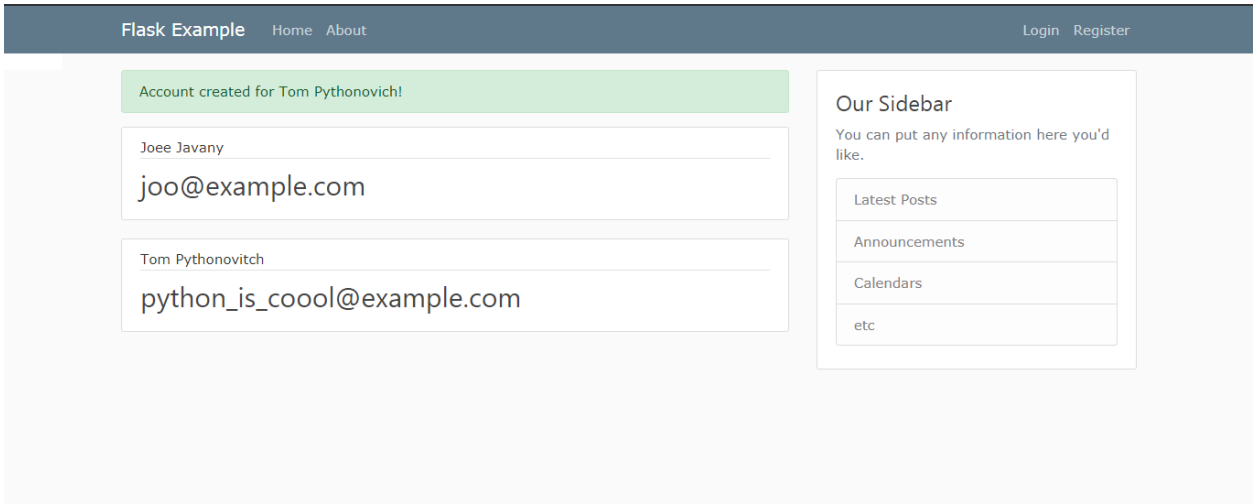
```
        {{ form.username(class="form-control form-control-lg is-invalid") }}
               <div class="invalid-feedback">
                {% for error in form.username.errors %}
                   <span>{{ error }}</span>
                {% endfor %}
                </div>
      {% else %}
        {{ form.username(class="form-control form-control-lg") }}
      {% endif %}
</div>
```

ואותו הקוד נבצע גם על שאר השדות ב-form .

אחרי שסיימנו את זה (ושמרנו!) נריץ את השרת שוב וננסה להירשם בצורה לא תקינה, למשל לא למלא את כל השדות וכו', אנחנו אמורים לקבל שגיאה שנראית פחות או יותר כך:



ובמידה ונרשמנו כמו שצריך ננותב למסך הבית עם הודעה מתאימה וזה יראה כך:



החלק של דף ההתחברות זהה כימעט לחלוטין לעמוד ההרשמה למעט שאין צורך בשם משתמש או אימות סיסמא.

בנוסף אפשר להוסיף גם שדה של remember me כך :

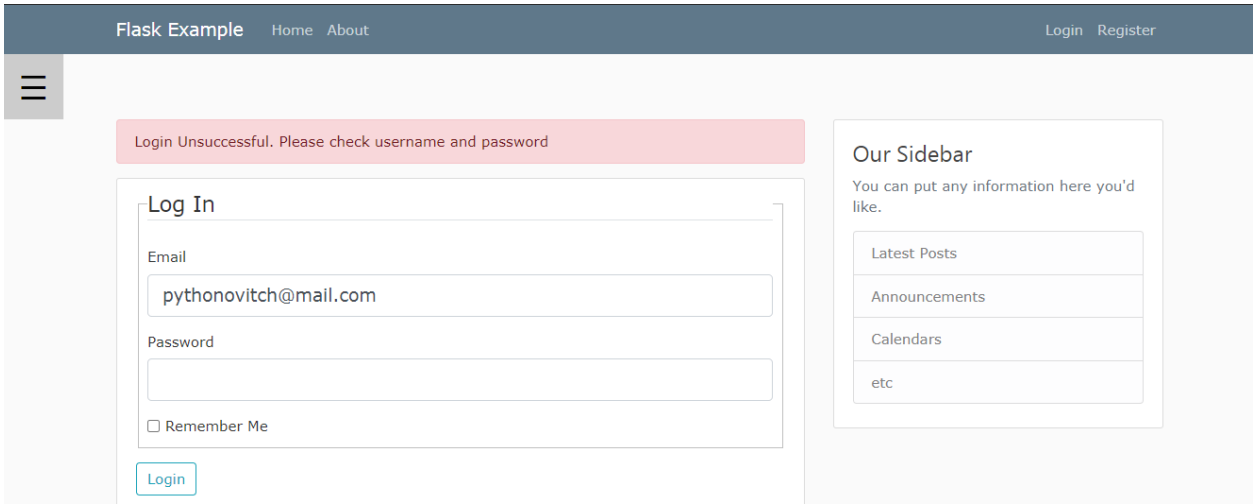
```
<div class="form-check">
     {{ form.remember(class="form-check-input") }}
     {{ form.remember.label(class="form-check-label") }}
</div>
```

וכמובן לשנות את פונקציית הניתוב ב-routes.py .

היות ועדיין לא התחברנו למסד הנתונים, ואנחנו רוצים לוודא שאכן המסמך עובד , נגדיר שאם המתשמש הכניס את הקלט admin@blog.com עם הסיסמא password אז האימות תקף, ובמידה ולא עדיין תשלח למשתמש הודעה שיש בעיה עם האימות עם הקטגוריה danger, רק בלי לנתב אותו לדף הבית:

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        if form.email.data == 'admin@blog.com' and form.password.data == 'password':
            flash('You have been logged in!', 'success')
            return redirect(url_for('home'))
        else:
            flash('Login Unsuccessful. Please check username and password', 'danger')
    return render_template('login.html', title='Login', form=form)
```

אם זה יעבוד כמו שצריך זה אמור להיראות בערך כך:



עוד משהו שאפשר להוסיף הוא קישור להרשמה מדף ההתחברות והפוך , או להוסיף fogot password וכו'.

אני יודע שזה היה מסמך ארוך וקשה מאוד לעקוב אחריו עם כל השינויים בקוד ובעיקר בחלק של העיצוב, אז אנסה לסכם בכמה מילים מה עשינו.

- התקננו את הספרייה flask-wtf שמסייעת בבניית מבנים (forms) עבור מסמכי html וחוסכת לנו את הצורך בבניית טפסים מהבסיס.
- בנינו שני אובייקטי form אחד עבור דף ההרשמה ואחד עבור דף התחברות.
- שיפרנו את העיצוב של האתר, ויצרנו את שני קבצי ה-register, html ו-login .

- למדנו מה זה post ו-get וכיצד להשתמש בהם בפונקציות ניתוב, ואיך לקבל קלט מהמשתמש על ידן.
- למדנו מה זה flash, url_for ו-redirect של flask.
- ראינו כיצד מגדירים קלטים שגויים ואיך ניתן להציג אותם על המסך.