FLASK AUTHENTICATION

סיכום ביניים של החלקים הקודמים:
בפרקים הקודמים ראינו כיצד לבנות template של flask על עמוד html ; ראינו איך אפשר לנתב בין דפים שונים של האפליקציה עם הקשטן route והפונקציות url_for, render_template ושינוי נתיב עם redirect ;
אח"כ הכרנו את sqlalchemy וכיצד להשתמש בספרייה כדי לשמור נתונים במסד דינאמי (ניתן לחבר אליו כמה סוגי מסדי נתונים) הבנוי בצורה של מחלקות; ובפרק הקודם הצגנו את wtforms ואיך אפשר לבנות טופס או מבנה html שיבצע תקינות קלט בשבילנו מבלי שנצטרך להגדיר זאת מראש, בסוף השיעור ראינו איך להעביר נתונים בין דף ה-html לשרת עם מתודות post ו-get.

השלב הבא הוא לקשר בין מסד-הנתונים לקלט מדף ה-html.

ביצוע hashing על הסיסמאות-
לפני שנתחיל, נרצה לשפר את אבטחת המידע בקוד.
כרגע הסיסמאות נשמרות כקובץ טקסט, זה יכול להיות מאוד בעייתי, כי אם למישהו תהיה גישה למסד הנתונים באיזושהי דרך הוא יכול לראות את הסיסמאות. יש כמה אלגוריתמים שימושיים כדי לטפל בבעיה, אנחנו נשמש באלגוריתם שנקרא bcrypt, שהיתרון בו שיש הרחבה במיוחד ל-flask שהיא מאוד פשוטה לשימוש שנקרא, כפי שבוודאי חלקכם ניחשתם flask-bcrypt . התקנה משורת הפקודה:

```
pip install flask-bcrypt
```

אז איך זה עובד? לאחר ההתקנה פיתוחו את המצב האינטרקטיבי של פייתון.
נייבא מהספרייה flask_bcrypt את המחלקה Bcrypt, ונשתמש במתודה generate_password_hash() שמקבלת כפרמטר מחרוזת שמייצגת סיסמא. המתודה תבצע על המחרוזת פונקציית hashing ותחזיר מחרוזת חדשה מאובטחת המייצגת את הסיסמא שהוכנסה כארגומנט:

```
>>> from flask_bcrypt import Bcrypt
>>> bcrypt = Bcrypt()
>>> bcrypt.generate_password_hash('password')
b'$2b$12$N.a9hC32DDOQFIGHMQlc3e.3tYn.038/o70jNF9m2q5HSXYgFM7yS'
```

האות b בתחילת המחרוזת מציינת שזאתי מחרוזת בניארית, אם נרצה מחרוזת רגילה ולא בינארית נוכל להתשמש במתודת decode('utf-8'). נשים לב שכל פעם שקוראים לפונקציה נוצרת מחרוזת חדשה:

```
>>> bcrypt.generate_password_hash('password').decode('utf-8')
'$2b$12$2iF4IqOHsdopaVUbloejBe4/WqK9hqczKpAjvBnnJBU1k2aPZiFWG'
>>> bcrypt.generate_password_hash('password').decode('utf-8')
'$2b$12$lnY/G0u6jfqWDXEnNapuE.G.Ymqe0PuiwkwxU6cL2sMUHmQcr1lsG'
```

כל פעם שהמשתמש יכניס את הסיסמא היא תבצע תהליך קידוד כך שהסיסמא תישאר נסתרת, אבל אם כל פעם הפונקציה מביאה לנו סיסמא מאובטחת חדשה איך נדע לזהות את אם המשתמש הכניס את אותה סיסמא?
בשביל זה נשתמש בפונקציה check_password_hash() שמקבלת מחרוזת עם הסיסמא המאובטחת ומחרוזת עם הסיסמא המקורית ובודקת האם הן מיצגות את אותו דבר:

```
>>> pass_hash = bcrypt.generate_password_hash('password').decode('utf-8')
>>> bcrypt.check_password_hash(pass_hash, 'password')
True
```

עכשיו שאנחנו יודעים את זה בואו נתאים את זה לאתר.
תחילה נחזור ל-__init__.py ונייבא את האובייקט Bcrypt כפי שראינו קודם. אח"כ נצטרך ליצור אינסטנס חדש של

המחלקה , היות וזהו אובייקט של flask אפשר להעביר לו כפרמטר את app ביצירת האינסטנס כך הוא יזהה את כל השדות של האפליקציה:

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

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

לפני שנמשיך, אנחנו משתמשים בעמודה phone במסד נתונים, אבל אין לנו ממש שימוש בעמודה הזאת, כמו כן לא הגדרנו סיסמא כאחת העמודות, לכן נעבור ל-model.py ונשנה את הטבלה של ה-user כך:

```
class User(db.Model):
    id = db.Column(db.Integer , primary_key= True)
    username = db.Column(db.String(20) , unique=True, nullable = False)
    email = db.Column(db.String(20) , unique=True, nullable = False)
    password = db.Column(db.String(60), nullable=False)
    profile_img = db.Column(db.String(20), nullable= False , default='default.jpg')
    posts = db.relationship('Post', backref='author', lazy=True)

    def __repr__(self):
        return f'User({self.username!r} ,{self.email!r},{self.profile_img!r})'
```

שמירה של אובייקט במסד נתונים-

נעבור ל-routes.py ונייבא גם את bcrypt וגם את db שנשתמש בו בהמשך.

```
from flask_example import app, db, bcrypt
```

נלך לפונקציית הניתוב לדף ההרשמה. צריך שם:

- להפעיל פונקציית hash על הסיסמא של המשתמש.
- לשמור את הנתונים במסד.

אנחנו כבר יודעים איך לקרוא את האובייקט שהגיע דרך מתודת post, ראינו בשיעור הקודם, וכמו כן ליצור אינסטנס של האובייקט User גם ראינו, ואיך להשמש בפונקציית hash ראינו הרגע. שימו לב שנרצה ליצור אינסטנס חדש ל-User עם הסיסמא המאובטחת ולא form.password.data .
אח"כ נשמש במשתנה db כדי לשמור את המשתמש החדש.
כמו כן כדאי שנשנה קצת את ההודעה שאנחנו שולחים למשתמש לאחר שההרשמה הצליחה, ונשלח אותו לדף ה-login כדי שיוכל להתחבר:

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

```
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username=form.username.data, email=form.email.data, password=hashed_password)
        db.session.add(user)
        db.session.commit()
        flash('Your account has been created! You are now able to log in', 'success')
        return redirect(url_for('login'))
    return render_template('register.html', title='Register', form=form)
```

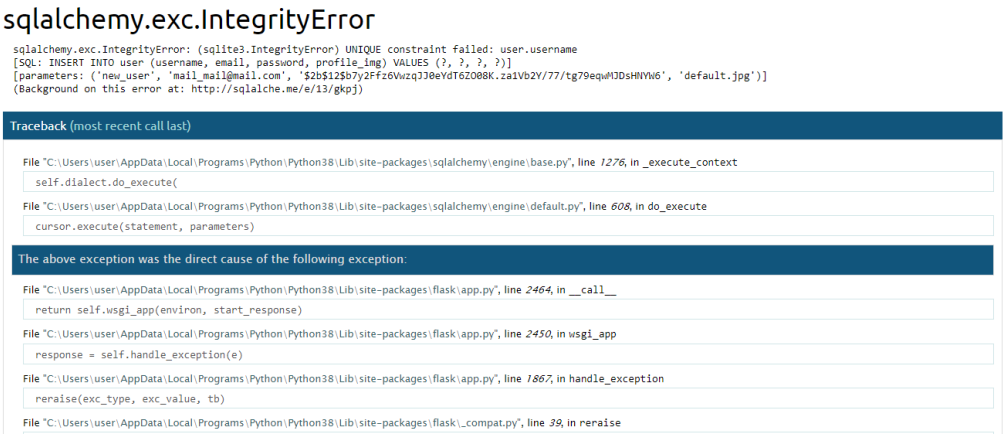
אפשר עכשיו לבחון האם התהליך הצליח- אם הועברנו לדף התחברות סימן שהכל עבר בשלום, כדי לוודא שאכן נשמר האובייקט במסד נתונים אפשר לפתוח חלון חדש של המסוף (cmd או טרמינל וכו') ולפעיל את פייתון, לייבא את הספרייה עם המחלקות כפי שראינו בשיעור של המסד נתונים ולברר שאכן הוכנס משתמש חדש למסד תחת user:

```
>>> from flask_example import db
...
>>> from flask_example.models import User
>>> User.query.all()
[User('new_user' ,'user@mail.com','default.jpg')]
>>> first = User.query.first()
>>> first.password
'$2b$12$bGiS7Hd9R.9pBGgb7KwAFupK2cBl3jNuyMe0/hrgFdGfpONQTNfky'
>>> first.username
'new_user'
```

מעולה.

validators מתואמים אישית-

טוב לא לגמרי מעולה, אם נכניס לאתר שם של משתמש שכבר קיים, או מייל שכבר קיים במסד נקבל שגיאה ונעבור לעמוד ה-debug:



זה משום שדאגנו שהמסד ישמור שמות משתמשים ומיילים כשדה מיוחד (unique) לכן אם נכניס משתמש שכבר קיים במערכת תיזרק שגיאה שתעביר אותנו לדף ה-debug. כדי להימנע מזה צריך להוסיף איזושהי פונקציית validation לאובייקטי form של ההרשמה, כך שמתי שמשתמש חדש ינסה להצטרף, אם השם שהוא הזין כבר שמור במערכת, אז במקום שתזרק שגיאה שתעביר לדף ה-debug נדאג שהשגיאה תוצג על ה-form.

נעבור למסמך form.py ונוסיף פונקציה validation חדשה. עכשיו לפי הדוקומנטציה של wtforms ניתן ליצור מתודות validators מותאמות אישית בתוך האובייקט אם שומרים על מבנה אחיד- שם המתודה היא validate_field() ובמקום המילה filed כותבים את שם המשתנה שעליו עושים את ה-validation, והפרמטרים של המתודה הם self ושם המשתנה שעליו עושים את ה-validation. בתוך הפונקציה צריך לבדוק שמתקיים תנאי כלשהו (תנאי תקפות לפרמטר) אם הוא לא מתקיים צריך לזרוק ValidationError עם טקסט מתאים. כמובן שבמקרה שלנו נצטרך לייבא את User מ-model קודם

כדי שנוכל לבצע את שאילתא על הטבלה User, ולייבא את ValidationError מ-wtforms.validators . בסוף זה יראה כך:

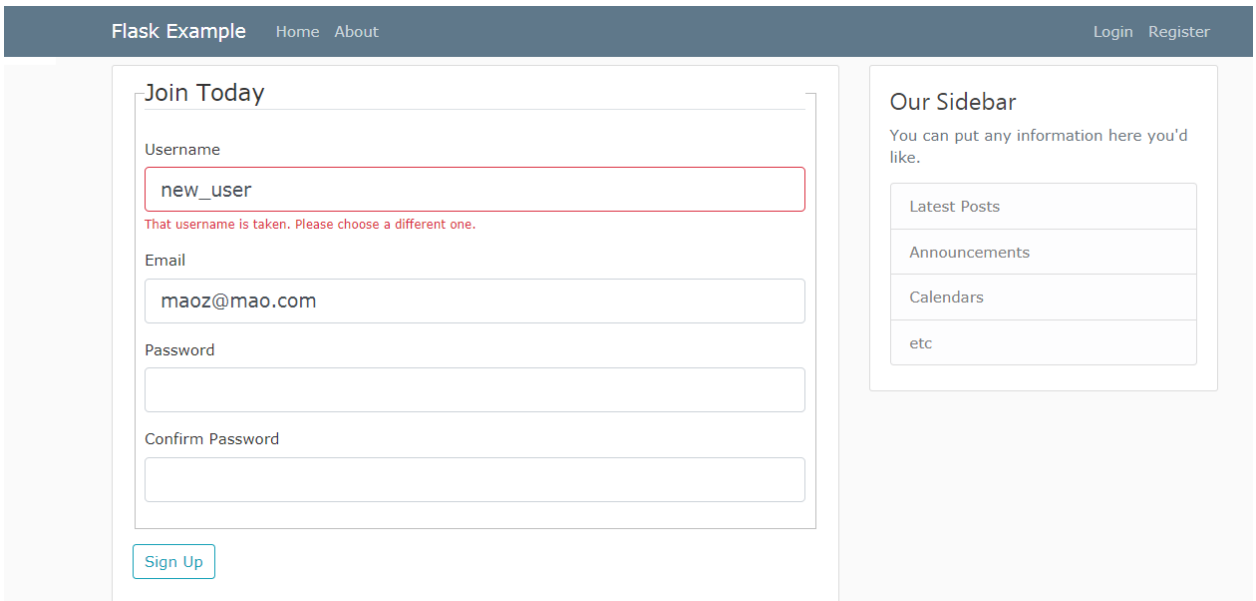
```
...
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from flask_example.models import User


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')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('That username is taken. Please choose a different one.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('That email is taken. Please choose a different one.')
...
```

ועכשיו אם ננסה להכניס שם או מייל של משתמש שכבר נקבל הערה על כך שהמשתמש כבר קיים המערכת:



login למערכת –

עכשיו שיש לנו מערכת רישום דיי טובה, צריך לשפר את מערכת ההתחברות.
כרגע אנחנו עדיין לא יצרנו את האותנטיקציה שהמייל והסיסמא שהוכנסו תואמים למה שיש במסד הנתונים.
כמו כן אנחנו רוצים שתיהיה אפשרות גם להתנתק מהאתר ולא רק להתחבר אליו.
בכדי לעשות את זה נשתמש בעוד הרחבה ל-flask שנקרא flask-login שמקלה על התהליך.
אז דבר ראשון בואו נתקין את ההרחבה עם pip :

```
pip install flask-login
```

לאחר ההתקנה יש להוסיף את הספרייה לקובץ __init__.py כפי שעשינו עם שאר ההרחבות, או יותר מדויק אנחנו רוצים רק את המחלקה LoginManager מתוך כל הספרייה:

```
from flask import Flask, render_template
...
from flask_login import LoginManager
```

ומשום שזה הרחבה ל-flask נוכל להשתמש ב-app כפרמטר לאינסטנס חדש, ככה הוא יזהה את השדות של האפליקציה.

```
...
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
```

לפי הדוקומנטציה של הספרייה, כדי שההרחבה תעבוד כמו שצריך חייבת להיות פונקציית loader שמחזירה את הפריט לפי id כלשהו, למשל אנחנו צריכים להחזיר את המשתמש לפי ה-id שלו. הפונקציה צריכה להיות פונקציה מקושטת בקשטן מיוחד של LoginManager שנקראת user_loader. את הפונקציה ניצור בסקריפט models.py, לכן גם נייבא את המשתנה login_manager מהקובץ __init__.py .

```
...
from flask_example import db , login_manager

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

ההרחבה מצפה שלמודל שעליה היא פועלת (במקרה שלנו User) צריך להיות תכונות מסוימות כמו מתודה שנקראת isauthenticated שמחזירה True אם אכן יש תאימות בין הקלטים של המשתמש למה שמופיע במערכת, עוד פונקציה היא isactive ויש עוד כמה. הפונקציות האלה כל-כך שכיחות שההרחבה מספקת לנו מחלקה, שנקראת UserMixin , ואפשר לרשת ממנה, אז במקום שנצטרך ליצור את הפונקציות האלה בעצמנו, נוכל פשוט לייבא את המחלקה מ-flask_login ולדאוג שהמודל User יירש גם ממנה:

```
...
from flask_login import UserMixin
...
class User(db.Model,UserMixin):
    id = db.Column(db.Integer , primary_key= True)
...
```

נחזור ל-routes.py . עכשיו שביצענו את כל ההכנות האלה נוודא שפונקציה הניתוב login אכן מצליחה להתחבר למערכת לפי מייל וסיסמא.
תזכורת: הפונקציה מקבלת בהודעת Post את המייל והסיסמא של המשתמש.
נצטרך לוודא שני דברים, דבר ראשון שיש אכן משתמש עם אותו מייל, דבר שני שהסיסמא שהוכנסה תואמת לסיסמא ששמורה במערכת. אם שני התנאים מתקיימים ננתב את המשתמש לעמוד הבית, ונשתמש בפונקציה login_user של flask_login (לא לשכוח לייבא את הפונקציה) עם הפרמטר remember ששומר את הערך של השדה remember me של הדף.
אם התנאים לא התקיימו נשלח את ההודעה שכבר שלחנו קודם לכן עם הפונקציה flash:

```
from flask_login import login_user
...

@app.route("/login", methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            return redirect(url_for('home'))
        else:
            flash('Login Unsuccessful. Please check email and password', 'danger')
    return render_template('login.html', title='Login', form=form)
```

נריץ ונוודא שהכל עובד כמו שצריך, נכניס את המשתמש ששמור במערכת ונוודא שאכן מגיעים לעמוד הראשי, ולאחר מכן נחזור לדף ההתחברות שוב, נכניס מייל או סיסמא שגויים ונוודא שהגענו שהמערכת אכן לא מזהה, ולא הגענו לעמוד ה-debug.
משהו שנשאר לתקן זה שכאשר מתחברים למערכת, צריך שהיא לא תאפשר להתחבר שוב, כי אחרי הכל כבר התחברנו ואין היגיון להמשיך לראות את האופציות האלה.
ל-flask-login יש משתנה שנקרא current_user וכשמו כן הוא, מורה מי המשתמש שעכשיו מחובר למערכת.
נשתמש במשתנה הזה כדי לבדוק שאכן יש משתמש שעבר אותנטיקציה ,ואם כן אז לא צריכה להיות אפשרות להתחבר למערכת שוב או לפחות אנחנו אמורים לחזור חזרה לעמוד הבית.
לשם כך נייבא את המשתנה ונוסיף בדיקה האם המשתנה כבר ביצע אותנטיקציה, במידה וכן נעשה redirect לעמוד הבית.
את זה נעשה בשתי הפונקציות ניתוב login ו-register:

```
...
from flask_login import login_user , current_user
...
@app.route("/register", methods=['GET', 'POST'])

def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
...

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

def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
```

```
...
```

logout והתאמה של ה-template-

עכשיו אם ננסה ללחוץ על register או על login הם לא יעבדו לנו, אבל עדיין רואים את שני הכפתורים האלה.
בוודאי נעדיף שבמקום יראו logout או כיתוב בסגנון.
נוסיף דף logout, אבל קודם ניצור פונקציית ניתוב למקרה של ניתוק. נשתמש באובייקט logout_user של flask_login
(לא לשכוח לייבא אותו גם) כדי לנתק את המשתמש מהמערכת וכמובן ננתב חזרה לדף הבית בסוף הניתוק:

```
...
from flask_login import login_user , current_user , logout_user
...

@app.route("/logout")
def logout():
    logout_user()
    return redirect(url_for('home'))
```

נלך עכשיו ל-layout.html ונשנה את החלק בקוד שבו מופיעים login ו-register (בערך שורה 34).
נבצע בדיקה אם המשתמש מחובר או לא ולפי הבדיקה נדע אם להוסיף אותם או את ה-logout:

```
<!-- Navbar Right Side -->
<div class="navbar-nav">
  {% if current_user.is_authenticated %}
  <a class="nav-item nav-link" href="{{ url_for('logout') }}">Logout</a>
{% else %}
  <a class="nav-item nav-link" href="{{ url_for('login') }}">Login</a>
  <a class="nav-item nav-link" href="{{ url_for('register') }}">Register</a>
{% endif %}
</div>
```

נשמור ונראה שאכן מופיע שם logout במקום register ו-login, ואם לוחצים עליו הוא מנתק אותנו מהערכת.

עמוד account-

נניח שהמשתמש מחובר למערכת אבל הוא שינה את המייל שלו והוא רוצה לעדכן אותה במערכת, או שהוא רוצה לשנות את הסיסמא שלו, צריך לבנות דף account שבו הוא יוכל לשנות את פרטיו.
תחילה ניצור איזשהו template לדף ה-account. בשביל החלק הזה מספק לנו שיוצג המשתנה רק שם המשתמש שיתחבר, בשיעור הבא נעבוד על כל השאר.
הדף אמור לשמור על ה-template המקורי ולהוסיף לו כותרת שהיא שם המשתמש.
מבלי לכתוב יותר מידי קוד זה אמור להיראות בערך כך:

```
{% extends "layout.html" %}
{% block content %}
    <h1>{{ current_user.username }}</h1>
{% endblock content %}
```

נחזור ל-routes.py. אם ננסה להוסיף פונקציית ניתוב לדף account זה יעבוד לנו מעולה, אבל מה יקרה אם נתנתק מאותו משתמש וניכנס לדף account/... ? נקבל פשוט דף ששומר על ה-template רק בלי תוכן.
במקום זה, עדיף לנתב לדף login כדי שהמשתמש ידע שהוא צריך להתחבר.
עכשיו אפשר לעשות בדיקה בתוך הפונקציה אם המשתמש מחובר כפי שעשינו אם הפונקציה register או login, אבל יש דרך פשוטה יותר. היות והפונקציה כולה צריכה שהמשתמש יהיה מחובר נוכל להשתמש בקשטן מיוחד שנקרא login_requierd, וכמובן שנצטרך לייבא אותו תחילה:

```
from flask_login import ... ,login_required
...
@app.route("/account")
@login_required
def account():
    return render_template('account.html', title='Account')
```

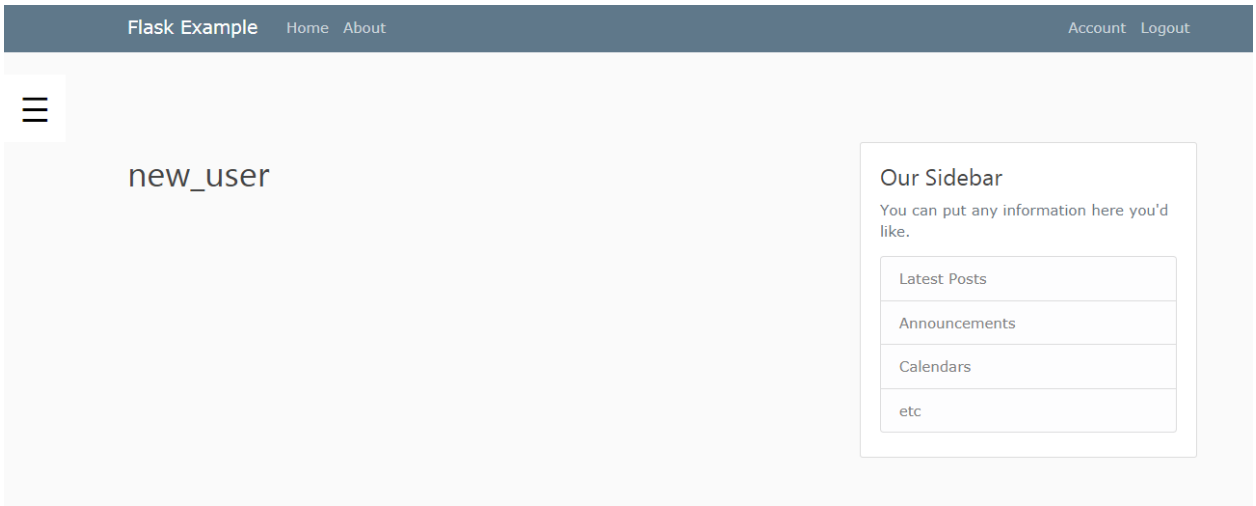
עכשיו צריך לעדכן את המשתנה login_manager היכן ממוקמת פונקציית ה-login שידע להעביר אליה. שם הפונקציה נשמר במשתנה שנקרא login_view.
עכשיו אפשר להגדיר את המשתנה בדף routes.py אבל כדי לשמור על הסדר, הגדרנו את משתנה login_manager ב-__init__.py אז עדיף להמשיך משם.
נחזור ל-__init__.py ונכניס את שם פונקציית הניתוב של הדף ההתחברות לתוך המשתנה login_view , חוץ מזה אם נרצה שההודעה שתוצג, במקרה שניסנו להיכנס ל-account בלי משתמש, תראה כמו ההודעות הישנות (כהודעה אינפורמטיבית) נצטרך להגדיר את הקטגוריה של סוג הבעיה. בנתיים ראינו שני סוגי קטגוריות- success, כמו כשמתחברים בהצלחה, או אזהרה, כמו שמנסים להכניס קלט לא תקין. נשתמש ב-info כדי להציג הודעה אינפורמטיבית.
את ההגדרה ניתן לעשות דרך המשתנה login_message_category :

```
...
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
login_manager.login_message_category = 'info'
```

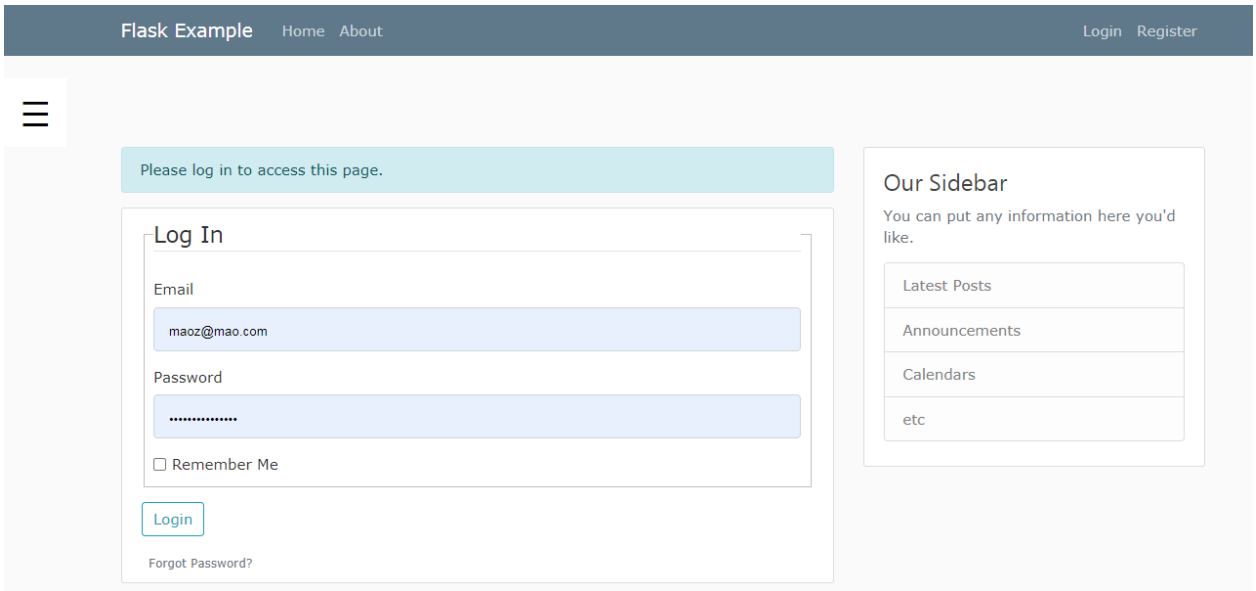
וכמובן שצריך לעדכן ב-layout.html להציג את הדף account.html אם אנחנו מחוברים (בערך שורה 35):

```
...
<div class="navbar-nav">
              {% if current_user.is_authenticated %}
                <a class="nav-item nav-link" href="{{ url_for('account') }}">Account</a>
              <a class="nav-item nav-link" href="{{ url_for('logout') }}">Logout</a>
            {% else %}
            ...
```

נשמור ונריץ, ננסה להיכנס למשתמש ששמור במערכת, ואנחנו אמורים לראות שאפשר להיכנס ל-account:



עכשיו נתנתק מהמשתמש וניכנס ל-localhost:5000/account ואנחנו אמורים לעבור אוטומטית לדף ההתחברות:



- Request

אני יודע שיצא ארוך, אבל זה הנושא האחרון למסמך זה.
כשניסינו להיכנס לדף account ללא משתמש והגענו לדף ההתחברות כתובת האתר הדף היתה משונה, במקום להיות localhost:5000/login היא היתה משהו כזה: localhost:5000/login?next=%2Faccount. זהו סוג של query, הוא מציג לנו את פונקציית הניתוב שהופעלה ואיזו פונקציה היתה אמורה להיות מופעלת.
כרגע אם נמשיך ונתחבר שוב לא נחזור לדף account אלא לדף הבית. אבל נוכל להשתמש ב-query כדי לנתב לדף account במקרה והגענו ממנו.
ל-flask יש אובייקט שמכיל בתוכו מילון עם כל הפרמטרים שמופיעים ב-query של ה-url. לאובייקט קוראים request והוא יאפשר לנו לבדוק אם קיים משתנה שנקרא next שמיצג את פונקציית הניתוב שהייתה אמורה להיקרא.
כדי להגיע אליו נשתמש במילון args של request ובמתודה get('next'), ולא באופרטור ['next'] היות והאופרטור זורק שגיאה אם לא קיים והפונקציה מחזירה None אם לא קיים.

נחזור ל-routes.py, נייבא את האובייקט request מ-flask ונעדכן את הפונקציה login בהתאם:

```
from flask import ... ,request

...
@app.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            next_page = request.args.get('next')
             return redirect(next_page) if next_page else redirect(url_for('home'))
        else:
            flash('Login Unsuccessful. Please check email and password', 'danger')
    return render_template('login.html', title='Login', form=form)
...
```