# FLASK- DB

לפני שניכנס למסדי נתונים של flask נסדר את הספרייה בצורה שתקל עלינו בהמשך, ולא נצטרך להכניס את כל הקוד לקובץ אחד.  
ראשית ניצור תיקייה חדשה שתכיל קובץ \_\_init\_\_.py , תזכורת קובץ \_\_init\_\_.py מגדיר תיקייה כ-namespace עבור כמה מודולים.  
לצורך הדוגמא נקרא לתיקייה flask\_example .  
נעביר את החלק הראשון של הקוד (ייבוא הספריות) מסקריפט הראשי לסקריפט \_\_init\_\_ כלומר בנתיים הוא יראה כך :

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

נכניס את templates לתיקייה ,נוסיף סקריפט חדש עם השם routes.py , ונעביר את כל שקשור לניתוב (לא כולל הפונקציית main) מהסקריפט הראשי לסקריפט הזה:

users = [

    {'name': 'Joee Javany',

    'email': 'joo@example.com',

    'phone': '111-1111'},

    {'name': 'Tom Pythonovitch',

    'email': 'python\_is\_coool@example.com',

    'phone': '222-2222'},

]

@app.route('/')

def hello\_world():

    return render\_template('home.html' , users = users)

ונשנה את השם של המודול הראשי להיות run.py כדי שיהיה קל לזהות אותו.   
כדי שנוכל להשתמש במשתנה app צריך לייבא אותו מהתיקייה שיצרנו. בסוף הסקריפט run.py צריך להיראות ככה:

from flask\_example import app

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug = True)

נצטרך לייבא את הפונקציה route\_template ואת המשתנה app גם לסקריפט routes.py כדי שנוכל להשתמש בו:

from flask import render\_template

from flask\_example import app

...

ולבסוף נטצרך גם לייבא את routes לסקריפט \_\_init\_\_ אבל את זה נעשה אחרי שניצור את המשתנה app, כי אם נייבא אותו לפני תיזרק שגיאה, שכן הקוד מנסה להפעיל את routes ,אבל routes מסתמך על אובייקט שעדיין לא הוגדר (על app):

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

from flask\_example import routes

ונריץ את run.py כדי לוודא שהכל עובד כשורה.

**Flask-sqlalchemy-**

אנחנו נשתמש במסד נתונים sqlalchemy.   
sqlalchemy היא ספריית ORM (Object Rational Mapper), שזה אומר בעיקרון שהיא מאפשרת לגשת למסד נתונים בדרך מונחת עצמים פשוטה ,והדבר הטוב ביותר בה שהיא מאפשר שימוש בכמה סוגים שונים של מסדי נתונים מבלי לשנות את קוד המקור יותר מידי, כך שאם למשל נרצה להשתמש ב-sqllite כדי לבחון את המערכת, ובמסד נתונים אחר בשביל הפרודוקציה בסה"כ צריך רק לתת כתובת uri שונה ל-sqlalchemy שעליה היא תבצע את יצירת השרת, כל שאר הקוד יישאר אותו הדבר.

התקנה:

pip install flask-sqlalchemy

הערה: יש ספריית sqlalchemy רגילה, אבל משום שיש הרחבה מיוחדת עבור flask שמספקת הרבה מאוד ברירות מחדל שימושיות עבור השרת נתקין אותה.  
לאחר ההתקנה נחזור לקובץ \_\_init\_\_ נייבא את המחלקה SQLAlcheny מהספרייה flask-sqlalchemy.  
עכשיו צריך להגדיר את המסד נתונים בו משתמשים, בשביל להגדיר אותו צריך להשתמש באובייקט config של flask.   
האובייקט מגדיר את הקונפיגורציה של השרת בזמן היצירה שלו, אנחנו נשמש בו כדי להגדיר את SQLALCHEMY\_DATABASE\_URI שהוא קונפיגורציית הקישוריות של מסד הנתונים של השרת.   
האובייקט עובד בצורה של מילון, הוא מקבל כמפתח את שם הקונפיגורציה והערך שלו ,במקרה זה לפחות, הוא מחרוזת עם ה-URI למסד. בינתיים נשתמש ב-sqlite בתור המסד, לכן בתחילת ה-uri נכתוב 'sqlite:///' השלוש קווים מסמנים path יחסי של המסד נתונים מהקובץ הנוכחי (relative path) , ואם המסד נתונים לא מוגדר בכלל הוא יוגדר באותה תיקייה שך הסקריפט; אחר כך נוסיף את שם המסד נתונים, נניח קוראים site.db .   
ולבסוף ניצור אינסטנס של המסד נתונים עם המחלקה שייבאנו:

from flask-sqlalchemy import SQLAlchemy  
from flask import Flask, render\_template

app = Flask(\_\_name\_\_)  
app.config['SQLALCHEMY\_DATABASE\_URI']= 'sqlite:///site.db'  
db = SQLAlchemy(app)

from flask\_example import routes

מה שמעולה ב-sqlalchemy שאנחנו יכולים לייצג את טבלאות המסד נתונים שלנו כמחלקות מונחות עצמים שנקראות מודלים (models), כל מודל הוא מחלקה שיורשת מאובייקט שנקרא Model ששייך למסד נתונים (db. Model ),   
כל עמודה מוגדרת כמחלקה שנקראת Column וחייב להכיל את טיפוס הנתונים שאמור לייצג אותה, למשל אם העמודה מייצגת גיל נצטרך להגדיר אותה כאינטג'ר(db.Integer), ושמות כמחרוזת (db.String() ואפשר להגדיר גם גודל מקסימלי, למשל 20 תווים: db.String(20)) וכו'. פרמטרים נוספים שניתן להגדיר הם אם העמודה היא primary key כלומר כזאת שניתן להשתמש בה כדי להסיק על שאר הערכים מטבלאות חיצוניות, למשל תעודת זהות היא primary-key כי היא מספר ייחודי לכל בן אדם ,ואם יש לי אותו אני אז יכול לקבל ספציפית את הערכים של האדם הזה, לעומת שם שיכול להיות שיש שני אנשים עם אותו שם, ואז אני אקבל את שניהם ולאו דווקא את הבן אדם הספציפי שרציתי לקבל; עוד משהו שניתן להגדיר זה האם העמודה ייחודית, כלומר כזאת שאין לה חזרות, האם היא nullable כלומר ניתן לתת לך ערך null, וכו'.  
ניצור שתי טבלאות- אחת שתשמש להכלה של המשתמשים, ואחת להכלה של הפוסטים שלהם.  
הטבלה של המשתמשים תכלול- איזשהו primary-key ,מייל, טלפון ותמונת פרופיל.  
הטבלה של הפוסטים תכלול- primary-key כלשהו, כותרת, תאריך ותוכן

כמו כן צריך להגדיר איזשהו קשר בין משתמש לרשימה של פוסטים שלו, לכן ניצור משתנה חדש מקבל את הערך החוזר מהפונקציה relationship(), הפונקציה מקבלת שם של מודל אחר, אצלנו היא תיהיה Post , ומגדירה קשר ישיר בין המחלקה הזאת למחלקה השנייה. יש לה פרמטר backref שמוסיף לאובייקט של המודל post עוד עמודה שמייצגת את המשתמש שכתב את הפוסט. עוד פרמטר שנשתמש הוא lazy שמגדיר שכאשר מטעינים את הנתונים מהמסד, הם יוטענו בבאת אחת (in one go).  
שימוש לב שהפוסטים הם relationship ולא עמודה של המודל User.  
בנוסף נרצה להוסיף מפתח זר לפוסטים שייצג את הid של הכותב שלו. בשביל להגדיר מפתח זר נצטרך להשתמש באובייקט ForeignKey שמקבל את הטבלה והעמודה בה הוא משתמש, הטבלה המקרה זה היא כשם המודל רק באותיות קטנות, למשל עבור User.id נכתוב user.id   
כדי שלא נתבלבל נשים את הקוד בסקריפט חדש שנקרא models.py ושלא נשתכח נייבא אליו גם את db מהמודול flask-example.

from flask\_example import db  
from datetime import datetime

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)

    phone = db.Column(db.Integer , unique = True)

    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.phone!r},{self.profile\_img!r})'

class Post(db.Model)::

    id = db.Column(db.Integer , primary\_key= True)

    title = db.Column(db.String(100), nullable =False)

    date\_posted = db.Column(db.DateTime, nullable = False , default = datetime.utcnow)

    content = db.Column( db.Text, nullable = False)  
 user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable =False)

    def \_\_repr\_\_(self):

        return f'Post({self.title} , {self.date\_posted})'

בואו נבחן את המסד נתונים ונראה אם הוא עובד כמו שצריך.   
נפתח את המצב האינטרקטיבי של פייתון ונייבא אליו את db ממודול flask\_example .  
אם ביצענו את התהליך כמו שצריך בחלק הראשון אנחנו אמורים לא לקבל שגיאות, אבל לקבל אזהרה מ-SQLALCHEMY אין מה לדאוג מזה זה סה"כ מידע.   
אח"כ נייבא את המחלקות User ו-Post מ-flask\_example.models .

כדי ליצור את המסד ניתונים נשתמש בפקודה db.create\_all() וזה אמור ליצור את כל המתודות של המסד.  
אם קיבלנו שגיאה זה אמור ליצור לנו מסד חדש עם השם site.db בתיקייה flask\_example:

>>> from flask\_example import db

.../\_\_init\_\_.py:833: FSADeprecationWarning: SQLALCHEMY\_TRACK\_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True or False to suppress this warning.

warnings.warn(FSADeprecationWarning(

>>> from flask\_example.models import User , Post

>>> db.create\_all()

>>>

כדי לבחון את המערכת ניצור שני user ונכניס להם כל מיני נתונים, ההגדרה של ה-id תיווצר אוטומטית ולא נצטרך להגדיר אותה ביצירת המשתמשים. כרגע נשמש בתמונה הדיפולטיבית.  
אחרי שהגדרנו כל משתמש צריך להשתמש בפונקציה db.session.add() והארגומנט אמור להיות שם המשתמש שמוסיפים. ולבסוף צריך לעשות גם session.commit() לפעולות שביצענו:

>>> user1 = User(username = 'Tom Pythonovitch', email ='Tom@mail.com', phone =5555555)

>>> db.session.add(user1)  
>>> user2 = User(username = 'joee javany', email ='Joee@mail.com', phone =44444444)

>>> db.session.add(user2)

>>> db.session.commit()

ועכשיו הנתונים אמורים להיות המסד נתונים.  
כדי לבדוק אם הנתונים באמת נמצאים במסד נוכל להשתמש בquery פשוט על המודול שנבחר.  
למשל כדי לראות את כל הנתונים של הטבלה User נשתמש ב- quert.all() על הטבלה:

>>> User.query.all()

[User('Tom Pythonovitch' ,'Tom@mail.com',5555555,'default.jpg'), User('joee javany' ,'Joee@mail.com',44444444,'default.jpg')]

יש כמה שאילתות שניתן לבצע על המסד ולא ניכנס לפרטים, כרגע מה שבעיקר חשוב לנו הוא לדעת איך לבקש משתמש ספציפי לפי איזשהו ערך של הטבלה, למשל לבקש את המשתמש שקוראים לו Tom Pythonivitch.  
בשביל זה נשתמש ב-filter.query() ובשאילתא נכניס לפי איזה פרמטר נרצה להשתמש. נוכל להשתמש ב-all() בשביל לקבל את כל המשתמשים שעונים על השאילתא וב-first() בשביל לקבל את הראשון וכו':

>>> User.query.filter\_by(username = 'Tom Pythonovitch').all()

[User('Tom Pythonovitch' ,'Tom@mail.com',5555555,'default.jpg')]

נשמור את המשתמש במשתנה כדי לגשת לשדות שפציפים שלו למשל id :

>>> user\_1 = User.query.filter\_by(username = 'Tom Pythonovitch').first()

>>> user\_1.username

'Tom Pythonovitch'

>>> user\_1.id

1

באותו אופן אפשר להשתמש בפונקציה query.get() ולהכניס לה את מספר ה-id של המשתמש ולקבל את התוצאה.

>>> user\_1 = User.query.get(1)

>>> user\_1.id

1

כרגע למשתמש אין שום פוסט, אם ננסה לגשת ל-user\_1.post נקבל רשימה ריקה.  
בואו נוסיף שני פוסטים חדשים, ונראה איך הם מחוברים למשתמש user\_1 שה-id שלו הוא 1:

>>> user\_1.posts

[]

>>> post1 = Post(title = 'First', content = '...' , user\_id = user\_1.id)

>>> post2 = Post(title = 'Second', content = '...' , user\_id = user\_1.id)

>>> db.session.add(post1)

>>> db.session.add(post2)

>>> db.session.commit()

>>> Post.query.all()

[Post('First', '2021-02-13 23:17:12.928932'), Post('Second', '2021-02-13 23:17:12.930933')]

>>> user\_1.posts

[Post('First', '2021-02-13 23:17:12.928932'), Post('Second', '2021-02-13 23:17:12.930933')]

כדי למחוק את המסד נתונים נשתמש בפונקציה drop\_all(), ואז כדי ליצור את המסד נתונים שוב נשתמש פעם נוספת בפונקציה create\_all():

>>> db.drop\_all()

>>> db.create\_all()