IAB207 – Rapid Web Application Development

2019 S2

Workshop 07

Forms and Database storage

Agenda

- Introduction
- Last Week Recap
- Assessment Discussion
- Exercise 1: Working with Flask Forms
- Exercise 2: Persisting information in the database
- Exercise 3: Verifying the functionality

Introduction

- Create a form with different fields
- Storing and retrieving data from database
- Use object relational mapping (ORM)

Recap

- Reused HTML
- Dynamic elements in the HTML
- Briefly worked on Forms
- An ‘almost done’ View destination details feature

Work this week

- Store Destination and retrieve
- Destination creation form
- Store comments posted by users

Assessment 3

- Please read the assessment brief carefully
  - Due: Week 13 Workshop
  - Don’t wait till the last minute to start

- Identify team members by end of this week
  - Decide together which team you’ll signup to
  - Blackboard Group Signup
    - Assessment → Assessment 3 (Due Week 13 Workshop) → Group Signup → (Your Workshop Link)

Trello

- www.trello.com

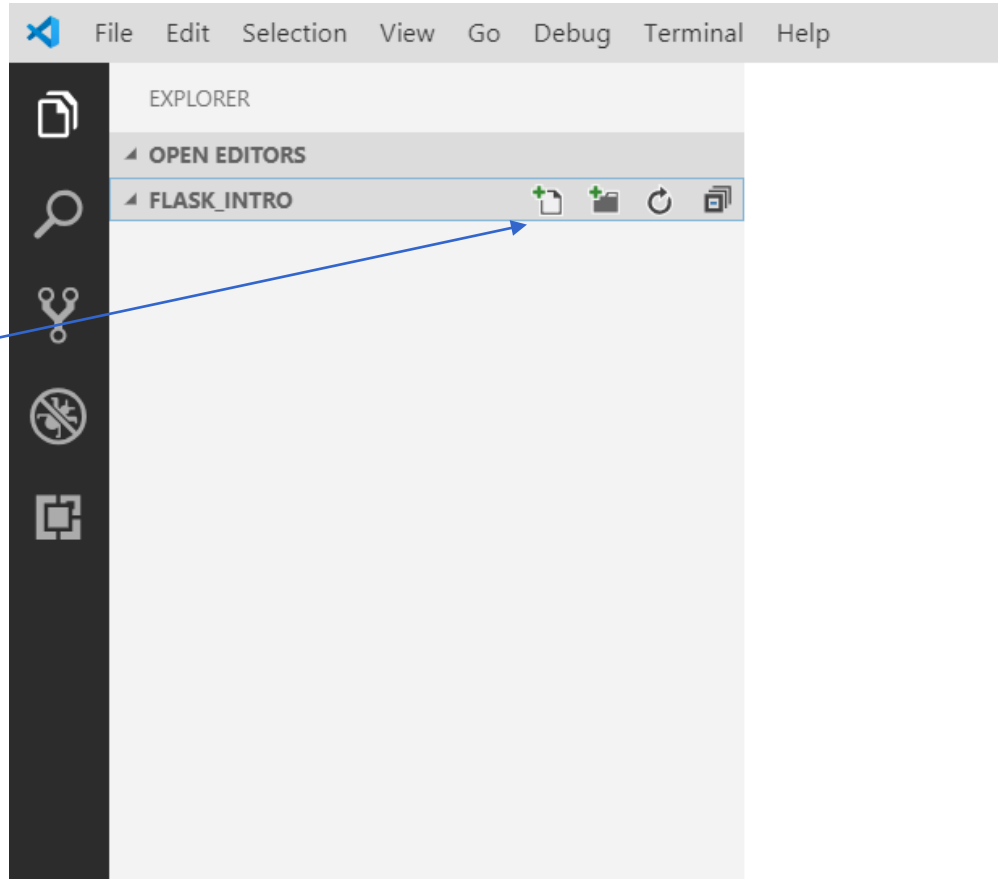
- All teams must maintain a Trello board for project
  - Common structure: To Do, Doing Done
  - Task breakdowns
  - Members assigned to tasks : Who'll do what?
    - All members will keep Trello updated with their progress
  - Add tutors to your board so we can review

Exercise 1 (30 minutes)

- Create a destination form

Create a VS Code Project

- Open Visual Studio Code
- File-> Open Folder-> Browse Directory-'New Folder'-> week7
- Copy the folder travel and main.py from week6
- Check the Python interpreter chosen by the IDE at the bottom left corner.



Adding a ‘Create’ Destination feature

1. Create DestinationForm
2. Create an HTML file that renders the form
3. Define a route and a view function
4. Update the function to render the form and ‘POST’ data captured by the form

1. Create a Form

- Edit forms.py to create a Flask DestinationForm
- We will use one validator called InputRequired which ensures some data is entered in the Form
- https://git.io/fjdxE

2. Create HTML that renders the forms

- In templates/destinations folder, add a new file create.html
- Use bootstrap wtf.quickform() to create the destinations form
- https://git.io/fjdpJ

2. Route and a View function

- Edit destinations.py to add the route and a view function
- A POST method would validate the form and flash a message

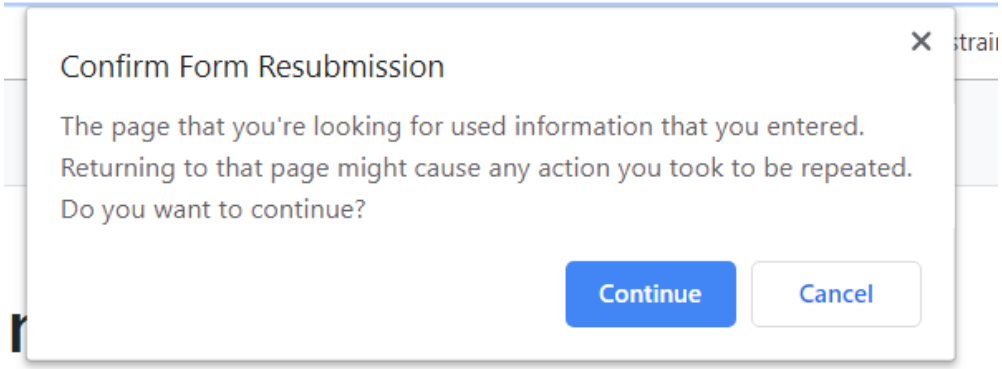
```
@bp.route('/create', methods = ['GET', 'POST'])
def create():
  print('Method type: ', request.method)
  form = DestinationForm()
  if form.validate_on_submit():
    print('Successfully created new travel destination', 'success')

  return render_template('destinations/create.html', form=form)
```

Run the Flask Application

- Run main.py
- Access the URL http://127.0.0.1:5000/destinations/create
- The form appears – add inputs and submit the form
  - Check if you can see the message regarding the successful posting of the Form

Refresh the HTML page

- Refresh the HTML page – You will see a message confirming a resubmission of the form



- This could lead to duplicate entries if the data were to be stored.

POST/REDIRECT/GET Pattern

- https://en.wikipedia.org/wiki/Post/Redirect/Get
- Always end the handling of a POST request with a redirect (which becomes a GET request)

```
@bp.route('/create', methods = ['GET', 'POST'])
def create():
  form = DestinationForm()
  print('Method type: ', request.method)
  if form.validate_on_submit():
    print('Successfully created new travel destination', 'success')
    return redirect(url_for('destination.create'))

  return render_template('destinations/create.html', form=form)
```

Exercise 2 (50 mins)

- Supporting database persistence using SQLAlchemy
  1. Initialize SQLAlchemy
  2. Create database objects and relationships (models.py)
  3. Create the database and the table
  4. Create a method to read from the database
  5. Create a method to write to the database

1. Initializing SQLAlchemy

- pip3 install Flask-SQLAlchemy
- If this doesn't work, use:
  - & /<YourPythonPath>/python.exe -m pip3 install Flask-SQLAlchemy
- In __init__.py
  - Create an SQLAlchemy database object (global variable) `db=SQLAlchemy()`
  - In the create_app() function
    - Configure flask environment variable to point to a Database `'SQLALCHEMY_DATABASE_URI'`
    - Initialize database with the Flask app

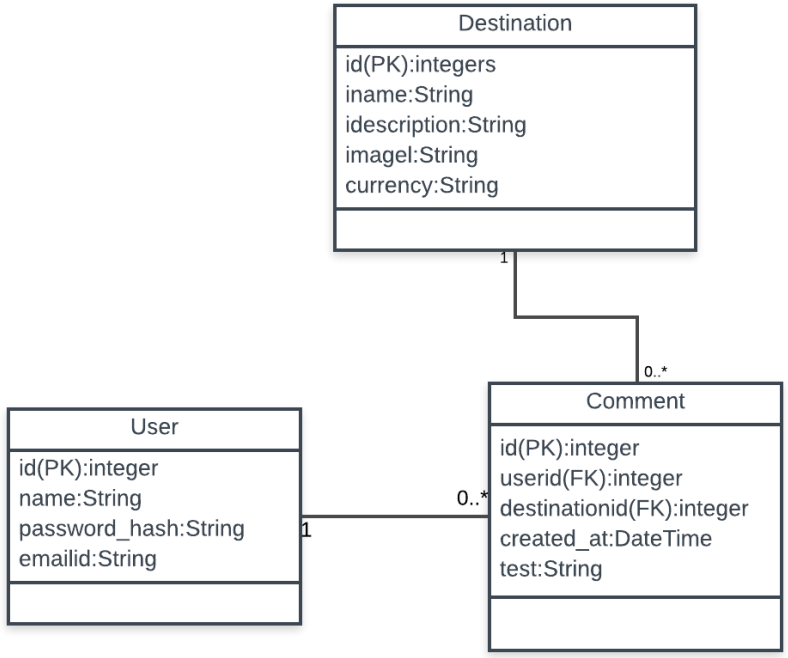
      `db.init_app(app)`
- https://git.io/fjdhK (Refer to code)

2. Create the ORM model

- Update models.py and replace the existing Destination and Comment class with a class that uses SQLAlchemy
  - Keep the attribute names (column names) the same to avoid rewriting the code that reads Destination and Comment in destinations/show.html

- For completeness, we will create a User class that has the name, emailID, and password

- Code Walk through of models.py



Class User

- Create the class User that has the name, email, password and a primary key **id***
- In models.py, import SQLAlchemy db class

```
from . import db
```

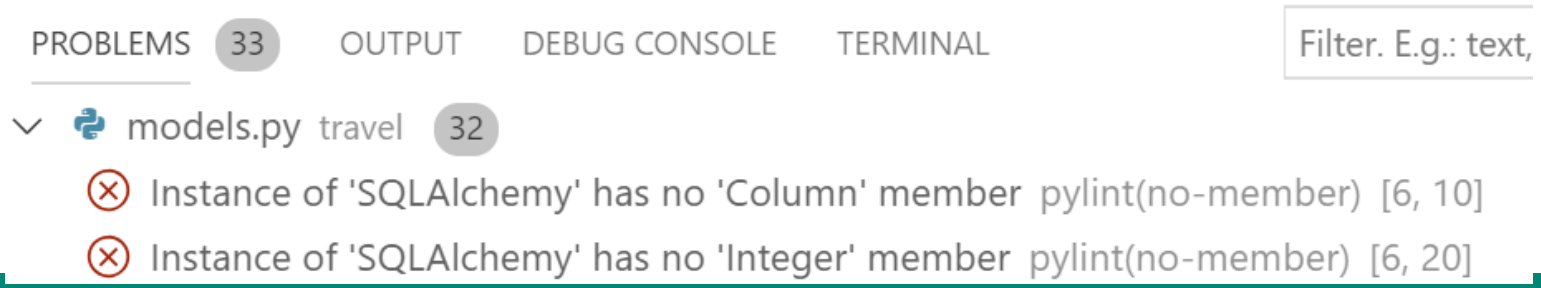
```
class User(db.Model):
        __tablename__='users' # good practice to specify table name
        id = db.Column(db.Integer, primary_key=True)
        name = db.Column(db.String(100), index=True, unique=True, nullable=False)
        emailid = db.Column(db.String(100), index=True, nullable=False)
    #password is never stored in the DB, an encrypted password is stored
    # the storage should be at least 255 chars long
        password_hash = db.Column(db.String(255), nullable=False)
```

* Please do not use any other variable name for user id, we will discuss this in the next workshop

Visual Studio Code Bug with SQLAlchemy

- Visual Studio has a bug and shows pylint error with SQLAlchemy
- Please ignore errors related to SQLAlchemy

```
class User(db.Model):
    __tablename__='users'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), index=True, unique=True,
```



Class Destination

- Create the class Destination with name, description, image and currency fields.
- Add a id as a primary key
- Specify the columns where nullable=false

```
class Destination(db.Model):
        __tablename__ = 'destinations'
        id = db.Column(db.Integer, primary_key=True)
        name = db.Column(db.String(80))
        description = db.Column(db.String(200))
        image = db.Column(db.String(400))
        currency = db.Column(db.String(3))

def __repr__(self): #string print method
        return "<Name: {}>".format(self.name)
```

Class Comment

- Create the class Comment with three fields – id, text, created_at
- Add two foreign key relationships
  - user_id
  - destination_id

```
class Comment(db.Model):
        __tablename__ = 'comments'
        id = db.Column(db.Integer, primary_key=True)
        text = db.Column(db.String(400))
        created_at = db.Column(db.DateTime, default=datetime.now())
#add the foreign keys
        user_id = db.Column(db.Integer, db.ForeignKey('users.id'))
        destination_id = db.Column(db.Integer,
                        db.ForeignKey('destinations.id'))

def __repr__(self):
        return "<Comment: {}>".format(self.text)
```

Updating specific relationships

- To access the related objects, we need to create a relationship in these objects
  - Hint: create relationship in the parent class or the class the foreign key refers to
- In this case, the User class and Destination class

Updating specific back-references

- Create a relationship in the Destination class to access comments

```
# relation to call destination.comments and comment.destination
comments = db.relationship('Comment', backref='destination')
```

- Create a relationship in the User class

```
# relation to call user.comments and comment.created_by
comments = db.relationship('Comment', backref='user')
```

- Refer to the code https://git.io/fjFYH

3. Create the database and the tables

- The database and the tables need to be created before using it in the flask application
- Open the command line or terminal
- Change directory (cd) to the week7 folder
- Run python command in the terminal

Create the database and the tables

from travel import db, create_app
app=create_app()

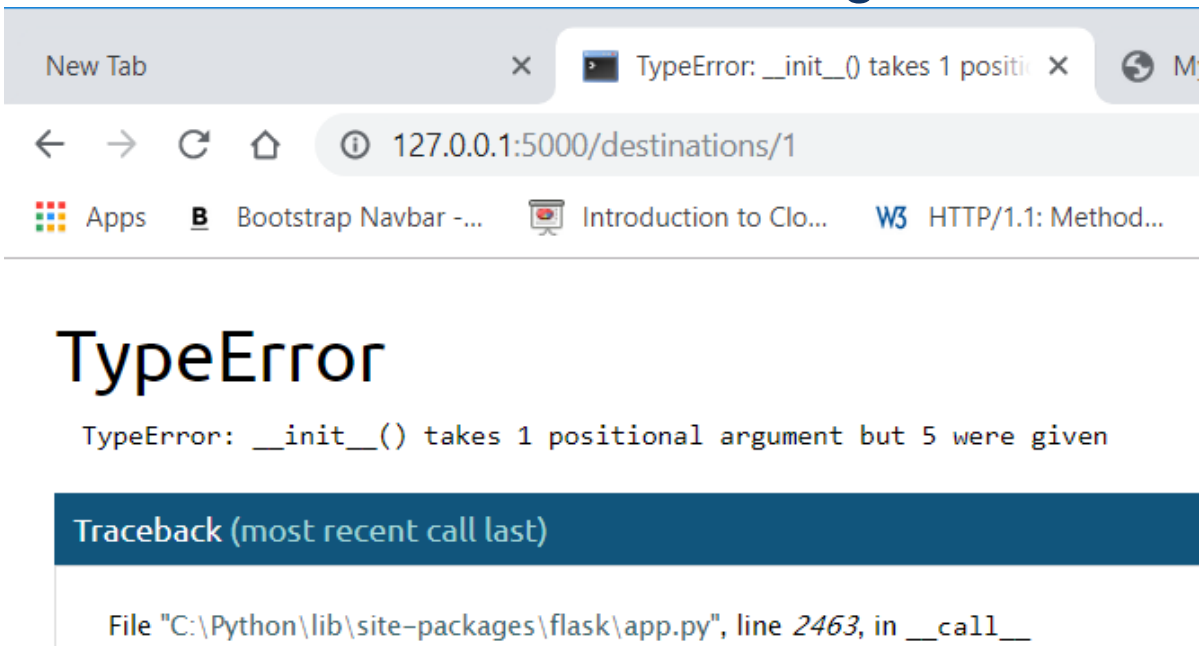
ctx=app.app_context() # flask maintains a context, some objects can be accessed in the app context

ctx.push()
db.create_all()

```
C:\R    \iab207_workshop\week7>C:\Python\python.exe
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09
Type "help", "copyright", "credits" or "license" for mor
>>> from travel import db,create_app
>>> app=create_app()
C:\Python\lib\site-packages\flask_sqlalchemy\__init__.py
significant overhead and will be disabled by default in
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overh
>>> ctx=app.app_context()
>>> ctx.push()
>>> db.create_all()
```

Running the application

- Run main.py – Flask application
  - You will get an error because the Destination class and Comment class is now using DB



4. Method to read from the database

- Reading destination from the database – query and filter by the destination id
- Add the following to the show() view function in destinations.py
- The following statement queries and filters by the column id

```
destination = Destination.query.filter_by(id=id).first()
```

5. Method to store/insert into database

- Update the create() function in the destinations.py to read the values entered in the form and insert into database
- Hint: import the db object in destinations.py

```
if form.validate_on_submit():
# if the form was successfully submitted, access form data
        destination = Destination(name=form.name.data,
                description=form.description.data,
                image=form.image.data,
                currency=form.currency.data)
        # add the object to the db session
        db.session.add(destination)
        # commit to the database
        db.session.commit()
```

Run the Flask Application

- Run the application (main.py)
  - Run Python file in Terminal

- Create a destination
- http://127.0.0.1:5000/destinations/create

- View the created destination
- http://127.0.0.1:5000/destinations/1

Create and view Comments

- The comment added to the form needs to be saved in the database
- Update the view function `comment(destination)`

```
@bp.route('/<destination>/comment', methods = ['GET', 'POST'])
def comment(destination):
        form = CommentForm()
        #get the destination object associated to the page and the comment
        destination_obj = Destination.query.filter_by(id=destination).first()
        if form.validate_on_submit():
                #read the comment from the form
                comment = Comment(text=form.text.data,
                        destination=destination_obj)
        #here the back-referencing works - comment.destination is set
        # and the link is created
                db.session.add(comment)
                db.session.commit()
                print('Your comment has been added', 'success')
        # using redirect sends a GET request to destination.show
        return redirect(url_for('destination.show', id=destination))
```

https://git.io/fjFOZ (Code to refer)

Destination ID in the URL

- The destination/show.html has the destination id hardcoded to 1

- This should be the id of the destination object retrieved from the database

```
{{wtf.quick_form(form,
"/destinations/{0}/comment".format(destination.id))}}
```

Run the application

- Run the application
- Access the URL
  http://127.0.0.1:5000/destinations/create
- Create more destinations
- Access the destinations
  - http://127.0.0.1:5000/destinations/2
  - http://127.0.0.1:5000/destinations/3

- **Next week, we will write code to ensure that the user does not need to know the id, to access a destination**

Summary

Destination Creation

Storing Destination in the Database

Accessing different destinations