

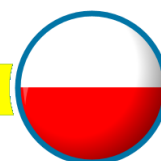


# Python Flask

# Aplikacje Webowe

Podręcznik kursowy

Mobilo © 2021



W tym miejscu zwykle pojawia się informacja o tym kto i jak może posługiwać się tym podręcznikiem. Bez regułek prawnych odwołam się po prostu do kilku prostych zasad, które oddają sens kto, kiedy i jak może wg zamysłu autora korzystać z tego materiału:

- Ten podręcznik jest integralnym elementem kursu online.
- Możesz z niego korzystać będąc uczestnikiem tego kursu. Podręcznik jest dla Ciebie i korzystaj z niego do woli – drukuj, wypełniaj, uzupełniaj, przeglądaj, póki Twoim celem jest samodzielne opanowanie tematu.
- Proszę nie umieszczaj go w publicznie dostępnych miejscach, jak blogi, repozytoria git hub, chomik itp.
- Nie wykorzystuj go w innych celach, np. organizacji własnych szkoleń, gdzie występujesz np. jako instruktor.
- Jeśli nie posłuchasz moich próśb, to jako autor zapiszę się do ZAKS-u i następnym razem kupując smartfona zapłacisz za niego kilkaset złotych więcej 😊, więc może lepiej po prostu przestrzegaj praw autorskich 😊

Z góry dziękuję!

Rafał Mobilo © 2020

Zapraszam do odwiedzenia strony:

<http://www.kursyonline24.eu/>

[Review 2021-05-29](#)

## Spis treści

Wstęp - Dlaczego Flask?	5
Jak się uczyć?	6
Instalacja Visual Studio Code	7
Środowisko wirtualne, instalacja Flask, pierwszy program	9
Terminal, zmienne środowiskowe i typowe problemy	11
Dynamiczny routing	13
Przekazywanie parametrów przez Query String	15
Przyjmowanie danych z formularzy	17
Obsługa GET i POST w jednej funkcji. Obiekt request	21
Funkcja url_for – Twój przyjaciel w budowaniu linków	23
Funkcja redirect, czyli “zapraszamy do kasy obok”	25
Statyczne elementy projektu	27
Definiowanie środowiska Flask	29
Projekt – 01 – Cook book	31
Szablony Jinja	35
Warunkowe wyświetlanie części szablonu Jinja	37
Pętle w szablonach Jinja	39
Funkcja flash() – czyli krótkie info o statusie	43
Makra Jinja	45
Dziedziczenie w szablonach i budowanie własnych standardów	47
Dołączanie szablonów Jinja - include	49
Flask Bootstrap	51
Korzystanie z Bootstrap bez Flask Bootstrap	53
Połączenie kodu z szablonami	55
Stosowanie formatów bootstrap	57
Projekt 02 – Formularz z pomysłami wycieczek	59
Instalacja SQLite 3 i podstawy pracy z danymi	63
Zapis danych z rekordu do bazy danych	65
Pobieranie rekordów z bazy danych	67
Kasowanie rekordu z pytaniem o potwierdzenie	69
Modyfikacja danych	71
Użytkownicy aplikacji - przygotowanie	73

Sesja w akcji – logowanie i wylogowanie użytkownika .....	75
Dodawanie użytkowników – zadbaj o wygodę użytkownika .....	77
Lista użytkowników .....	79
Edycja użytkownika .....	81
Edycja uprawnień .....	83
Implementacja uprawnień w aplikacji .....	85
Projekt – 03 – Aplikacja CRUD .....	87
Flask SQLAlchemy .....	89
Dodawanie i pobieranie danych z bazy danych .....	91
Filtrowanie danych .....	93
Sortowanie, zliczanie i ograniczanie liczby rekordów .....	95
Dodawanie i usuwanie danych .....	97
Relacyjne bazy danych .....	99
Porównanie metod pracy z bazą danych .....	101
Flask-WTF – instalacja i pierwszy formularz .....	103
Sprawdzanie poprawności danych (validators) .....	105
Zapisywanie danych z formularzy w obiektach .....	107
Dziedziczenie z klasy formularza .....	109
Korzystanie z kontrolek WTFForms .....	111
Kontrolki HTML5 .....	113
Flask Login – testowa aplikacja .....	115
Flask-Login instalacja i wykorzystanie .....	117
Przekierowanie użytkownika do strony logowania .....	119
Wymuszenie ponownego logowania .....	121
Projekt – 04 – Aplikacja CRUD – drugie podejście .....	123
Co dalej? .....	124
Dodatek: Typowe błędy .....	125
Spróbuj też! .....	128

## Wstęp - Dlaczego Flask?

Python to jeden z najpopularniejszych języków programowania. Można w nim budować rozwiązania dla różnego rodzaju problemów. Czy to automatyzacja, czy machine learning, czy aplikacja użytkowa, czy tylko prosty skrypt. Każdy z tych rodzajów aplikacji prędzej czy później wymaga jakiegoś interfejsu.

Budując ten interfejs możesz zdecydować się na budowanie aplikacji okienkowej – ot po prostu przyjemna aplikacja uruchamiania w interfejsie graficznym użytkownika. Jak najbardziej, tak można. Okazuje się jednak, że niewiele więcej trzeba, aby aplikacja mogła być uruchamiana zdalnie przez przeglądarkę! Ponieważ na każdym systemie przeglądarka jest dostępna, to i aplikacja będzie mogła być wykorzystywana praktycznie wszędzie – nawet lokalnie na komputerze!

Flask umożliwia budowanie aplikacji webowych, obsługiwanych przez przeglądarkę i nie tylko przez przeglądarkę. Tworzenie takiej aplikacji nie jest trudne, bo Flask jest do tego stopnia prosty, że czasami jest nazywany mikro frameworkiem. Czy to jednak nie przerażające, że Flask jest „micro”? Czy to znaczy, że da się w nim zrobić tylko bardzo ograniczoną część aplikacji? Nic z tych rzeczy. Do Flaski można dołączać rozszerzenia (extensions/modules), które pozwalają robić cuda! Jeden moduł i pracujesz z bazą danych. Drugi moduł i masz w aplikacji oprogramowany proces logowania. Kolejny moduł i formularze zaczynają walidować dane wprowadzane przez użytkowników!

To właśnie główne powody, dlaczego warto zainteresować się Flaskiem. Z jednej strony proste, z drugiej strony potężne, a z .... trzeciej strony – pozwala zbudować ładny i funkcjonalny interfejs do logiki zaprogramowanej już wcześniej w warstwie logiki biznesowej aplikacji

Ponieważ najlepiej uczyć się praktycznie, to do szkolenia dołączony jest podręcznik kursowy (właśnie go czytasz). Dla każdej lekcji znajdziesz w nim krótkie podsumowanie wiadomości – notatkę z lekcji. Do tego jest również zestaw zadań z rozwiązaniami i kilka pytań pozwalających samodzielnie rozważyć działanie prezentowanych poleceń i funkcji.

Zapraszam do nauki Flaski – jednego z przyjemniejszych modułów do budowania aplikacji Webowej

Powodzenia!

Rafał

## Jak się uczyć?

Skoro tutaj zaglądasz, to znaczy, że planujesz poznać Flaska i samodzielnie budować aplikacje webowe. To świetnie!

Naukę oczywiście zorganizujesz sobie po swojemu, ale pozwól, że zaproponuję kilka sposobów nauki, a Ty sam/a wybierzesz, co z tego Ci się podoba, a co wolisz zrobić po swojemu

1. **Co za dużo to niezdrowo** – nie rób na raz za dużo materiału. Nie od razu Kraków zbudowano. Jedna lub dwie lekcje na dzień powinny wystarczyć.
2. **Liczy się regularność** – niekoniecznie uczyć trzeba się codziennie, ale jeśli postanowisz przerabiać lekcje we wtorki, czwartki i soboty to już coś!
3. **Wykonuj zadania praktyczne**. Od samego oglądania filmów nie staniesz się programistą. Trzeba tworzyć zapytania samodzielnie
4. **Zmieniaj treść poleceń na własną rękę**. Wykonaj podobny przykład na innej tabeli, zmień dane. Im więcej kreatywności podczas nauki, tym więcej się zapamiętuje
5. Uczę się uczyć, a do głowy nie wchodzi – wszyscy tak mamy i to pewnie dlatego nauka w szkole trwa aż tyle lat! **Od czasu do czasu zrób sobie powtórkę**. Przecież nikt Cię nie goni i nie rozlicza z postępów.
6. Sugeruję wrócić do zadań. Jeśli potrafisz je rozwiązać – świetnie przerabiaj następną lekcję
7. Jeśli zadania sprawiają problem, wróć do notatki lub lekcji – zobaczysz, że słuchając drugi raz tego samego, materiał nie będzie już taki trudny
8. Notatki w podręczniku są dla Twojej wygody. Niestety wygoda leży blisko lenistwa. Nie bądź leniem. Przygotuj sobie zeszyt lub kilka luźnych kartek i **zapisuj to czego się uczysz**. To co wejdzie oczami lub uszami, będzie wychodzić rękami i... nie ma wyjścia – po drodze zahaczy o mózg 😊
9. Jeśli możesz – wydrukuj sobie podręcznik, dopisuj do niego własne notatki, uwagi itp.
10. Kiedy osiągniesz jakiś „kamień milowy”, ukończysz sekcję kursu, a może nawet cały kurs – **daj sobie nagrodę** – to niesamowicie zwiększa motywację!
11. **Nie bój się korzystać z innych materiałów**: książek, blogów, forów itp. Część z nich na początku może być nieco za trudna, ale nic nie stoi na przeszkodzie, żeby na początku nic nie mówić, tylko słuchać 😊.
12. Kiedy już ukończysz kurs – **zaktualizuj CV na LinkedIn**, pochwal się swoim certyfikatem, daj się odnaleźć rekrutrom, zaproś mnie do znajomych (link w profilu). Chętnie potwierdzę Twoją nową umiejętność!

Powodzenia!

Rafał

## Instalacja Visual Studio Code

### Notatka

- Visual Studio Code czasami nazywany Visual Code Editor (VCE) to jeden z wielu dostępnych edytorów kodu Pythona. Jest wydany przez Microsoft. Inne popularne to:
  - PyCharm
  - Jupyter Notebook
  - Spyder
- VSC jest dostępny na:
  - Windows
  - Linux
  - Mac
- Sam w sobie VSC jest prosty, ale jest dla niego dostępnych mnóstwo dodatków pozwalających na programowanie w wielu językach programowania
- Jedną z popularniejszych wtyczek dla Pythona jest wydana również przez Microsoft
- VSC dobrze integruje się z systemem operacyjnym. Daje możliwość uruchomienia wielu terminali. Na Windows będzie to np. PowerShell lub cmd, a w Linuxie shell

### Laboratorium

1. Pobierz i zainstaluj Visual Studio Code względnie przygotuj na swoim komputerze inne środowisko programistyczne, które lubisz
2. Jeśli to Twój pierwszy kontakt z VSC, to napisz i sprawdź działanie jakiegoś prostego programu, np. „Hello World”.
3. Na własną rękę przejrzyj artykuły porównujące środowiska programistyczne (użyj Google), albo zerknij do artykułu:  
<https://realpython.com/python-ides-code-editors-guide/>

Pusta strona wstawiona celowo



## Środowisko wirtualne, instalacja Flask, pierwszy program

### Notatka

- Dobrą praktyką jest tworzenie środowisk wirtualnych. Dzięki temu:
  - Każde środowisko jest wyizolowane
  - Posiada swoje własne pakiety niewidoczne dla innych projektów
  - Umożliwiają konfigurację wymaganych pakietów w odpowiedniej wersji bez ryzyka przypadkowej aktualizacji pakietu
  - Pozwala na udokumentowanie wykorzystywanych zależności do innych pakietów
  - Ułatwia późniejszą instalację
- Nowe środowisko tworzy się komendą (druga nazwa venv to nazwa tworzonego środowiska i może ona być dowolna):

```
python -m venv venv
```

- Środowisko aktywuje się skryptem **activate** i znajduje się ono w katalogu”
  - venv/scripts na systemie Windows
  - venv/bin na Linux
- Do dezaktywacji środowiska służy polecenie **deactivate**
- Po aktywacji środowiska można w nim np. instalować wymagane pakiety
- Do instalacji Flaska wykorzystaj polecenie:

```
Pip install flask
```

- Wraz z instalacją Flaska instalowane są inne wymagane pakiety
- Domyślna nazwa pliku aplikacji Flask to app.py
- Podstawowa postać skryptu to

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
    return '<h1>Hello world!!!</h1>'
if __name__ == '__main__':
    app.run()
```

- Aby uruchomić aplikację, w terminalu, będąc w katalogu, w którym znajduje się aplikacja, z aktywnym bieżącym środowiskiem uruchom

```
flask run
```

- Ostatnie dwie linijki skryptu są wymagane, gdy program uruchamiasz stosując

```
python app.py
```

- Jeśli aplikacja nie działa w trybie DEBUG, to po dokonaniu zmian w kodzie Flaska należy zatrzymać i uruchomić go ponownie

## Laboratorium

1. Samodzielnie utwórz nowy projekt i utwórz w nim środowisko wirtualne
2. Stwórz aplikację, który na stronie głównej wyświetli godzinę wygenerowania strony:
  - a. Załaduj moduł **datetime**,
  - b. bieżącą godzinę uzyskasz uruchamiając:

```
time_now = datetime.now().strftime('%H:%M:%S')
```

3. Uruchom ten program. Zmień delikatnie formatowanie (np. dodaj znacznik h1.)
4. Zatrzymaj i uruchom Flaska, aby zobaczyć zmiany w przeglądarce.
5. Uruchom program korzystając z polecenia python
6. Dodaj route /links, która wyświetli statyczne linki do dwóch wyszukiwarek internetowych

## Sprawdź się!

1. Domyślnie, po dokonaniu zmian w kodzie te zmiany są widoczne dla aplikacji od razu?
2. Co odpowiada za to, jaka funkcja zostanie uruchomiona przez Flask, gdy w przeglądarce zostanie wprowadzony pewien adres?
3. Co zwracają funkcje w aplikacji Flask?

## Propozycja rozwiązania (tylko kod aplikacji)

```
from flask import Flask
from datetime import datetime

app = Flask(__name__)

@app.route('/')
def index():
    time_now = datetime.now().strftime('%H:%M:%S')
    return '<h1>Hello world: {}</h1>'.format(time_now)

@app.route('/links')
def links():
    body = '<a href="http://www.google.com" target="_blank">Google</a> <br />'
    body += '<a href="http://www.bing.com" target="_blank">Default search engine to'
    body += 'find Google</a>'
    return body

if __name__ == '__main__':
    app.run()
```

## Terminal, zmienne środowiskowe i typowe problemy

### Notatka

- Uruchamiając programy w Flask istotne jest, aby aktywne było właściwe środowisko
- Dotyczy to nie tylko środowiska widocznego dla edytora, ale też środowisko terminala
- Terminal może pracować pod kontrolą PowerShell, Cmd (na Windows) lub pod kontrolą shella (na Linux)
- W przypadku, jeśli w VSC terminal nie pracuje w kontekście właściwego środowiska, to można albo uruchomić polecenie **activate**, albo zamknąć i uruchomić terminal jeszcze raz
- Flask uruchamia program:
  - Wyszukując go po nazwie, która powinna być `app.py`
  - Wyszukując go po nazwie zapisanej w zmiennej środowiskowej `FLASK_APP`
- Zmienne środowiskowe definiuje się inaczej w Windows:
  - W Windows

```
set FLASK_APP=my_app.py
```

- Pod Linuxem

```
export FLASK_APP=my_app.py
```

- Gdy chcesz debuggować program należy ustawić zmienną `FLASK_DEBUG=1`
- Podczas debugowania
  - Gdy dojdzie do błędu można skorzystać z okrojonego debuggera przez stronę web
  - Zmiany w kodzie aplikacji spowodują automatyczny restart Flaska
  - Dostęp do debugger otrzymuje się po wprowadzeniu pinu wyświetlanego przez Flask

### Laboratorium

1. Zmień typ terminala na command line (dotyczy Windows)
2. Zmień nazwę aplikacji na **time\_app.py**
3. Wykorzystując jedną ze zmiennych środowiskowych, spraw aby ta aplikacja była ładowana przez Flask
4. Zdefiniuj też zmienną włączającą debugger

### Sprawdź się!

1. Jak nazywa się zmienna środowiskowa, która definiuje nazwę głównego pliku aplikacji?
2. Jak nazywa się zmienna środowiskowa włączająca tryb debug?
3. Co zrobić, aby Flask przeładowywał automatycznie zmieniony kod zaraz po jego zapisaniu?
4. Jakie polecenie służy do tworzenia zmiennych środowiskowych na Windows, a jakie pod Linuxem?

## Propozycja rozwiązania

```
set FLASK_APP=time_app.py  
set FLASK_DEBUG=1
```

Na Linux

```
export FLASK_APP=time_app.py  
export FLASK_DEBUG=1
```

## Dynamiczny routing

### Notatka

- Routing dynamiczny pozwala na przekazywanie do funkcji stowarzyszonej z route pewne parametry
- Route pozwalający na przyjęcie parametru currency i amount może wyglądać tak:

```
@app.route('/cantor/<currency>/<amount>')
```

- Dynamicznie zmieniane elementy route są umieszczane w nawiasach ostrych <>
- Funkcja stowarzyszona z route musi przyjmować parametry o tej samej nazwie. Funkcja może korzystać z tych parametrów, jak zwykle w Pythonie
- W definicji route można używać znaczników określających, że dynamiczne elementy route mają być określonego typu, np.:

```
@app.route('/cantor/<string:currency>/<int:amount>')
```

- Dopuszczalne wartości identyfikatora typu, z którego można skorzystać w dynamicznym routingu to int, float, string, path (jeśli napis może zawierać znak slash) i UUID (string reprezentujący unikalny identyfikator)

### Laboratorium

1. Wyobraź sobie, że tworzysz oprogramowanie, które będzie wykorzystywane w skomputeryzowanej kuchni w restauracji. Kucharz będzie miał na ekranie wyświetloną instrukcję do wykonania, a kiedy już zrobi to, co należało zrobić, kliknie na ekranie dotykowym „Dalej” i zostanie wyświetlony kolejny krok w przepisie. Linki do kolejnych kroków mogły by wyglądać tak:  
/cook/pancake/1  
/cook/pancake/2  
/cook/pancake/3  
/cook/tomato\_soup/1  
/cook/tomato\_soup/1  
ltd.  
Napisz route i funkcję, która w odpowiedni sposób przyjmie nazwę przepisu oraz krok, który ma być wyświetlony. Funkcja może wyświetlać po prostu przyjęte parametry
2. Przejrzyj stronę dokumentacji Flaska dotyczącą tematu routingu (sekcja Routing i Variable Rules) znajdującą się pod adresem: [Quickstart — Flask Documentation \(1.1.x\) \(palletsprojects.com\)](https://palletsprojects.com/en/1.1.x/quickstart/)

### Sprawdź się!

1. Co stanie się jeśli route oczekuje parametru będącego liczbą, a użytkownik odwoła się do adresu wprowadzając napis?
2. Czy definicja funkcji stowarzyszonej z route musi mieć takie same nazwy parametrów co route? Czy kolejność parametrów musi się zgadzać?
3. Dlaczego warto używać znaczników typów definiując route?

### Propozycja rozwiązania

```
@app.route('/cook/<string:receipt>/<int:step>')
def cook(receipt, step):
    body = f'''<H1>In the receipt {receipt} you are on step {step}</H1>'''
    return body
```

## Przekazywanie parametrów przez Query String

### Notatka

- QueryString to dodatkowy tekst dodawany do adresu strony w postaci:

```
https://example.com/path/to/page?name=ferret&color=purple
```

- Query string jest widoczny w pasku adresu przeglądarki
- Aby pracować z właściwościami żądania wysyłanego z przeglądarki należy zaimportować z Flask obiekt request
- Pełny query string otrzymuje się przez:

```
request.query_string
```

- Aby pobrać wartość konkretnego parametru (np. color) lepiej jest korzystać z

```
request.args['color']
```

- Aby sprawdzić czy dany parametr w ogóle był wysłany przez przeglądarkę, można sprawdzać, czy oczekiwana nazwa parametru występuje w args:

```
if 'color' in request.args:
```

- Aby wyświetlić wszystkie przesłane parametry można napisać pętlę przechodzącą przez tablicę request.args

```
for p in request.args:  
    print(p, request.args[p])
```

Uwaga: bezpośrednie wykorzystanie wartości wprowadzanych przez użytkownika w budowanych instrukcjach SQL lub w zwracanych użytkownikowi wyników może być niebezpieczne. Użytkownik może spreparować query string i uzyskać wynik zupełnie inny niż oczekiwany!

### Laboratorium

1. Dodaj do funkcji z poprzedniego modułu funkcjonalność polegającą na tym, że w query string będzie można przekazać dodatkową właściwość font-size, która wpłynie na wielkość czcionki zastosowanej w zwracanym wyniku (ukryta opcja dla niedowidzącego kucharza):  
<http://27.0.0.1:5000/cook/pomidorowa/133?font-size=200%>
  - a. Wskazówka: w generowanej odpowiedzi skorzystaj w znaczniku H1 stylu css:  
<H1 style="font-size: ...tu coś trzeba napisać...">
  - b. Zadbaj o to, aby można skorzystać ze strony bez wprowadzania parametru w query string

### Sprawdź się!

1. Jaki obiekt należy zaimportować, aby korzystać z query string?
2. Jak uzyskać dostęp do całego query string w surowej postaci, a jak do konkretnych i znanych ci wartości parametrów?

3. Czy potrafisz zhakować rozwiązanie zadania z lab, tak aby przed wyświetlanym tekstem dodatkowo pojawił się napis „I was here!”

#### Propozycja rozwiązania

```
@app.route('/cook/<string:receipt>/<int:step>')
def cook(receipt, step):
    print(request.query_string)
    print('----')
    print(request.args)
    font_size='100%'
    if 'font-size' in request.args:
        font_size = request.args['font-size']
    body = f'''<H1 style="font-size: { font_size }">In the receipt {receipt} you are
on step {step}</H1>'''
    return body
```



## Przyjmowanie danych z formularzy

### Notatka

- Formularze obsługiwane przez Flask, to standardowe formularze pochodzące z języka html. W razie potrzeby dokumentacja poszczególnych kontrolki i przykłady ich wykorzystywania znajdują się między innymi pod adresem: [https://www.w3schools.com/html/html\\_forms.asp](https://www.w3schools.com/html/html_forms.asp)
- Dla formularza najważniejsze atrybuty z punktu widzenia Flask to:
  - Action – na jaki adres należy wysłać wypełniony formularz
  - Method – jaka metoda ma być wykorzystana do wysyłania formularza
- Dla pól input najważniejsze atrybuty to:
  - Type, np. text określa, że pole ma być wyświetlane w przeglądarce jako pole tekstowe
  - Name – nazwa pod jaką wartość pola będzie wysyłana do aplikacji. Często id i name ustawia się na taką samą wartość, ale nie jest to konieczne.
  - Value – domyślna wartość, jaka będzie się znajdować w zmiennej po wyświetleniu formularza
- Wartości pól formularza znajdują się w zmiennej request.form
- Przed skorzystaniem z pola formularza dobrze jest sprawdzić, czy na formularzu takie pole się znajduje:

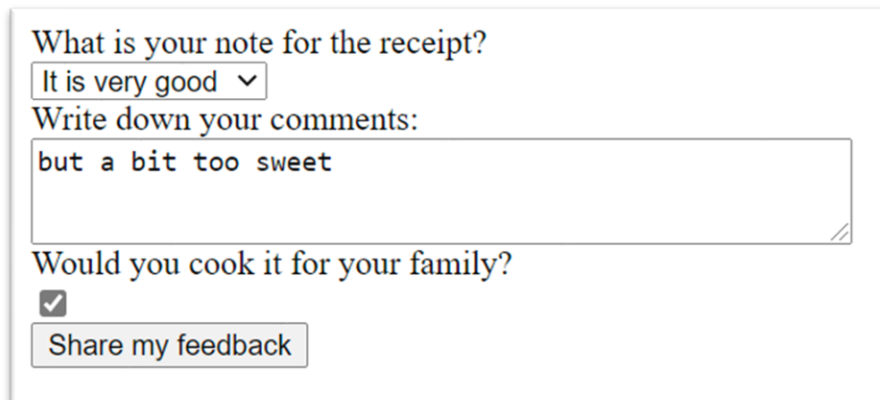
```
if 'currency' in request.form:  
    currcy = request.form['currency']
```

- Aby określona route mogła przyjmować dane z formularza wysyłane metodą POST, to przy definicji route należy dodać parametr route, który jest listą dopuszczalnych metod:

```
@app.route('/exchange_process', methods=['POST'])
```

### Laboratorium

- Twoim zadaniem jest zbudowanie formularza, który zapyta kucharza o jego zdanie na temat przepisu. Ciągłe jeszcze brzydki formularz powinien wyglądać mniej więcej tak:



What is your note for the receipt?

It is very good ▾

Write down your comments:

but a bit too sweet

Would you cook it for your family?

☒

Share my feedback

2. Możesz zacząć programować na własną rękę, albo skorzystać z poniższych dokładniejszych instrukcji:
  - a. Utwórz route `/rate_receipt` i stowarzyszoną funkcję
  - b. Wygeneruj formularz. Składnia języka HTML jest poza zakresem tego kursu, więc nie tłumaczmy go dokładnie. Dla własnej wygody możesz skorzystać z poniższego fragmentu:

```
<form id="rating" action="/rate_receipt_save" method="POST">
  <label for=note>what is your note for the receipt?</label><br>
  <select id="note" name="note">
    <option value="5">It is great!</option>
    <option value="4">It is very good</option>
    <option value="3" selected>It is just good</option>
    <option value="2">It was poor</option>
    <option value="1">It was horrible!</option>
  </select><br>

  <label for=comment>Write down your comments:</label><br>
  <textarea id="comment" name="comment" rows="3" cols="50">
    ...
  </textarea><br>

  <label for="decision">would you cook it for your family?</label><br>
  <input type="checkbox" id="decision" name="decision"><br>

  <input type="submit" value="Share my feedback">
</form>
```

- c. Tak utworzony formularz zwróć do przeglądarki
- d. Utwórz route `/rate_rating_save` i stowarzyszoną funkcję. Route ma akceptować metodę POST.
- e. Przyjmij dane z formularza nadając im wartości domyślne (np. `note=3`, `comment=""`, `decision=False`)  
Uwaga – pole input typu checkbox działa w nieco odmienny sposób niż dla text. Jeśli pole zostało zaznaczone w formularzu, to znajduje się ono w `request.form`. Jeśli jednak pole nie było zaznaczone, to po prostu nie występuje w `request.form`. Jedną z możliwych metod przyjęcia tego parametru to:

```
decision = False
if 'decision' in request.form:
    decision = True
```

- f. Wyświetl odczytane informacje do przeglądarki
3. Przetestuj działanie programu podając różne wartości wejściowe

[Sprawdź się!](#)

1. Jak wskazuje się, że route obsługuje metodę POST?
2. W jakim obiekcie znajdują się wartości wskazane w formularzu przez użytkownika?
3. Jak można sprawdzić, czy określone pole znajdowało się w formularzu?

## Propozycja rozwiązania

```
from flask import Flask, request

# ...

@app.route('/rate_receipt')
def rate_receipt():
    body = '''
        <form id="rating" action="/rate_receipt_save" method="POST">
            <label for=note>What is your note for the receipt?</label><br>
            <select id="note" name="note">
                <option value="5">It is great!</option>
                <option value="4">It is very good</option>
                <option value="3" selected>It is just good</option>
                <option value="2">It was poor</option>
                <option value="1">It was horrible!</option>
            </select><br>

            <label for=comment>Write down your comments:</label><br>
            <textarea id="comment" name="comment" rows="3" cols="50">
            </textarea><br>

            <label for="decision">would you cook it for your family?</label><br>
            <input type="checkbox" id="decision" name="decision"><br>

            <input type="submit" value="Share my feedback">
        ... </form>
    '''
    return body

@app.route('/rate_receipt_save', methods=['POST'])
def rate_receipt_save():
    note = 3
    if 'note' in request.form:
        note = request.form['note']
    comment = ''
    if 'comment' in request.form:
        comment = request.form['comment']
    decision = False
    if 'decision' in request.form:
        decision = True

    message = f'''Your rating was: {note}<br>
                Your comment was: {comment}<br>
                Your decision was {decision}
    '''
    return message
```

Pusta strona wstawiona celowo

## Obsługa GET i POST w jednej funkcji. Obiekt request

### Notatka

- Często generowanie formularza i przyjmowanie wartości z tego formularza jest obsługiwane po stronie aplikacji przez jedną funkcję
- Aby dana funkcja pracowała w tych dwóch scenariuszach, należy dodać do dekoratora

```
@app.route(parametr method=['GET', 'POST'])
```

- Do rozróżnienia, czy funkcja jest wywołana po raz pierwszy metodą GET, czy po raz kolejny metodą POST można posłużyć się właściwością `method` dla obiektu `request`:

```
@app.route('/action', methods = ['GET', 'POST'])
def action():
    if request.method == 'POST':
        # generate and return a form
    else:
        # process the form fields
```

- Dokumentacja dla kontekstu `request`:  
<https://flask.palletsprojects.com/en/1.1.x/reqcontext/>
- Dokumentacja dla API `request`:  
<https://flask.palletsprojects.com/en/1.1.x/api/#flask.request>
- Chociaż z obiektu `request` korzysta się jak ze zmiennej globalnej, to jednak nie jest to zmienna globalna. To kontekst pojedynczego żądania wysyłanego z przeglądarki do aplikacji
- Jeden z typowych błędów popełnianych przy definiowaniu formularzy, to pominięcie parametru `methods` w dekoratorze `app.route`

### Laboratorium

1. Zmień definicję funkcji `rate_receipt` tak, aby zarówno generowanie formularza, jak i jego obsługa były w jednej funkcji

### Sprawdź się!

1. Jakie najważniejsze informacje można znaleźć w obiekcie `request`?
2. Jak rozpoznać, czy funkcja jest wywoływana w skutek żądania GET czy POST?
3. Co należy dodać do dekoratora, aby jedna funkcja mogła jednocześnie generować formularz przy żądaniu GET i przyjmować jego dane przy żądaniu POST?

## Propozycja rozwiązania

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/rate_receipt', methods=['GET', 'POST'])
def rate_receipt():
    if request.method == 'GET':
        body = '''
            <form id="rating" action="/rate_receipt" method="POST">
                <label for=note>what is your note for the receipt?</label><br>
                <select id="note" name="note">
                    <option value="5">It is great!</option>
                    <option value="4">It is very good</option>
                    <option value="3" selected>It is just good</option>
                    <option value="2">It was poor</option>
                    <option value="1">It was horrible!</option>
                </select><br>

                <label for=comment>Write down your comments:</label><br>
                <textarea id="comment" name="comment" rows="3" cols="50">
                </textarea><br>

                <label for="decision">would you cook it for your family?</label><br>
                <input type="checkbox" id="decision" name="decision"><br>

                <input type="submit" value="Share my feedback">
            ... </form>
        '''
        return body
    else:
        note = 3
        if 'note' in request.form:
            note = request.form['note']
        comment = ''
        if 'comment' in request.form:
            comment = request.form['comment']
        decision = False
        if 'decision' in request.form:
            decision = True

        message = f'''Your rating was: {note}<br>
                    Your comment was: {comment}<br>
                    Your decision was {decision}
        '''
        return message

# ...
```

## Funkcja url\_for – Twój przyjaciel w budowaniu linków

### Notatka

- Nawigacja pomiędzy poszczególnymi stronami (widokami) w aplikacji może polegać na ręcznym budowaniu odnośników html

```
<a href=""> ... </a>
```

- Żeby zachować możliwość zmiany nazw route w aplikacji, atrybucie href można użyć dynamicznie generowany adres korzystając z funkcji url\_for
- url\_for należy zaimportować z modułu Flask
- Jedyny wymagany parametr w tej metodzie to nazwa funkcji widoku, która jest powiązana z route, np.:

```
<a href="{ url_for('exchange') }"> ... </a>
```

- Aby przekazać do docelowej funkcji parametry (np. wymagane przez dynamiczny routing), to zapisujemy je w postaci nazwa\_argumentu = wartość argumentu

```
<a href="{ url_for('cantor', currency='CHF', amount=50) }"> ... </a>
```

- Aby wygenerować adres względny należy skorzystać z dodatkowego parametru \_external=True

```
<a href="{ url_for('cantor', currency='CHF', amount=50, _external=True) }"> ... </a>
```

### Laboratorium

1. W poprzednim zadaniu została zbudowana funkcja, która mogła komunikować się z serwerem metodą GET i POST. Jeśli wywołano ją metodą GET, to funkcja generowała formularz. Ten formularz atrybutem action wskazuje obecnie na sztywno wpisany route.
2. Zmień ten odnośnik na generowany dynamicznie z wykorzystaniem funkcji url\_for
3. 😊 Jeśli jeszcze tego nie zrobiłeś/aś, to „weź się pomyśl” i zamiast nazwy funkcji, jako argument prześlij route 😊

### Sprawdź się!

1. Co jest jedynym wymagany argumentem w funkcji url\_for?
2. Jaki argument dodać do url\_for, aby został wygenerowany adres bezwzględny (zawierający adres domeny)?
3. Załóżmy, że generowany przez url\_for odnośnik ma wskazywać na funkcję, która przyjmuje pewne argumenty. W jaki sposób przekazać te argumenty?

## Propozycja rozwiązania

```
from flask import Flask, request, url_for
app = Flask(__name__)
@app.route('/rate_receipt', methods=['GET', 'POST'])
def rate_receipt():
    if request.method == 'GET':
        body = f'''
            <form id="rating" action="{ url_for('rate_receipt') }" method="POST">
#... etc
```



## Funkcja redirect, czyli “zapraszamy do kasy obok”

### Notatka

- Polecenie redirect może być zastosowane np. w celu przekierowania użytkownika na stronę logowania, jeśli użytkownik sięga do zastrzeżonej części aplikacji, ale jeszcze nie jest zalogowany
- Funkcję redirect należy zaimportować z modułu Flask
- Aby przenieść użytkownika w inną część aplikacji należy w funkcji widoku zwrócić wynik redirect:

```
return redirect(url_for('index'))
```

- Podczas przekierowywania użytkownika, dochodzi do dodatkowej komunikacji między klientem (przeglądarką) a serwerem
- Podczas przekierowywania, można korzystać z opcji oferowanych przez url\_for, co przyda się np. wtedy, gdy przekierowanie ma być wykonane na adres konstruowany dynamicznie:

```
return redirect(url_for('cantor', currency=currency, amount=amount))
```

### Laboratorium

1. Podczas budowania aplikacji, często zaczyna się od prototypu, który może prezentować się niespecjalnie ładnie i na dodatek... zupełnie nie działać. Takie prototypy pozwalają jednak wyobrazić sobie sposób w jaki aplikacja będzie działać. Właśnie zostałeś poproszony o przygotowanie funkcji do takiego niedziałającego i brzydkiego prototypu!
2. Przygotuj route /not\_implemented/<message>. Funkcja powiązana z tą route ma po prostu wyświetlić na ekranie message
3. Ilekroć w aplikacji programista będzie chciał zaznaczyć, że dana funkcja nie jest jeszcze do końca gotowa, będzie przekierowywał do route /not\_implemented przekazując odpowiedni komunikat
  - a. Przygotuj route i funkcję new\_receipt. Ponieważ ta funkcja jeszcze nie jest gotowa to przekieruj użytkownika do route /not\_implemented z komunikatem „Function new\_receipt is not ready yet”.
  - b. Podobnie przygotuj funkcję i route /delete\_receipt/<name> i tak jak poprzednio w przypadku odwołania do tej route, przekieruj użytkownika do /not\_implemented z komunikatem „Function delete\_receipt is not ready yet”

### Sprawdź się!

1. Przekierowując użytkownika korzystasz z funkcji redirect, ale jedna z „best practice” mówi też, że warto skorzystać z jeszcze jednej pomocniczej funkcji. Jakiej?
2. Czy przekierowując użytkownika na dynamic route, dodatkowo trzeba przekazywać jakieś parametry bezpośrednio do funkcji redirect?

## Propozycja rozwiązania

```
from flask import Flask, url_for, redirect
app = Flask(__name__)
@app.route('/not_implemented/<message>')
def not_implemented(message):
    return '<h1 style="color:red">{}</h1>'.format(message)
@app.route('/new_receipt')
def new_receipt():
    return redirect(url_for('not_implemented', message="Function new_receipt is not
ready yet"))
@app.route('/delete_receipt/<name>')
def delete_receipt(name):
    return redirect(url_for('not_implemented', message="Function new_receipt is not
ready yet"))
```

## Statyczne elementy projektu

### Notatka

- Elementy statyczne zazwyczaj gromadzimy w dedykowanym do tego katalogu, którym domyślnie we Flasku jest katalog static
- Aby wyświetlić element graficzny należy wskazać na niego odpowiednim znacznikiem HTML
- Aby zbudować łącze do elementu statycznego skorzystaj ze znanej funkcji `url_for`
  - Pierwszy argument tej funkcji to słowo static (nazwa folderu)
  - Drugi argument (nazwany jako `filename`) to nazwa pliku znajdującego się w katalogu static
  - Jeśli w katalogu static miałyby się znajdować podkatalogi, to drugi argument powinien zawierać nazwę podkatalogu i pliku:

```
url_for('static', filename='images/logo.png')
```

- Gdy chcesz uzyskać dostęp do lokalnej ścieżki zasobu znajdującego się w katalogu static skorzystaj z:
  - Właściwości `static_folder` w obiekcie `app`
  - Nazwy pliku
  - Funkcji `path.join` z modułu `os`:

```
os.path.join(app.static_folder, 'data/inventory.txt')
```

### Laboratorium

1. Twoja aplikacja ma udostępniać do pobrania kilka plików statycznych. Przygotuj najpierw odpowiednie katalogi:
  - a. Dodaj katalog static do projektu
  - b. W tym katalogu załóż podkatalog `download`
  - c. W tym katalogu umieść kilka plików: może to być plik tekstowy lub obraz
2. Dodaj route i funkcję do adresu `/download/<file>`  
File będzie odpowiadać za plik, który mamy udostępnić użytkownikowi
3. W funkcji skojarzonej z tym route:
  - a. Utwórz zmienną `subpath` będącą połączeniem nazwy katalogu `download`, znaku slash (/) i przekazanej zmiennej `file`
  - b. W zmiennej `local_path_name` zapamiętaj ścieżkę lokalną do potencjalnego pliku, który ma być zwrócony użytkownikowi (nie martw się o pomieszczenie znaków backslash i slash na Windows – system sobie z tym poradzi)
    - i. Jeśli plik określony przez `local_path_name` istnieje, to przekieruj użytkownika, do adresu tego pliku
    - ii. W przeciwnym razie wyświetl komunikat o błędzie

### Sprawdź się!

1. Jak domyślnie nazywa się katalog, w którym umieszcza się statyczne obrazy wykorzystywane przez aplikację?
2. Co przechowuje właściwość `static_folder` w obiekcie `app`?
3. Jaka funkcja pozwala wygenerować webowy odnośnik do statycznego zasobu? Jakie ma parametry?

### Propozycja rozwiązania

```
from flask import Flask, url_for, redirect
import os

app = Flask(__name__)

@app.route('/download/<file>')
def download(file):
    subpath = 'download/'+file
    local_file_path = os.path.join(app.static_folder, subpath)

    if(os.path.isfile(local_file_path)):
        print('file found')
        return redirect(url_for('static', filename=subpath))
    else:
        print('sorry file not found')
        return 'File not found!'
```

## Definiowanie środowiska Flask

### Notatka

- Kilka najważniejszych zmiennych środowiskowych Flask:
  - FLASK\_APP – nazwa skryptu programu
  - FLASK\_DEBUG – włączenie trybu debug aplikacji
  - FLASK\_ENV – definiuje środowiska np. jako development (wielkość liter jest istotna)
  -
- Kilka najważniejszych działań skryptu flask:
  - flask run  
uruchamia proces web serwera udostępniającego aplikację Flask
  - flask routes  
wyświetla zestawienie route i endpoint i metody
  - flask run –host 0.0.0.0  
flask nasłuchuje na wszystkich adresach IP przypisanych do komputera
- Tryb debug – uruchamia środowisko do przeprowadzenia debugowania w interfejsie przeglądarki, powoduje automatyczne przeładowanie serwera www po zapisaniu zmiany w kodzie aplikacji
- Tryb development – środowisko jest rozpoznawane jako developerskie i został włączony tryb debug a wraz z nim cechy związane z typem debug
- Zmienne środowiskowe można zdefiniować w skrypcie definiującym środowisko wirtualne:
  - Na Windows `.\venv\scripts\activate.bat` lub `.\venv\scripts\activate.ps1`
  - Na Linux `./venv/bin/activate`
- Polecenie definiujące zmienne środowiskowe to:
  - Na Windows `SET FLASK_ENV=development`
  - Na Linux `EXPORT FLASK_ENV=development`

### Laboratorium

1. Poprzez modyfikację skryptu activate, zdefiniuj środowisko jako development
2. Wyświetl route wykorzystywane przez aplikację

### Sprawdź się!

1. Jaki wpływ na działanie programu ma ustawienie zmiennej FLASK\_DEBUG?
2. Jak można zdefiniować środowisko produkcyjne i deweloperskie?
3. Jak można wyświetlić help dla skryptu flask?

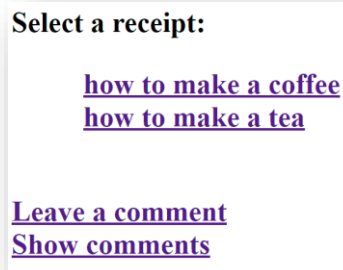
### Propozycja rozwiązania

w pliku `activate` dodaj: `SET FLASK_ENV=development` lub `EXPORT FLASK_ENV` an `Linuxie`  
`flask routes`

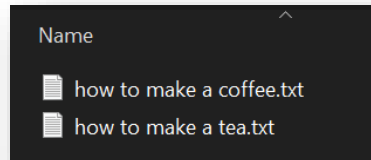
## Projekt – 01 – Cook book

Przerabiając materiał do tej pory możesz stwierdzić „uczę się uczyć i nie wiem co umiem”. Jeśli chcesz, to możesz napisać swój pierwszy prosty programik. Będzie pewnie (sorry za to) brzydki i toporny ale jednocześnie działający! A ponieważ to w końcu TWOJE DZIEŁO, więc zamiast widzieć w nim wady zobacz w nim dobre rzeczy. To zadanie jest absolutnie nieobowiązkowe i jest tylko propozycją pewnej aktywności, którą możesz wykonać, aby poczuć, że już coś umiesz. Możesz na tym etapie stworzyć też coś całkiem innego – Twój wybór.

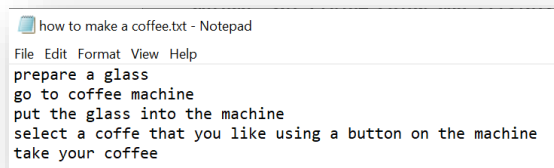
O co chodzi? Piszemy aplikację dla kucharza w snak-barze. Na stronie głównej wyświetlimy menu:



Pozycje tego menu pochodzą z plików tekstowych umieszczonych w katalogu ./static/receipts:



Nazwa pliku to jednocześnie nazwa przepisu. Każdy z tych plików zawiera jeden krok przepisu, np.:



Po wybraniu przepisu jest prezentowana strona z pierwszym krokiem do wykonania. Oprócz tego jest małe menu pozwalające na

- przejście do strony głównej
- przejście do kolejnego kroku (jeśli prezentowany krok jest ostatni, to tą pozycję można ukryć lub stale przenosić do ostatniego kroku)

- przejście do poprzedniego kroku (jeśli prezentowany jest krok pierwszy, to można tą pozycję ukryć lub stale przenosić do pierwszego kroku)

## The step 1 for receipt how to make a coffee:

prepare a glass

- [Home](#)
- [Next](#)
- [Previous](#)

Oprócz tego na głównej stronie aplikacji mamy możliwość przejścia do strony formularza pozwalającego wprowadzić własny komentarz formularzem:

Send and return home

Kiedy użytkownik wprowadzi swoją opinię, to:

- formularz należy odesłać do tej samej funkcji, która go wygenerowała
- w przesłanych danych należy zamienić znak < na &lt; a znak > na &gt; (pozwoli to wykluczyć potencjalne znaczniki html wprowadzone przez złośliwego użytkownika)
- zapisać te dane w pliku ./static/comments/comments.txt
- przenieść użytkownika do strony głównej

Jeśli na stronie głównej zostanie wybrany odnośnik „Show comments”, to należy wyświetlić dane z pliku comments.txt

## The comments

- I like Flask a lot! It is so nice!
- Are you blocking <b>HTML tags</b>?
- Wow, I can send data from web browser to local files!!!

[Back to home](#)



Proponowane route aplikacji:

Endpoint	Methods	Rule
comment	GET, POST	/comment
index	GET	/
receipt	GET	/receipt/<name>/<int:step>
show_comments	GET	/show_comments
static	GET	/static/<path:filename>

Pisząc ten mały programik pewnie w pewnych momentach natkniesz się na to, że czegoś nie wiesz, np. jak wylistować zawartość katalogu, jak otworzyć lub zapisać plik, jak zamienić fragment tekstu na inny, jak wstawić pole tekstowe (textarea) do formularza itp. Ponieważ na tym kursie skupiamy się na Flasku, to rzeczywiście nie opowiedzieliśmy tu o tych innych zagadnieniach. W takim przypadku poszukaj odpowiednich funkcji korzystając z wyszukiwarki. Nie ma na świecie programisty, który wiedział by wszystko – każdy programista szuka rozwiązań na własną rękę.

Na własną rękę możesz też:

- upiększyć rozwiązanie 😊
- dodać więcej pól do formularza
- zadbać o obsługę błędów (np. kiedy pliku nie daje się otworzyć)
- rozbudować funkcjonalność

Pamiętaj, każda samodzielnie napisana linijka kodu i każdy samodzielnie rozwiązany problem zbliża Cię do celu jakim jest opanowanie Flask-a i Pythona!

Pusta strona wstawiona celowo

## Szablony Jinja

### Notatka

- Szablony pozwalają oddzielić od siebie logikę biznesową tworzonej aplikacji od jej interfejsu graficznego
- Szablony są plikami html zapisywanymi w katalogu Templates
- Aby przesłać zapisaną stronę do przeglądarki, funkcja powinna zwrócić wynik wywołania funkcji `render_template`:

```
return render_template('my_form.html')
```

- Kiedy w szablonie chcesz umieścić zawartość dynamiczną, to należy ją umieścić w podwójnych nawiasach klamrowych:

```
<h1>Hello {{ name }}</h1>
<form id="my_form", method="POST" action="{{ url_for('show_form')}}">
```

- Aby szablon “wiedział”, co wstawić w miejscu zmiennych umieszczonych w podwójnych nawiasach klamrowych, należy je przesłać jako kolejne parametry funkcji `render_template` w postaci `nazwa_zmiennej_na_szablonie=nazwa_zmiennej_w_programie`:

```
return render_template('my_form.html', name=name)
```

### Laboratorium

1. Przygotuj prosty formularz pozwalający na opisanie usterki w pokoju hotelowym. Przykładowe pola:
  - a. Numer pokoju (`room_number`)
  - b. Nazwisko zgłaszającego (`guest_name`)
  - c. Opis usterki (`notification`) – uwaga – w Pythonie nazwa `Notification` może też być wykorzystywana systemowo, jeśli chcesz uniknąć konfliktów możesz rozważyć stosowanie innej nazwy)Umieść formularz w szablonie `notification.html`
2. Przygotuj statyczną stronę `notification_content.html` wyświetlającą opis i wartość w/w pól
3. Napisz funkcję `notification` związaną z route `/notification`, która będąc wywołana przez metodę GET wyświetli zrenderowany formularz z `notification.html`, a przy wywołaniu przez POST wyświetli zrenderowaną stronę `notification_content.html`

### Sprawdź się!

1. W jakim katalogu zapisuje się szablony?
2. Jaka funkcja przetwarza szablony i przesyła go do użytkownika?
3. Jak przekazać dynamiczne parametry wymagane do renderowania zwracanej strony?

## Propozycja rozwiązania

```
# templates/notification.html
<!DOCTYPE html>
<html>
  <head>
    <title>Hotel 101</title>
  </head>
  <body>
    <form id="notification_form" method="POST" action="{{ url_for('notification') }}">
      <label for="room_number">Room number</label>
      <input type="text" name="room_number" id="room_number"><br>
      <label for="guest_name">Guest name</label>
      <input type="text" name="guest_name" id="guest_name"><br>
      <label for="notification_text">Notification</label>
      <textarea rows="4" cols="50" name="notification_text"></textarea><br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>

# templates/notification_content.html
<!DOCTYPE html>
<html>
  <head>
    <title>Hotel 101</title>
  </head>
  <body>
    <h1>Notification content</h1>
    <ul>
      <li style="font-weight: bold;">Room number</li>
      <li>{{ room_number }}</li>
      <li style="font-weight: bold;">Guest name</li>
      <li>{{ guest_name }}</li>
      <li style="font-weight: bold;">Notification</li>
      <li>{{ notification_text }}</li>
    </ul>
  </body>
</html>

# app.py
from flask import Flask, url_for, request, render_template

app = Flask(__name__)

@app.route('/notification', methods=['GET', 'POST'])
def notification():
    if request.method == 'GET':
        return render_template('notification.html')
    else:
        room_number = request.form['room_number'] if 'room_number' in request.form
        guest_name = request.form['guest_name'] if 'guest_name' in request.form else ''
        notification_text = request.form['notification_text'] if 'notification_text' in request.form else ''

        return render_template('notification_content.html',
                                room_number=room_number, guest_name=guest_name,
                                notification_text=notification_text)
```

## Warunkowe wyświetlanie części szablonu Jinja

### Notatka

- Dobrą praktyką jest staranie się rozdzielenie kodu aplikacji od interfejsu graficznego aplikacji, dlatego jeśli tylko się da, kod powinien się znajdować w skrypcie, a elementy języka HTML w szablonach Jinja
- Jeśli z jakiegoś powodu nie można tego zrobić, to może się przydać warunkowe wyświetlanie fragmentów szablonu Jinja
- Wyrażenie warunkowe Jinja wygląda następująco:

```
{% if currency=='EUR' %}  
...  
{% elif currency=='USD' %}  
...  
{% endif %}
```

- Ogólnie przy wyrażeniu if, w Jinja obowiązują podobne reguły, jak w Pythonie, np. elseif lub else są opcjonalne

### Laboratorium

1. W tym zadaniu zmodyfikujesz formularz z poprzedniego zadania. Zgłaszając awarię, będzie można określić priorytet zdarzenia:
  - a. Dodaj pole select o nazwie priority, przyjmujące wartości high, medium, normal
  - b. Tekst wyświetlany dla tych priorytetów to „HIGH”, „MEDIUM” i „NOT URGENT”
  - c. Domyślnie ma być wybrany normal
2. Przyjmij nowe pole formularza w funkcji wywoływanej po stronie Pythona i przekaz go do kolejnego formularza w wywołaniu funkcji render\_template
3. W szablonie wyświetlającym nowe zgłoszenie, warunkowo wyświetl w nagłówku H1:
  - a. “Critical notification content” dla priority high
  - b. “Important notification content” dla priority medium
  - c. “Notification content” dla priority normal

### Sprawdź się!

1. Z jakiego powodu tak bardzo zależy nam na rozdzieleniu kodu od interfejsu?
2. Czy wyrażenie {% elif %} jest wymagane w prostej postaci if?
3. Czy wyrażenie {% endif %} jest wymagane w prostej postaci if?

## Propozycja rozwiązania

```
# template notification.html (only added part)
    <label for="priority">Priority</label>
    <select name="priority" id="priority">
        <option value="high">HIGH PRIORITY</option>
        <option value="medium">MEDIUM</option>
        <option value="normal" selected>NOT URGENT</option>
    </select><br>

# ----- getting new field value from the request (only modified part)
@app.route('/notification', methods=['GET', 'POST'])
def notification():
    if request.method == 'GET':
        return render_template('notification.html')
    else:
        room_number = request.form['room_number'] if 'room_number' in request.form
    else ''
        guest_name = request.form['guest_name'] if 'guest_name' in request.form else
    ''
        notification_text = request.form['notification_text'] if 'notification_text'
in request.form else ''
        priority = request.form['priority'] if 'priority' in request.form else
'normal'

        return render_template('notification_content.html',
                                room_number=room_number, guest_name=guest_name,
                                notification_text=notification_text, priority=priority)

# template notification_confirmation.html. Only added part
<body>
    {% if priority=='high' %}
        <h1>Critical notification content</h1>
    {% elif priority=='medium' %}
        <h1>Important notification content</h1>
    {% else %}
        <h1>Notification content</h1>
    {% endif %}
```

## Pętle w szablonach Jinja

### Notatka

- Wszystkie metody i funkcjonalności, które znasz, jak np. ładowanie modułów, budowanie klas itp. działają również we Flask. Skrypty mogą np. pracować na odpowiednio przygotowanych klasach i obiektach
- Kiedy chcesz przetestować działanie klas w skrypcie app.py, to możesz zaimportować te klasy:

```
from app import MyClass
```

- Kiedy w szablonie Jinja chcesz wykonać pewną czynność wielokrotnie możesz posłużyć się wyrażeniem for. Polecenie to będzie iterować przez obiekt dostarczony w funkcji render\_template:

```
{% for currency in list_of_currencies %}  
... put something here...  
{%endfor %}
```

- Takie sprytne połączenie obiektu w kodzie aplikacji z dynamicznie budowanym w Jinja dokumentem HTML, pozwala wyeliminować statyczne elementy aplikacji.

### Laboratorium

1. W poprzednim zadaniu dodaliśmy do formularza pole określające priorytet zgłoszenia. Tutaj wyeliminujemy wbudowane w szablon statyczne priorytety
2. Zdefiniuj klasę PriorityType o polach: code, description i selected
3. Zdefiniuj klasę NotificationPriorities:
  - a. W metodzie init dodaj do self pustą listę list\_of\_priorities
  - b. W metodzie load\_priorities wypełnij listę priorytetami zgodnymi z poprzednim lab
  - c. Dodaj metodę get\_priority\_by\_code pozwalającą odszukać priorytet znając jego kod
4. W funkcji notification
  - a. utwórz obiekt notification\_priorities jako instancję klasy NotificationPriorities
  - b. wypełnij ten obiekt wywołując metodę load\_priorities.
  - c. przekaż ten obiekt do szablonu Jinja w funkcji render\_template
5. W szablonie notification.html napisz pętlę for, zastępującą poprzednio zdefiniowane statyczne definicje priorytetów.
6. W funkcji notification, przy wywołaniu metodą post,
  - a. znajdź obiekt PriorityType w oparciu o przesłaną wartość priority.
  - b. znaleziony obiekt przekaż w render\_template do szablonu notification\_content.html
7. Zaktualizuj szablon notification\_content.html tak, aby pracował z obiektem PriorityType
8. Dodaj priorytet 'low' z opisem „REMARK” do listy priorytetów w metodzie load\_priorities klasy NotificationPriorities. Aplikacja będzie go obsługiwać bez dalszych zmian kodu lub szablonów

Jeśli nie znasz programowania obiektowego, po prostu skopiuj ten fragment z rozwiązania!

### Sprawdź się!

1. Dlaczego warto korzystać z instrukcji for w szablonie Jinja?  
Jak wygląda składnia polecenia for?

## Propozycja rozwiązania

app.py:

```
from flask import Flask, url_for, request, redirect, render_template
import os

app = Flask(__name__)

class PriorityType:
    def __init__(self, code, description, selected):
        self.code = code
        self.description = description
        self.selected = selected

class NotificationPriorities:
    def __init__(self):
        self.list_of_priorities = []

    def load_priorities(self):
        self.list_of_priorities.append(PriorityType('high', 'HIGH PRIORITY', False))
        self.list_of_priorities.append(PriorityType('medium', 'MEDIUM', False))
        self.list_of_priorities.append(PriorityType('normal', 'NOT URGENT', True))
        self.list_of_priorities.append(PriorityType('low', 'REMARK', False))

    def get_priority_by_code(self, code):
        for p in self.list_of_priorities:
            if p.code == code:
                return p
        return PriorityType('normal', 'NOT URGENT', True)

@app.route('/notification', methods=['GET', 'POST'])
def notification():
    notification_priorities = NotificationPriorities()
    notification_priorities.load_priorities()

    if request.method == 'GET':
        return render_template('notification.html',
                               list_of_priorities=notification_priorities.list_of_priorities)
    else:
        room_number = request.form['room_number'] if 'room_number' in request.form else ''
        guest_name = request.form['guest_name'] if 'guest_name' in request.form else ''
        notification_text = request.form['notification_text'] if 'notification_text' in
request.form else ''
        priority = request.form['priority'] if 'priority' in request.form else 'normal'

        priority_type = notification_priorities.get_priority_by_code(priority)
        print('found', priority_type.code)

        return render_template('notification_content.html',
                               room_number=room_number, guest_name=guest_name,
                               notification_text=notification_text, priority_type=priority_type)
```



notification.html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hotel 101</title>
  </head>
  <body>
    <form id="notification_form" method="POST" action="{{ url_for('notification') }}">
      <label for="room_number">Room number</label>
      <input type="text" name="room_number" id="room_number"><br>
      <label for="guest_name">Guest name</label>
      <input type="text" name="guest_name" id="guest_name"><br>
      <label for="notification_text">Notification</label>
      <textarea rows="4" cols="50" name="notification_text"></textarea><br>
      <label for="priority">Priority</label>
      <select name="priority" id="priority">
        {% for priority in list_of_priorities %}
          <option value="{{ priority.code }}" {{ 'selected' if priority.selected }}>{{
priority.description }}</option>
        {% endfor %}
      </select><br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>
```

notification\_confirmation.html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hotel 101</title>
  </head>
  <body>
    {% if priority_type.code=='high' %}
      <h1>Critical notification content</h1>
    {% elif priority_type.code=='medium' %}
      <h1>Important notification content</h1>
    {% else %}
      <h1>Notification content</h1>
    {% endif %}
    <ul>
      <li style="font-weight: bold;">Room number</li>
      <li>{{ room_number }}</li>
      <li style="font-weight: bold;">Guest name</li>
      <li>{{ guest_name }}</li>
      <li style="font-weight: bold;">Notification</li>
      <li>{{ notification_text }}</li>
    </ul>
  </body>
</html>
```

Pusta strona wstawiona celowo

## Funkcja flash() – czyli krótkie info o statusie

### Notatka

- flash() służy do zapisania w cookie komunikatu, który powinien zostać wyświetlony poprzez wykorzystanie get\_flashed\_messages(). Funkcję należy zaimportować z modułu flash
- Funkcję flash można wywołać w dowolnym miejscu aplikacji:

```
flash('Record has been saved')
```

- Odłożone do wyświetlenia wiadomości można pobrać w szablonie Jinja poprzez wywołanie get\_flashed\_messages()

```
{% for message in get_flashed_messages() %}  
  <div>{{ message }}</div>  
{% endfor %}
```

- Do obsługi plików cookie konieczne jest zdefiniowanie tzw. SECRET\_KEY. Definiuje się go w słowniku app.config:

```
app.config['SECRET_KEY']='a_secret_string'
```

### Laboratorium

1. Wracamy do zgłoszeń o awariach w hotelu.
2. Podczas przyjmowania zgłoszenia wyświetl na stronie dodatkową informację „Notification has been sent”
3. Wprowadzamy kolejne „usprawnienie”. Jeśli zgłoszenie o statusie ‘medium’ jest zgłaszane w nocy, to priorytet ma się automatycznie zmienić na high. (Testując program dopasuj krytyczne godziny do aktualnej pory dnia). Podczas takiej konwersji priorytetu należy dodatkowo wyświetlić informację „Rising priority from medium to high”

### Sprawdź się!

1. Jak definiuje się SECRET\_KEY?
2. Jaka funkcja zapisuje krótkie informacje o statusie, a jaka je odczytuje?

## Propozycja rozwiązania

```
# in the app.py
from flask import flash
from datetime import datetime

# ...

app = Flask(__name__)
app.config['SECRET_KEY'] = '123GoniszTy!'

# ...

# somewhere in the function notification()

    flash('Notification has been sent')

    the_hour = datetime.now().hour
    raise_priority = (the_hour >= 20 or the_hour < 6) and priority == 'medium'

    if raise_priority:
        priority = 'high'
        flash('Rising priority from medium to high')

# in the nothification_content.html

{% for message in get_flashed_messages() %}
    <div>{{ message }}</dev>
{% endfor %}
```

## Makra Jinja

### Notatka

- Makro to rodzaj funkcji dostępnej w szablonie Jinja
- Makra wykorzystuje się do generowania powtarzalnego kodu html
- Makra zapisuje się w pliku html w katalogu templates, np.:

```
{% macro show_message(message) %}  
<div style="color: red">{{ message }}</div>  
{% endmacro %}
```

- Aby skorzystać z makra należy je zaimportować:

```
{% from "macros.html" import show_message %}
```

- Aby wywołać makro stosujesz podwójny nawias klamrowy

```
{{ show_message('Data has been saved!') }}
```

- Plik z makrami może zawierać definicję wielu makr, a makra te mogą siebie nawzajem wywoływać

### Laboratorium

1. Utwórz makro wyświetlające wiadomości flash w postaci listy:

```
<ul>  
<li>Message 1</li>  
<li>Message 2</li>  
</ul>
```

### Sprawdź się!

1. Dynamiczne elementy w szablonach Jinja wprowadza się poprzez znaczniki {% %} lub {{ }}. Kiedy korzystamy z której formy zapisu?
2. Gdzie należy umieścić plik z makrami?
3. Co trzeba zrobić, aby skorzystać w szablonie z makr znajdujących się w innym pliku?
4. W jakich sytuacjach wyobrażasz sobie korzystanie z makr?

## Propozycja rozwiązania

```
# file macros.html

{% macro show_flash() %}
  <ul>
    {% for message in get_flashed_messages() %}
      <li>{{ message }}</li>
    {% endfor %}
  </ul>
{% endmacro %}

# file notification_content.html

{% from "macros.html" import show_flash %}
<!DOCTYPE html>
<html>
  <head>
    <title>Hotel 101</title>
  </head>
  <body>

    {{ show_flash() }}

    {% if priority_type.code=='high' %}
      <h1>Critical notification content</h1>
    {% elif priority_type.code=='medium' %}
      <h1>Important notification content</h1>
    {% else %}
      <h1>Notification content</h1>
    {% endif %}
    <ul>
      <li style="font-weight: bold;">Room number</li>
      <li>{{ room_number }}</li>
      <li style="font-weight: bold;">Guest name</li>
      <li>{{ guest_name }}</li>
      <li style="font-weight: bold;">Notification</li>
      <li>{{ notification_text }}</li>
    </ul>
  </body>
</html>
```

## Dziedziczenie w szablonach i budowanie własnych standardów

### Notatka

- Dziedziczenie szablonów pozwala na ustandaryzowanie interfejsu aplikacji oraz na powtórne wykorzystanie raz zdefiniowanych elementów kodu html
- Szablon rodzicielski to zwykły szablon Jinja, w którym pozostawione są wspólne elementy dla wszystkich stron potomnych oraz bloki, które mogą być dynamicznie podmieniane przez strony potomne:

```
{% block page_body %}    {% endblock %}
```

- Strony potomne:
  - Dziedziczą ze strony rodzicielskiej

```
{% extends "base.html" %}
```

- Wypełniają bloki swoimi danymi

```
{% block page_body %} This is my page content {% endblock %}
```

- Jeśli blok zdefiniowany w rodzicu, posiadał już jakąś zawartość, to podczas generowania strony potomnej oryginalna zawartość strony jest zastępowana nową zawartością zdefiniowaną na stronie potomnej.
- Jeśli z jakiś powodów, chcesz, aby oryginalna zawartość strony zdefiniowana w bloku u rodzica pozostała w stronie potomnej, to w bloku u potomka należy wywołać funkcję `super()`:

```
{% block page_body %}  
    {{ super() }}  
    This is my page content  
{% endblock %}
```

### Laboratorium

1. Przebuduj swoją aplikację, tak aby wykorzystywała dziedziczenie szablonów:
  - a. Szablon rodzicielski nazwij „base.html”
  - b. Zdefiniuj co najmniej jeden blok w stronie rodzicielskiej o nazwie `page_body`.
  - c. Strony potomne powinny w tym bloku umieszczać swoją zawartość, tj.:
    - i. Formularz w przypadku `notification.html`
    - ii. Potwierdzenie przyjęcia zgłoszenia w przypadku `notification_content.html`

### Sprawdź się!

1. Jakie instrukcje definiują bloki u rodzica i u potomka?
2. Jakie polecenie powoduje, że strona potomna dziedziczy z rodzica?
3. Jakie jest działanie polecenia `super()`

## Propozycja rozwiązania

```
# base.html :

{% from "macros.html" import show_flash %}
<!DOCTYPE html>
<html>
  <head>
    <title>Hotel 101</title>
  </head>
  <body>
    {{ show_flash() }}

    {% block page_body %}
    {% endblock %}
  </body>
</html>

# notification.html :

{% extends "base.html" %}

{% block page_body %}
  <form id="notification_form" method="POST" action="{{ url_for('notification') }}">
    <label for="room_number">Room number</label>
    <input type="text" name="room_number" id="room_number"><br>
    <label for="guest_name">Guest name</label>
    <input type="text" name="guest_name" id="guest_name"><br>
    <label for="notification_text">Notification</label>
    <textarea rows="4" cols="50" name="notification_text"></textarea><br>
    <label for="priority">Priority</label>
    <select name="priority" id="priority">
      {% for priority in list_of_priorities %}
        <option value="{{ priority.code }}" {{ 'selected' if
priority.selected }}>{{ priority.description }}</option>
      {% endfor %}
    </select><br>
    <input type="submit" value="Send">
  </form>
{% endblock %}

# notification_content.html

{% extends "base.html" %}

{% block page_body %}
  {% if priority_type.code=='high' %}
    <h1>Critical notification content</h1>
  {% elif priority_type.code=='medium' %}
    <h1>Important notification content</h1>
  {% else %}
    <h1>Notification content</h1>
  {% endif %}
  <ul>
    <li style="font-weight: bold;">Room number</li>
    <li>{{ room_number }}</li>
    <li style="font-weight: bold;">Guest name</li>
    <li>{{ guest_name }}</li>
    <li style="font-weight: bold;">Notification</li>
    <li>{{ notification_text }}</li>
  </ul>
{% endblock %}
```



## Dołączanie szablonów Jinja - include

### Notatka

- Include pozwala dołączyć w wybrane miejsce w szablonie innego pliku html (potencjalnie zawierającego inne kontrolki formatu Jinja)
- Include pozwala ustandaryzować elementy interfejsu aplikacji (np. dołączyć menu do każdej lub do wybranych stron). Dołączenie można wykonać nawet w stronie, z której dziedziczą kolejne strony
- Składnia wygląda następująco:

```
{% include "menu.html" %}
```

### Laboratorium

1. Klient aplikacji tworzonej w ramach poprzednich labów zażyczył sobie, aby na dole strony umieszczać pewne statyczne elementy (np. nazwę i adres hotelu). Normalnie tego rodzaju elementy można by włożyć do szablonu strony rodzicielskiej. Tym jednak razem klient chce, aby ów statyczny blok był łatwy w modyfikacji – najlepiej odłożony w osobnym pliku, dlatego skorzystaj z include!
2. Utwórz w templates plik footer.html np. o następującej zawartości:

```
<div style="text-align: center; border-style: dotted; font-weight: bold;">  
  Py-Hotel  
</div>
```

3. Dołącz w szablonie rodzicielskim na dole strony zawartość pliku footer.html

### Sprawdź się!

1. Kiedy zastosujesz instrukcje:
  - a. include
  - b. extends
  - c. import

## Propozycja rozwiązania

```
# file footer.html
<div style="text-align: center; border-style: dotted; font-weight: bold;">
    Py-Hotel
</div>

# file base.html

{% from "macros.html" import show_flash %}
<!DOCTYPE html>
<html>
    <head>
        <title>Hotel 101</title>
    </head>
    <body>
        {{ show_flash() }}

        {% block page_body %}
        {% endblock %}

        {% include 'footer.html' %}
    </body>
</html>
```

## Flask Bootstrap

### Notatka

- Bootstrap to darmowa biblioteka stylów CSS i skryptów JavaScript pozwalająca w łatwy sposób poprawić wygląd generowanej w aplikacji strony. Korzystanie z tego modułu jest absolutnie nieobowiązkowe, ale bootstrap jest znanym i uznanym sposobem na formatowanie stron
- Flask Bootstrap to moduł pozwalający (przynajmniej teoretycznie) korzystać z funkcjonalności Bootstrapa po stronie Flaska bez znajomości szczegółów implementacyjnych Bootstrapa
- Flask Bootstrap oddaje do dyspozycji programisty Flaska szablon wzorcowy z blokami, z których najpopularniejsze to:
  - Title – miejsce na umieszczenie tytułu
  - Navbar – miejsce na umieszczenie menu
  - Content – miejsce na umieszczenie zawartości generowanej strony
- Instalację modułu wykonujesz komendą

### Pip install flask-bootstrap

- Aby utworzyć instancję aplikacji świadomej nowego modułu należy skorzystać z instrukcji:

```
app = Flask(__name__)  
bootstrap = Bootstrap(app)
```

- Aby wypełniać bloki udostępnione przez Flask-Bootstrap szablony Jinja powinny dziedziczyć z szablonu dostarczonego przez Flask-Bootstrap. Od tej pory można wypełniać bloki definiowane przez Flask-Bootstrap umieszczając w nich kontrolki pochodzące z Bootstrap! (Przynajmniej tak powinno teoretycznie być)

```
{% extends "bootstrap/base.html" %}
```

### Laboratorium

1. Zróbmy coś, co dla odmiany... zadziała! Stwórz zupełnie nowy projekt. Stwórz wirtualne środowisko, zainstaluj Flaska oraz Flask-Bootstrap. Zmień środowisko na development i stwórz aplikację z route do strony głównej
2. W szablonie stowarzyszonym ze stroną główną odziedzicz wygląd z base.html z Flask-Bootstrap i w bloku content umieść formatowanie „badge” z adresu [Badges · Bootstrap v5.0 \(getbootstrap.com\)](https://getbootstrap.com/docs/5.0/components/badge/). Na wypadek zmian na docelowej stronie kod znajdziesz też w rozwiązaniach

Example heading New

Example heading New

Example heading New

Example heading New

Example heading New

- 3.

### Sprawdź się!

1. Jakim cudem mały framework Bootstrap może mieć szeroką funkcjonalność?
2. Jak instaluje się moduły Flask?
3. Co to jest Bootstrap, a co to jest Flask Bootstrap?

## Propozycja rozwiązania

# Przygotowanie środowiska:

```
Python -m venv venv
.\venv\scripts\activate.bat
Pip install flask
Pip install flask-bootstrap
SET FLASK_ENV=development
```

# Kod aplikacji

```
from flask import Flask, render_template
from flask_bootstrap import Bootstrap

app = Flask(__name__)
bootstrap = Bootstrap(app)

@app.route('/')
def index():
    return render_template('index.html')
```

# Kod szablonu index.html

```
{% extends "bootstrap/base.html" %}

{%block content %}
    <h1>Example heading <span class="badge bg-secondary">New</span></h1>
    <h2>Example heading <span class="badge bg-secondary">New</span></h2>
    <h3>Example heading <span class="badge bg-secondary">New</span></h3>
    <h4>Example heading <span class="badge bg-secondary">New</span></h4>
    <h5>Example heading <span class="badge bg-secondary">New</span></h5>
    <h6>Example heading <span class="badge bg-secondary">New</span></h6>
{% endblock %}
```

## Korzystanie z Bootstrap bez Flask Bootstrap

### Notatka

- Bootstrap składa się z:
  - Pliku CSS
  - Jednego lub dwóch plików skryptów JavaScript
- W celu wykorzystania Bootstrap w swoim projekcie można
  - Zbudować szablon rodzicielski importujący bootstrap i definiujący wybrane przez autora bloki
  - W pozostałych szablonach dziedziczyć z tego szablonu i definiować treść zdefiniowanych bloków

### Laboratorium

1. Zbuduj własny szablon base.html importujący elementy wymagane przez bootstrap
2. Zdefiniuj w nim bloki
  - a. Title – na tytuł strony
  - b. Content – na zawartość strony
3. W szablonie index.html
  - a. Odziedzicz ustawienia z szablonu base.html
  - b. Zmień tytuł strony
  - c. Dodaj do bloku content przycisk i okienko dialogowe zaprezentowane na stronie <https://getbootstrap.com/docs/5.0/components/modal/#static-backdrop> (jeśli docelowy adres nie jest już poprawny wykorzystaj dowolną inną kontrolkę bootstrap lub wykorzystaj kod z rozwiązania)
4. Jeśli korzystasz z aplikacji stworzonej w poprzednim lab: z pliku aplikacji usuń:
  - a. Importowanie modułu flask\_bootstrap
  - b. Instrukcję tworzącą instancję bootstrap

### Sprawdź się!

1. Czy korzystanie z modułów poszerzających możliwości Flaska jest jedynym sposobem na dodanie funkcjonalności do aplikacji?
2. Z czego mogą wynikać problemy z korzystania z dedykowanych modułów łączących Flask z innymi modułami Python?

## Propozycja rozwiązania

```
# File base.html

<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-BmbxuPwQa21c/FVZBCNJ7UAYjXM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlo1f1f1"
crossorigin="anonymous">
    <title>{% block title %} {% endblock %}</title>
  </head>
  <body>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/js/bootstrap.bundle.min.js" integrity="sha384-b5kHyXgcpbZJO/ty9U17kGkf1S0CWuKcCD3818YkeH8z8QjE0GmW1gYU5S9FonJ0"
crossorigin="anonymous"></script>
  </body>
</html>

# file index.html

{% extends "base.html" %}

{% block title %} Demo page for bootstrap in flask {% endblock %}

{%block content %}

<!-- Button trigger modal -->
<button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-
target="#staticBackdrop">
  Launch static backdrop modal
</button>

<!-- Modal -->
<div class="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-
keyboard="false" tabindex="-1" aria-labelledby="staticBackdropLabel" aria-
hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="staticBackdropLabel">Modal title</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
      </div>
      <div class="modal-body">
        .
        .
        .
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Understood</button>
      </div>
    </div>
  </div>
</div>

{% endblock %}
```

## Połączenie kodu z szablonami

### Notatka

- Flask daje różne możliwości wykorzystania metod programistycznych typowych dla Pythona oraz możliwości ustrukturyzowania szablonów odpowiadających za wygląd interfejsu
- W aplikacjach można korzystać ze wszystkich możliwości języka: ogólnie znane moduły jak i moduły tworzone specjalnie pod kątem aplikacji, klasy itp. to tylko kilka przykładów tego, co może być wykorzystane w aplikacji
- Szablony Jinja można połączyć ze sobą na dziesiątki sposobów. Zależnie od potrzeb (a niekiedy od historii aplikacji), poszczególne szablony mogą od siebie dziedziczyć (extends), być dołączane (include), mogą zawierać definicje funkcji (makra)

### Laboratorium

1. Zmień interfejs aplikacji pozwalającej na przyjmowanie zgłoszeń w hotelu korzystając z komponentów bootstrap:
  - a. Obmyśl sposób dziedziczenia i dołączania plików Jinja (np. rodzicielski base.html, w którym dołączane jest menu.html. Pozostałe strony dziedziczą z base.html)
  - b. Na stronie index umieść np. obiekt carousel z 3 obrazkami przedstawiającymi hotele
  - c. W menu utwórz pozycje Home (kierującą do funkcji index), Add notification (kierującą do funkcji notification), Notifications (historia – zrobimy później), About (kilka informacji o stronie, wykorzystaj np. accordion)

### Sprawdź się!

1. W jakiej sytuacji stosujemy w szablonach Jinja:
  - a. Notację {{ }}
  - b. A kiedy {% %}
  - c. Za co odpowiada instrukcja include
  - d. A za co extends?

### Propozycja rozwiązania

```
# menu.html
<ul class="nav nav-pills">
  <li class="nav-item">
    <a class="nav-link active" aria-current="page" href="{{ url_for('index') }}">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Notifications</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="{{ url_for('notification') }}">New notification</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="{{ url_for('about') }}" tabindex="-1">About this app</a>
  </li>
</ul>

# base.html
...
<title>{% block title %} {% endblock %}</title>
...
{% include 'menu.html' %}
```

```

    {% block content %} {% endblock %}
...
# index.html
{% extends "base.html" %}

{% block title %} Hotel notifications {% endblock %}

{%block content %}
<div id="carouselExampleIndicators" class="carousel slide" data-bs-ride="carousel">
  <div class="carousel-indicators">
    <button type="button" data-bs-target="#carouselExampleIndicators" data-bs-slide-to="0"
class="active" aria-current="true" aria-label="Slide 1"></button>
    <button type="button" data-bs-target="#carouselExampleIndicators" data-bs-slide-to="1" aria-
label="Slide 2"></button>
  </div>
  <div class="carousel-inner">
    <div class="carousel-item active">
      <div class="d-flex justify-content-center">
        
      </div>
    </div>
    <div class="carousel-item">
      <div class="d-flex justify-content-center">
        
      </div>
    </div>
  </div>
  <button class="carousel-control-prev" type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-bs-target="#carouselExampleIndicators"
data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
{% endblock %}

# app.py

from flask import Flask, url_for, request, redirect, render_template, flash
from datetime import datetime
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = '123GoniszTy!'

class PriorityType:
    ...

class NotificationPriorities:
    ...

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

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/')
def index():
    return render_template('index.html')

```



## Stosowanie formatów Bootstrap

### Notatka

- Pomysłem Bootstrapa na ułożenie elementów na stronie jest tworzenie grid-a
  - W kontenerze umieszcza się wiersze,
  - A każdy wiersz dzieli się na 12 kolumn
  - 12 domyślnych kolumn można sklejać, np. po 2, po 3, ... dzięki czemu można dość dowolnie rozmieszczać elementy na stronie
- Bootstrap pozwala też na umieszczanie elementów innych typów, np. obrazów
- Specjalna sekcja dokumentacji bootstrap prezentuje sposoby budowania formularzy. Znajdują się tam przykłady typowych pól formularzy i np. przycisków
- Do wyświetlania alertów można posłużyć się kontrolkami z grupy Alerts

### Laboratorium

1. Zmień formatowanie formularza do zgłoszeń hotelowych w pliku notification.html (zastosuj wybrane przez siebie formatowanie)
2. Przyjmując wartości z formularza wyświetl w nagłówku automatycznie generowaną informację o priorytecie zdarzenia (użyj tej samej logiki, co w poprzednich zadaniach), a poniżej umieść numer pokoju, nazwisko gościa i tekst zgłoszenia korzystając z tych samych pól, co na oryginalnym formularzu z dodanym do każdego z nich atrybutem readonly. Wyświetlając wartość skorzystaj z atrybutu value dla pól input.

### Sprawdź się!

1. Strony opisywane przez bootstrap dzielą wiersze na kolumny czy kolumny na wiersze?
2. W jakiej sekcji dokumentacji poszukasz informacji o polu formularza typu select?

### Propozycja rozwiązania

```
# form.html - only the modified part
<form id="notification_form" method="POST" action="{{ url_for('notification') }}">
  <div class="container px-4">
    <div class="row mb-3">
      <div class="col-12"><h2>Notify shift manager:</h2></div>
    </div>
    <div class="row mb-3">
      <div class="col-3"><label for="room_number" class="form-label">Room
number</label></div>
      <div class="col-6"><input type="text" name="room_number" id="room_number"
class="form-control"></div>
      <div class="col-3"></div>
    </div>
    <div class="row mb-3">
      <div class="col-3"><label for="guest_name" class="form-label">Guest
name</label></div>
      <div class="col-6"><input type="text" name="guest_name" id="guest_name" class="form-
control"></div>
      <div class="col-3"></div>
    </div>
    <div class="row mb-3">
      <div class="col-3"><label for="notification_text" class="form-
label">Notification</label></div>
      <div class="col-6"><textarea rows="4" cols="50" name="notification_text" class="form-
control"></textarea></div>
      <div class="col-3"></div>
    </div>
  </div>
</form>
```

```

    </div>
    <div class="row mb-3">
      <div class="col-3"><label for="priority" class="form-label">Priority</label></div>
      <div class="col-6">
        <select name="priority" id="priority" class="form-select">
          {% for priority in list_of_priorities %}
            <option value="{{ priority.code }}" {{ 'selected' if
priority.selected}}>{{ priority.description }}</option>
          {% endfor %}
        </select>
      </div>
    </div>
    <div class="col-3"></div>
  </div>
  <div class="row mb-3">
    <div class="col-3"></div>
    <div class="col-6"><input type="submit" value="Send" class="btn btn-primary"></div>
    <div class="col-3"></div>
  </div>
</div>
</form>

# notification_confirmation.html

<div class="container px-4">
  <div class="row mb-3">
    <div class="col-12">
      <h2>
        {% if priority_type.code=='high' %}
          Critical notification content
        {% elif priority_type.code=='medium' %}
          Important notification content
        {% else %}
          Notification content
        {% endif %}
      </h2>
    </div>
  </div>
  <div class="row mb-3">
    <div class="col-3"><label for="room_number" class="form-label">Room
number</label></div>
    <div class="col-6"><input type="text" name="room_number" id="room_number"
class="form-control" value="{{ room_number }}" readonly></div>
    <div class="col-3"></div>
  </div>
  <div class="row mb-3">
    <div class="col-3"><label for="guest_name" class="form-label">Guest
name</label></div>
    <div class="col-6"><input type="text" name="guest_name" id="guest_name" class="form-
control" value="{{ guest_name }}" readonly></div>
    <div class="col-3"></div>
  </div>
  <div class="row mb-3">
    <div class="col-3"><label for="notification_text" class="form-
label">Notification</label></div>
    <div class="col-6">
      <textarea rows="4" cols="50" name="notification_text" class="form-control">
        {{ notification_text }}
      </textarea>
    </div>
    <div class="col-3"></div>
  </div>
</div>

```

## Projekt 02 – Formularz z pomysłami wycieczek

Jeśli masz ochotę na chwilę się zatrzymać, osobiście się przekonać, że jesteś w stanie samodzielnie stworzyć coś, co działa, jeśli nie boisz się zmierzyć z kilkoma wyzwaniami, to zapraszam do samodzielnego stworzenia mini stronki do zbierania pomysłów wycieczek organizowanych przez biuro turystyczne.

Byłeś/-aś kiedyś na zorganizowanej wycieczce? Nie powinno być zaskoczeniem, że takie wycieczki ktoś planuje, a ten plan zaczyna się od pomysłu. Im lepszy pomysł, tym później lepiej sprzedająca się wycieczka. Szef firmy wymyślił, że trzeba dać szansę pracownikom na zgłaszanie pomysłów wycieczek. Pewnie 80% to i tak nie będą oryginalne propozycje, ale może wśród pozostałych znajdą się jakieś perełki? Twoim zadaniem jest napisanie aplikacji, która:

- Pozwoli wypełnić formularz z nazwą wycieczki, adresem email pomysłodawcy, krótkim opisem wycieczki i uwaga – coś czego nie robiliśmy – polem radio box – pozwalającym na zaznaczenie czy pracownik ma już kalkulację dla tej wycieczki, czy nie i polem checkbox określającym, czy pracownik chce dalej prowadzić prace nad wycieczką czy nie. Formularz mógłby wyglądać następująco:

The screenshot shows a web application interface for adding a new trip idea. At the top, there is a navigation bar with 'TIA', 'Home', and 'Add new trip'. The main heading is 'Add new trip idea!'. Below this, there are several input fields and checkboxes:

- Trip name:** A text input field containing 'Taste of Sicily'.
- Your email:** A text input field containing 'Roberto.Morandi@tripandtrip.com'.
- Short description:** A text area containing 'A week long trip allowing to visit the most popular places in Sicily with special attention to local kitchen and specialities'.
- Completeness:** Two radio buttons. The first is selected and labeled 'Yes - the idea is complete including price proposal'. The second is labeled 'No - this is just pure idea'.
- May we contact you for details?:** Two checkboxes. The first is selected and labeled 'Yes, I agree'.
- Send proposal:** A green button.

- Wprowadzone dane powinny być zapisane w pliku trips.txt w katalogu static. Do zapisu można wykorzystać format CSV, a sam odczyt i zapis wykonać przy pomocy następujących funkcji:

```
def read_csv_data():  
    full_file_path = os.path.join(app.static_folder, 'trips.txt')  
    fieldnames = ['trip_name', 'email', 'description', 'completeness', 'contact_ok']  
    entries = []
```

```

with open(full_file_path, mode='r', encoding="utf-8") as f:
    csv_reader = csv.DictReader(f, fieldnames=fieldnames)
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            line_count += 1
        else:
            entries.append(row)
            line_count += 1
    return entries

def append_csv_data(data):
    full_file_path = os.path.join(app.static_folder, 'trips.txt')
    fieldnames = ['trip_name', 'email', 'description', 'completeness', 'contact_ok']

    if not os.path.exists(full_file_path):
        with open(full_file_path, 'w+', newline='', encoding="utf-8") as f:
            writer = csv.DictWriter(f, fieldnames=fieldnames)
            writer.writeheader()

    with open(full_file_path, 'a', newline='', encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writerow(data)

```

- Na stronie domowej zbuduj prosty formularz pozwalający wybrać wycieczkę „po nazwie”.

The screenshot shows a web application with a dark header containing the text 'TIA Home Add new trip'. Below the header is a yellow notification bar with the text 'Trip idea has been saved!' and a close button (X). The main content area has the heading 'Check existing ideas of trips:' followed by the text 'Please select a name:'. Below this is a label 'Trip name' and a dropdown menu with the selected value 'Animals of north Europe'. At the bottom of the form is a blue 'Submit' button.

- Po wyborze powinien zostać wyświetlony bliźniaczy formularz do tego, w którym dane były wprowadzone. Pola wyświetlane w tym formularzu powinny być nieaktywne:

TIA Home Add new trip

## Details for a trip

Trip name

Sport in Croatia

Your email

adam@gmail.com

Short description

A trip to Croatia with a trainer (and a pilot in one). The participants have some organized sport activities daily in the morning. Later they can spent the time as they want.

Completeness

☒ Yes - the idea is complete including price proposal  
☐ No - this is just pure idea

May we contact you for details?

☒ Yes, I agree

- Staraj się wykorzystać poznane do tej pory metody, np.:
  - Szablony Jinja, dziedziczenie, dołączanie, tworzenie bloków
  - Kontrolki bootstrap: menu, podział wiersza na 12 kolumn, kontrolki do formularza, atrybuty kontrolek, domyślne wartości wbudowywane w szablon przez `{{}}` lub `{%}`
  - Obsługa metody GET i POST w jednej funkcji,
  - Funkcje `redirect`, `render_template`, `url_for`
  - Komunikaty flash
- Jeśli zechcesz, dodaj możliwość usunięcia wpisu lub jego modyfikacji.
- Postaraj się zadbać o podstawową kontrolę formularza. Wydaje się, że minimalna informacja z formularza, to tytuł wycieczki. Jeśli więc ktoś próbuje przestać pomysł bez tytułu, to należy wrócić na stronę formularza i wyświetlić odpowiedni komunikat. Dobrze by też było, żeby już wcześniej wprowadzone inne dane nie zniknęły wtedy z formularza.

### Propozycja rozwiązania (fragmenty)

Przyjmowanie parametrów formularza:

```
if request.method == 'GET':
    return render_template('new_trip.html',
        trip_name=trip_name, email=email, description=description, completeness=completeness, contact_ok=contact_ok)
else:
    trip_name = '' if 'trip_name' not in request.form else request.form['trip_name']
    email = '' if 'email' not in request.form else request.form['email']
    description = '' if 'description' not in request.form else request.form['description']
    completeness = False if 'completeness' not in request.form else request.form['completeness']
    contact_ok = False if 'contact_ok' not in request.form else True
```

Kod formularza przyjmującego pomysły:

```
<form id="trip_form" method="POST" action="{{ url_for('new_trip') }}">
<div class="container">
  <div class="row mb-3">
    <div class="col-1">
    </div>
    <div class="col-10">
      <div class="form-group row mb-3">
        <label for="trip_name" class="col-sm-4 col-form-label">Trip name</label>
        <div class="col-sm-8">
          <input type="text" class="form-control" id="trip_name" name="trip_name" value="{{trip_name}}">
        </div>
      </div>
      <div class="form-group row mb-3">
        <label for="email" class="col-sm-4 col-form-label">Your email</label>
        <div class="col-sm-8">
          <input type="email" class="form-control" id="email" name="email" value="{{email}}">
        </div>
      </div>
      <div class="form-group row mb-3">
        <label for="description" class="col-sm-4 col-form-label">Short description</label>
        <div class="col-sm-8">
          <textarea rows="4" cols="50" class="form-control" id="description" name="description">{{description}}</textarea><br>
        </div>
      </div>
      <fieldset class="form-group mb-3">
        <div class="row">
          <legend class="col-form-label col-sm-4 pt-0">Completeness</legend>
          <div class="col-sm-8">
            <div class="form-check">
              <input type="radio" name="completeness" id="completeness_1" value="yes" {{'checked' if completeness else ''}}>
              <label class="form-check-label" for="completeness_1">
                Yes - the idea is complete including price proposal
              </label>
            </div>
            <div class="form-check">
              <input type="radio" name="completeness" id="completeness_2" value="no" {{'checked' if not completeness}}>
              <label class="form-check-label" for="completeness_2">
                No - this is just pure idea
              </label>
            </div>
          </div>
        </div>
      </fieldset>
      <div class="form-group row mb-3">
        <div class="col-sm-4">May we contact you for details?</div>
        <div class="col-sm-8">
          <div class="form-check">
            <input type="checkbox" id="contact_ok" name="contact_ok" value="yes" {{'checked' if contact_ok}}>
            <label class="form-check-label" for="contact_ok">
              Yes, I agree
            </label>
          </div>
        </div>
      </div>
      <div class="form-group row mb-3">
        <div class="col-sm-12">
          <button type="submit" class="btn btn-success btn-block">Send proposal</button>
        </div>
      </div>
    </div>
  </div>
</div>
</form>
```

## Instalacja SQLite 3 i podstawy pracy z danymi

### Notatka

- Baza danych SQLite jest prosta do zainstalowania i wykorzystania we Flasku
- Prawdziwe aplikacje produkcyjne prawdopodobnie będą korzystały z dedykowanych silników bazy danych
- Instalacja zależy od systemu operacyjnego. O ile użytkownicy Linuxa są przyzwyczajeni do modyfikowania ustawień systemowych po instalacji pakietu, o tyle użytkownicy Windows mogą być zaskoczeni tym, że trzeba samodzielnie zmodyfikować zmienną PATH wskazując na katalog z binariami SQLite 3
- Aby utworzyć nową bazę danych i zacząć z nią pracować wystarczy wykonać polecenie:

```
sqlite3 mydb.db
```

- Aby utworzyć tabelę można posłużyć się poleceniem:

```
create table furniture(id integer primary key autoincrement, name varchar(50), cost int, mod_date date default(date()));
```

- Aby wstawić do takiej tabeli rekord posługujemy się poleceniem insert. Jeśli kolumny tabeli mają wartości domyślne, to można ich w tym poleceniu nie definiować:

```
insert into furniture(name, cost) values('table', 700);
```

- Aby wyświetlić wszystkie rekordy z tabeli uruchom

```
select * from furnitures
```

- Aby wyjść z SQLite posłuż się poleceniem .quit

### Laboratorium

1. Pobierz i zainstaluj na swoim komputerze SQLite3
2. Utwórz bazę danych notifications.db, a w niej tabelę notifications z polami:
  - a. id – klucz podstawowy, autonumerowanie
  - b. room\_number – napis, maksymalnie 10 znaków
  - c. guest\_name – napis, maksymalnie 30 znaków
  - d. notification – napis bez określonej długości – to specjalny typ text
  - e. priority – napis, maksymalnie 20 znaków
3. Wstaw do tej tabeli ręcznie jeden rekord.
4. Wśród poleceń dostępnych w SQLite3 poszukaj komendy pozwalającej wyświetlenie nagłówka przed wynikami zwracanymi przez polecenie select. Włącz tą opcję.
5. Wyświetl wszystkie rekordy z tabeli notifications
6. Wyjdź z SQLite3

### Sprawdź się!

1. Czy SQLite3 jest wbudowanym elementem Flask?
2. Czy tworząc aplikacje Flaskowe warto opanować jeszcze jakąś bazę danych?
3. Jakie polecenie służy do wstawiania, a jakie do wyświetlania danych?

### Propozycja rozwiązania

```
sqlite3 notification.db
```

```
sqlite> create table notifications(id integer primary key autoincrement, room_number  
varchar(10), guest_name varchar(30), notification text, priority varchar(20));
```

```
sqlite> insert into notifications(room_number, guest_name, notification, priority)  
values('10A', 'Frederico Romane', 'The window cannot be open', 'MEDIUM');
```

```
sqlite>.help
```

```
sqlite> .header on
```

```
sqlite> select * from notifications;  
id|room_number|guest_name|notification|priority  
1|10A|Frederico Romane|The window cannot be open|MEDIUM
```

```
.quit
```



## Zapis danych z rekordu do bazy danych

### Notatka

- Do korzystania z sqlite3 z poziomu Flaska potrzebny jest moduł sqlite3
- Dane globalne na skalę pojedynczego żądania (request) można przechowywać w zmiennej g, którą należy zaimportować z Flaska
- Dobrą praktyką jest unikanie częstego otwierania połączeń do bazy danych, dlatego warto przy żądaniu połączenie otworzyć tylko raz i przechowywać je w g
- Pochodzące z <https://flask.palletsprojects.com/en/1.1.x/patterns/sqlite3/?highlight=sqlite3> przydatne funkcje pozwalające na otwieranie i zamykanie połączenia to:

```
def get_db():
    if not hasattr(g, 'sqlite_db'):
        conn = sqlite3.connect(app_info['db_file'])
        conn.row_factory = sqlite3.Row
        g.sqlite_db = conn
    return g.sqlite_db

@app.teardown_appcontext
def close_db(error):
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()
```

- Zapis rekordu do tabeli wykonuje się w następujący sposób:

```
db = get_db()
sql_command = 'insert into transactions(currency, amount, user) values(?, ?, ?)'
db.execute(sql_command, [currency, amount, 'admin'])
db.commit()
```

### Laboratorium

1. Do aplikacji dotyczącej zgłoszeń hotelowych dodaj funkcje pozwalające na pracę z bazą danych. Pamiętaj również o dodaniu słownika opisującego konfigurację aplikacji
2. Dodaj kod pozwalający zapisać nowe zgłoszenie po jego przyjęciu. Przetestuj aplikację sprawdzając czy w bazie danych pojawia się nowy rekord zgodny z wprowadzonymi na stronie danymi.

### Sprawdź się!

1. Do czego służy g ?
2. Jakie polecenie zatwierdza w bazie danych wykonane wcześniej zmiany?
3. Dlaczego przejmujemy się tym, aby zbyt często nie nawiązywać połączenia z bazą danych?

## Propozycja rozwiązania

```
from flask import Flask, url_for, request, redirect, render_template, flash, g
from datetime import datetime
import sqlite3
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = '123GoniszTy!'

app_info = { 'db_file' : r"D:\FLASK\hotel\data\notification.db" }

def get_db():
    if not hasattr(g, 'sqlite_db'):
        conn = sqlite3.connect(app_info['db_file'])
        conn.row_factory = sqlite3.Row
        g.sqlite_db = conn
    return g.sqlite_db

@app.teardown_appcontext
def close_db(error):
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()

@app.route('/notification', methods=['GET', 'POST'])
def notification():
    notification_priorities = NotificationPriorities()
    notification_priorities.load_priorities()

    if request.method == 'GET':
        return render_template('notification.html',
                               list_of_priorities=notification_priorities.list_of_priorities)
    else:
        room_number = request.form['room_number'] if 'room_number' in request.form else ''
        guest_name = request.form['guest_name'] if 'guest_name' in request.form else ''
        notification_text = request.form['notification_text'] if 'notification_text' in request.form else ''
        priority = request.form['priority'] if 'priority' in request.form else 'normal'

        priority_type = notification_priorities.get_priority_by_code(priority)
        print('found', priority_type.code)

        flash('Notification has been sent')

        the_hour = datetime.now().hour
        raise_priority = (the_hour >= 20 or the_hour < 10) and priority == 'medium'

        if raise_priority:
            priority = 'high'
            flash('Rising priority from medium to high')

        db = get_db()
        sql_command = 'insert into notifications(room_number, guest_name, notification, priority)'
        values(?, ?, ?, ?)
        db.execute(sql_command, [room_number, guest_name, notification_text, priority])
        db.commit()

        return render_template('notification_content.html',
                               room_number=room_number, guest_name=guest_name,
                               notification_text=notification_text, priority_type=priority_type)
```

## Pobieranie rekordów z bazy danych

### Notatka

- Do odczytywania rekordów z bazy danych należy zbudować kursor:

```
db = get_db()
sql_command = 'select id, currency, amount, trans_date from transactions;'
cur = db.execute(sql_command)
transactions = cur.fetchall()
```

- Podczas renderowania strony prezentującej wyniki, wystarczy przekazać pobrane z bazy danych rekordy jako parametr

```
return render_template('hist.html', active_menu='history', transactions=transactions)
```

- Podczas renderowania zawartości strony wystarczy przechodzić przez wszystkie elementy kolekcji transactions i korzystać z wartości rekordów wpisując ich nazwy po kropce:

```
{% for transaction in transactions %}
<tr>
  <th scope="row">{{ transaction.id }}</th>
  <td>{{ transaction.currency }}</td>
  <td>{{ transaction.amount }}</td>
  <td>{{ transaction.trans_date }}</td>
  <td>
    <a href="#" class="btn btn-primary btn-sm" role="button">Actions...</a>
    <a href="#" class="btn btn-success btn-sm" role="button">Edit...</a>
    <a href="#" class="btn btn-danger btn-sm" role="button">Delete...</a>
  </td>
</tr>
{% endfor %}
```

### Laboratorium

1. W hotelowej aplikacji do zgłoszeń, oprogramuj stronę prezentującą listę wszystkich zgłoszeń:
  - a. Dodaj route i funkcję z nią związaną
  - b. Pobierz dane z bazy danych
  - c. Zaprezentuj je w postaci tabeli (możesz wybrać styl z galerii Bootstrapa)
2. Jeśli trzeba, oprogramuj menu, tak aby było widać jaka pozycja jest w danej chwili aktywna

### Sprawdź się!

1. Do czego służy kursor?
2. Jak jest generowana strona z wierszami z tabeli?
3. Czy prezentowanie w interfejsie użytkownika identyfikatora rekordu generowanego w bazie danych to dobra, czy zła praktyka?

## Propozycja rozwiązania

```
# app.py - only the new notification function (add active_menu param to each render template):
@app.route('/notifications')
def notifications():
    db = get_db()
    sql_command = 'select id, room_number, guest_name, notification, priority from
notifications;'
    cur = db.execute(sql_command)
    notifications = cur.fetchall()

    return render_template('notifications.html', active_menu='notifications',
notifications=notifications)

# notifications.html
{% extends "base.html" %}

{% block title %}
    Current notifications
{% endblock %}

{% block content %}
<div class="container">
    <table class="table">
        <thead>
            <tr>
                <th scope="col">#</th>
                <th scope="col">Room Number</th>
                <th scope="col">Guest Name</th>
                <th scope="col">Notification</th>
                <th scope="col">Priority</th>
                <th scope="col">Operations</th>
            </tr>
        </thead>
        <tbody>
            {% for n in notifications %}
            <tr>
                <th scope="row">{{ n.id }}</th>
                <td>{{ n.room_number }}</td>
                <td>{{ n.guest_name }}</td>
                <td>{{ n.notification }}</td>
                <td>{{ n.priority }}</td>
                <td>
                    <a href="#" class="btn btn-primary btn-sm" role="button">Actions...</a>
                    <a href="#" class="btn btn-success btn-sm" role="button">Edit...</a>
                    <a href="#" class="btn btn-danger btn-sm" role="button">Delete...</a>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock %}

# menu.html
<ul class="nav nav-pills">
    <li class="nav-item">
        <a class="nav-link {{ 'active' if active_menu=='index' }}" aria-current="page" href="{{
url_for('index') }}">Home</a>
    </li>
    <li class="nav-item">
        <a class="nav-link {{ 'active' if active_menu=='notifications' }}" href="{{
url_for('notifications') }}">Notifications</a>
    </li>
    <li class="nav-item">
        <a class="nav-link {{ 'active' if active_menu=='notification' }}" href="{{
url_for('notification') }}">New notification</a>
    </li>
    <li class="nav-item">
        <a class="nav-link {{ 'active' if active_menu=='about' }}" href="{{ url_for('about') }}"
tabindex="-1">About this app</a>
    </li>
</ul>
```

## Kasowanie rekordu z pytaniem o potwierdzenie

### Notatka

- Do usunięcia rekordu można się posłużyć poleceniem:

```
db = get_db()
sql_statement = 'delete from transactions where id = ?;'
db.execute(sql_statement, [transaction_id])
db.commit()
```

- Przed usunięciem rekordu warto zapytać użytkownika o potwierdzenie
  - Odpowiedni formularz może być umieszczony wielokrotnie (osobno dla każdego wiersza, co jednak znacznie powiększa plik html)
  - Formularz może być na stronie umieszczony jeden raz i być dynamicznie łączony z przyciskami generowanymi dla każdego wiersza na stronie (patrz kod w LAB)

### Laboratorium

1. Krótko – oprogramuj usuwanie notyfikacji w systemie hotelowym. W wersji prostszej zrób to bez dodatkowego pytania o potwierdzenie. Kod JQuery do potwierdzania usuwania znajdziesz tu:

```
<!-- Modal -->
<div class="modal fade" id="confirmDeleteModal" tabindex="-1" aria-labelledby="exampleModalLabel"
aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">This entry will be deleted:</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="close"></button>
      </div>
      <div class="modal-body" id="idDeleteModalBody">
        <div>
        </div>
      </div>
      <div class="modal-footer">
        <form action="" method="GET" id="confirmDeleteForm">
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
          <button type="submit" class="btn btn-danger">Delete</button>
        </form>
      </div>
    </div>
  </div>
</div>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function () {
  // For A Delete Record Popup
  // This function is applied to all elements with class ending with ".delete-confirm"
  $(''.delete-confirm').click(function () {
    // get attributes of the found element
    var desc = $(this).attr('data-desc');
    var url = $(this).attr('data-url');
    // the #... designates id of an element - change the text in the modal window
    $('#idDeleteModalBody').text(desc);
    $('#confirmDeleteForm').attr("action", url);
  });
});
</script>
```

### Sprawdź się!

1. Jakie polecenie SQL jest wykorzystywane do usuwania rekordów?
2. Jak nazywa się grupa kontrolek bootstrap wyświetlających okna dialogowe?

## Propozycja rozwiązania

```
# file notifications.html - only fragment with buttons:
<a href="#" class="btn btn-primary btn-sm" role="button">Actions...</a>
<a href="#" class="btn btn-success btn-sm" role="button">Edit...</a>

<a type="button" class="btn btn-danger btn-sm delete-confirm"
  data-bs-toggle="modal" data-bs-target="#confirmDeleteModal"
  data-desc="{{ 'Delete notification for {} {}'.format(n.room_number, n.guest_name)
  }}"
  data-url="{{ url_for('delete_notification', notification_id=n.id) }}">
  Delete
</a>

# file app.py - only method deleting a row
@app.route('/delete_notification/<int:notification_id>')
def delete_notification(notification_id):
    db = get_db()
    sql_statement = 'delete from notifications where id = ?;'
    db.execute(sql_statement, [notification_id])
    db.commit()

    return redirect(url_for('notifications'))
```

## Modyfikacja danych

### Notatka

- Formularz do edycji danych często jest podobny do formularza do wprowadzania danych, z tą różnicą, że już na początku prezentuje dane oryginalnego rekordu
- Do pobrania aktualnej wartości z bazy danych należy skorzystać z metody fetchone() zamiast fetchall(). Dzięki temu zostanie zwrócony tylko jeden rekord – ten, który jest zmieniany:

```
sql_statement = 'select id, currency, amount from transactions where id=?;'
cur = db.execute(sql_statement, [transaction_id])
transaction = cur.fetchone()
```

- Do aktualizacji rekordu należy skorzystać z polecenia update np.:

```
sql_command = '''update transactions set currency=?, amount=?, user=?, trans_date=?
                  where id=?'''
db.execute(sql_command, [currency, amount, 'admin', date.today(), transaction_id])
db.commit()
```

### Laboratorium

1. Dodaj do aplikacji hotelowej
  - a. Route i funkcję pozwalającą na edytowanie rekordu
  - b. Dodaj szablon notification\_edit, pozwalający na edytowanie rekordu. Zadbaj o wypełnienie pól formularza po jego wyświetleniu. Pamiętaj, że podczas porównywania napisów wielkie i małe litery są rozróżniane
  - c. Zmień definicję linków w przyciskach w pliku notifications.html tak, aby można było uruchomić stworzoną funkcję

### Sprawdź się!

1. Jakie polecenie pozwala na wysłanie zmian do bazy danych
2. Jaka funkcja po stronie modułu sqlite3 odpowiada za pobranie jednego rekordu
3. Jakie są kolejne kroki wykonywane przez aplikację w przypadku typowej modyfikacji rekordu?

### Propozycja rozwiązania

```
# app.py - only the edit_notification function:
@app.route('/edit_notification/<int:notification_id>', methods=['GET', 'POST'])
def edit_notification(notification_id):
    db = get_db()
    notification_priorities = NotificationPriorities()
    notification_priorities.load_priorities()
    if request.method == 'GET':
        sql_statement = 'select id, room_number, guest_name, notification, priority from
notifications where id = ?'
        cur = db.execute(sql_statement, [notification_id])
        notif_obj = cur.fetchone()
        if notif_obj == None:
            flash('No such notification...')
            return redirect('notifications')
        else:
            return render_template('edit_notification.html', active_menu='notifications',
notif_obj=notif_obj,
                                list_of_priorities=notification_priorities.list_of_priorities)
    else:
```

```

        room_number = request.form['room_number'] if 'room_number' in request.form else ''
        guest_name = request.form['guest_name'] if 'guest_name' in request.form else ''
        notification_text = request.form['notification_text'] if 'notification_text' in
request.form else ''
        priority = request.form['priority'] if 'priority' in request.form else 'normal'
        priority_type = notification_priorities.get_priority_by_code(priority)

        sql_command = '''update notifications set room_number = ?, guest_name = ?,
                        notification = ?, priority = ? where id = ?'''
        db.execute(sql_command, [room_number, guest_name, notification_text, priority,
notification_id])
        db.commit()
        flash('Notification has been updated')
        return redirect(url_for('notifications'))

# notifications.html - how to start edit action:

<a href="{{ url_for('edit_notification', notification_id=n.id) }}" class="btn btn-success btn-sm"
role="button">Edit...</a>

# edit_notification.html

{% extends "base.html" %}
{% block title %} Hotel notifications {% endblock %}
{% block content %}
<form id="notification_form" method="POST" action="{{ url_for('edit_notification',
notification_id=notif_obj.id) }}">
    <div class="container px-4">
        <div class="row mb-3">
            <div class="col-12"><h2>Edit notification details:</h2></div>
        </div>
        <div class="row mb-3">
            <div class="col-3"><label for="room_number" class="form-label">Room
number</label></div>
            <div class="col-6"><input type="text" name="room_number" id="room_number"
class="form-control"
                value="{{ notif_obj.room_number }}"></div>
        </div>
        <div class="row mb-3">
            <div class="col-3"><label for="guest_name" class="form-label">Guest
name</label></div>
            <div class="col-6"><input type="text" name="guest_name" id="guest_name" class="form-
control"
                value="{{ notif_obj.guest_name }}"></div>
        </div>
        <div class="row mb-3">
            <div class="col-3"><label for="notification_text" class="form-
label">Notification</label></div>
            <div class="col-6">
                <textarea rows="4" cols="50" name="notification_text" class="form-control">{{
notif_obj.notification }}</textarea>
            </div>
        </div>
        <div class="row mb-3">
            <div class="col-3"><label for="priority" class="form-label">Priority</label></div>
            <div class="col-6">
                <select name="priority" id="priority" class="form-select">
                    {% for priority in list_of_priorities %}
                    <option value="{{ priority.code }}" {{ 'selected' if
priority.code==notif_obj.priority }}>{{ priority.description }}</option>
                    {% endfor %}
                </select>
            </div>
        </div>
        <div class="col-3"></div>
    </div>
    <div class="row mb-3">
        <div class="col-3"></div>
        <div class="col-6"><input type="submit" value="Send" class="btn btn-primary"></div>
    </div>
</div>
</form>
{% endblock %}

```



## Użytkownicy aplikacji - przygotowanie

### Notatka

- Logowanie użytkownika i system uprawnień to wspólny element aplikacji webowych. Można podjąć się samodzielnego oprogramowania tej funkcjonalności albo korzystać z gotowych modułów stworzonych w tym celu
- Hasła zazwyczaj przechowuje się w tabeli bazy danych w postaci „zahaszowanej” i „posolonej”

### Laboratorium

1. Do hotelowej aplikacji dodaj funkcje opisane w lekcji. Skorzystamy z nich w kolejnych lekcjach. Odpowiedni kod znajdziesz poniżej,

### Sprawdź się!

1. Jakie ryzyka niesłoby zapisywanie w bazie danych hasła w oryginalnej postaci?
2. Na czym polega haszowanie hasła? Czy haszowanie jest czynnością odwracalną?
3. Co to jest i po co stosujemy solenie haszowanych wartości?

### Propozycja rozwiązania

```
# SQL to create a table
CREATE TABLE users(
    id integer primary key autoincrement,
    name varchar(100) not null unique,
    email varchar(100) not null unique,
    password text,
    is_active boolean not null default 0,
    is_admin boolean not null default 0
);

# app.py - import statements

import random
import string
import hashlib
import binascii

# app.py - helper UserPass class

class UserPass:

    def __init__(self, user='', password=''):
        self.user = user
        self.password = password

    def hash_password(self):
        """Hash a password for storing."""
        # the value generated using os.urandom(60)
        os_urandom_static =
b"ID_\x12p:\x8d\xe7&\xcb\xf0=H1\xc1\x16\xac\xe5BX\xd7\xd6j\xe3i\x11\xbe\xaa\x05\xccc\xc2\xe8K\xcf
\xfl\xac\x9bFy(\xfbn.`\xe9\xcd\xdd'\xdf ~vm\xae\xf2\x93wD\x04"
        salt = hashlib.sha256(os_urandom_static).hexdigest().encode('ascii')
        pwdhash = hashlib.pbkdf2_hmac('sha512', self.password.encode('utf-8'), salt, 100000)
        pwdhash = binascii.hexlify(pwdhash)
        return (salt + pwdhash).decode('ascii')
```

```

def verify_password(self, stored_password, provided_password):
    """Verify a stored password against one provided by user"""
    salt = stored_password[:64]
    stored_password = stored_password[64:]
    pwdhash = hashlib.pbkdf2_hmac('sha512', provided_password.encode('utf-8'),
    salt.encode('ascii'), 100000)
    pwdhash = binascii.hexlify(pwdhash).decode('ascii')
    return pwdhash == stored_password

def get_random_user_password(self):
    random_user = ''.join(random.choice(string.ascii_lowercase) for i in range(3))
    self.user = random_user

    password_characters = string.ascii_letters + string.digits + string.punctuation
    random_password = ''.join(random.choice(password_characters) for i in range(3))
    self.password = random_password

# app.py - init route and function

@app.route('/init_app')
def init_app():

    # check if there are users defined (at least one active admin required)
    db = get_db()
    sql_statement = 'select count(*) as cnt from users where is_active and is_admin;'
    cur = db.execute(sql_statement)
    active_admins = cur.fetchone()

    if active_admins!=None and active_admins['cnt']>0:
        flash('Application is already set-up. Nothing to do')
        return redirect(url_for('index'))

    # if not - create/update admin account with a new password and admin privileges, display
    random_username
    user_pass = UserPass()
    user_pass.get_random_user_password()
    sql_statement = '''insert into users(name, email, password, is_active, is_admin)
    values(?,?,?,True, True);'''
    db.execute(sql_statement, [user_pass.user, 'noone@nowhere.no', user_pass.hash_password()])
    db.commit()
    flash('User {} with password {} has been created'.format(user_pass.user, user_pass.password))
    return redirect(url_for('index'))

```

## Sesja w akcji – logowanie i wylogowanie użytkownika

### Notatka

- Nie należy przesadzać z ilością danych przechowywanych w sesji. Na pewno nie należy przechowywać w niej właściwych danych aplikacji
- Do pracy z sesją zaimportuj session
- Jeśli aplikacja wymaga modyfikacji czasu trwania sesji, to należy zmodyfikować wartości parametrów permanent i permanent\_session\_lifetime
- Aby w sesji zapisać jakąś wartość posłuż się instrukcją podobną do poniższej:

```
session['user'] = request.form['user']
```

- Aby sprawdzić, czy w sesji znajduje się określona kluczem wartość skorzystaj z:

```
if 'user' in session:
```

- Aby pobrać wartość sesji użyj:
- Session['user']
- Aby usunąć wartość z sesji, usuń ją z obiektu session:

```
session.pop('user', None)
```

### Laboratorium

1. Korzystając z umieszczonego w rozwiązaniu kodu, dodaj do swojej aplikacji możliwość zalogowania się i wylogowania się. Jeżeli masz inne pomysły na formularz, albo funkcje sprawdzające użytkownika i jego hasło, to skorzystaj z nich!

### Sprawdź się!

1. Dlaczego w sesji nie należy przechowywać poufnych danych?
2. Jak sprawdzić, czy w sesji jest zapisana wartość dla danego klucza?
3. Jak zapisywać i jak odczytywać dane z sesji?

### Propozycja rozwiązania

```
# app.py
from flask import session

class UserPass:
    # ... - the code as in the previous version

    def login_user(self):
        db = get_db()
        sql_statement = 'select id, name, email, password, is_active, is_admin from users where
name=?'
        cur = db.execute(sql_statement, [self.user])
        user_record = cur.fetchone()

        if user_record != None and self.verify_password(user_record['password'], self.password):
            return user_record
```

```

        else:
            self.user = None
            self.password = None
            return None

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template('login.html', active_menu='login')
    else:
        user_name = '' if 'user_name' not in request.form else request.form['user_name']
        user_pass = '' if 'user_pass' not in request.form else request.form['user_pass']

        login = UserPass(user_name, user_pass)
        login_record = login.login_user()

        if login_record != None:
            session['user'] = user_name
            flash('Logon successful, welcome {}'.format(user_name))
            return redirect(url_for('index'))
        else:
            flash('Logon failed, try again')
            return render_template('login.html')

@app.route('/logout')
def logout():
    if 'user' in session:
        session.pop('user', None)
        flash('You are logged out')
        return redirect(url_for('login'))

# login.html
{% extends "base.html" %}
{% block title %}Logon{% endblock %}

{% block content %}
<div class="container">
  <form method="POST" action="{{url_for('login')}}">
    <div class="row mt-3">
      <div class="col-6 h1">Login</div>
    </div>

    <div class="row mt-3">
      <div class="col-2 col-form-label"><label for="user_name">User name</label></div>
      <div class="col-4">
        <input type="text" id="user_name" name="user_name" class="form-control">
      </div>
    </div>

    <div class="row mt-3">
      <div class="col-2 col-form-label"><label for="user_pas">Password</label></div>
      <div class="col-4">
        <input type="password" id="user_pass" name="user_pass" class="form-control">
      </div>
    </div>

    <div class="row mt-3">
      <div class="col-2"></div>
      <div class="col-4"><input type="submit" value="Login" class="btn btn-primary"></div>
    </div>
  </form>
</div>
{% endblock %}

# menu.html
<li class="nav-item">
  <a class="nav-link" {{ 'active' if active_menu=='login' }} href="{{ url_for('login') }}"
  tabindex="-1">Login</a>
</li>
<li class="nav-item">
  <a class="nav-link" {{ 'active' if active_menu=='logout' }} href="{{ url_for('logout') }}"
  tabindex="-1">
    Logout {{ session['user'] if 'user' in session }}</a>
</li>

```

## Dodawanie użytkowników – zadбай o wygodę użytkownika

### Notatka

- Moduł zarządzania użytkownikami to częsty wspólny element wielu aplikacji
- Dodając więcej powiązanych ze sobą funkcji do istniejącej aplikacji warto zdefiniować najpierw szkielet aplikacji. Pozwala to już od początku budować odnośniki z url\_for
- Kontrole danych powinny być wykonywane na każdym kroku – w przeglądarce, w aplikacji, w bazie danych itp. Pozwala to uniknąć wadliwych danych
- Po wykryciu błędu w danych, odsyłając użytkownika na stronę edycji danych, warto jest prezentować już wstępnie wypełnione w poprzednim kroku pola formularzy

### Laboratorium

1. Dodaj w swojej aplikacji hotelowej formularz pozwalający na dodawanie użytkowników. Możesz posłużyć się kodem z rozwiązania – po prostu wbuduj go w swój program

### Sprawdź się!

1. Na jakim etapie przyjmowania danych od użytkownika należy je sprawdzać?
2. Jak sprawić, aby użytkownik powracający do formularza, miał już częściowo wypełnione pola?

### Propozycja rozwiązania

```
# to add to app.py
@app.route('/users')
def users():
    return 'not implemented'

@app.route('/user_status_change/<action>/<user_name>')
def user_status_change(action, user_name):
    return 'not implemented'

@app.route('/edit_user/<user_name>', methods=['GET', 'POST'])
def edit_user(user_name):
    return 'not implemented'

@app.route('/user_delete/<user_name>')
def delete_user(user_name):
    return 'not implemented'

@app.route('/new_user', methods=['GET', 'POST'])
def new_user():
    if not 'user' in session:
        return redirect(url_for('login'))
    login = session['user']

    db = get_db()
    message = None
    user = {}

    if request.method == 'GET':
        return render_template('new_user.html', active_menu='users', user=user)
    else:
        user['user_name'] = '' if not 'user_name' in request.form else request.form['user_name']
        user['email'] = '' if not 'email' in request.form else request.form['email']
        user['user_pass'] = '' if not 'user_pass' in request.form else request.form['user_pass']

        cursor = db.execute('select count(*) as cnt from users where name = ?',
                             [user['user_name']])
        record = cursor.fetchone()
        is_user_name_unique = (record['cnt'] == 0)
```

```

cursor = db.execute('select count(*) as cnt from users where email = ?', [user['email']])
record = cursor.fetchone()
is_user_email_unique = (record['cnt'] == 0)

if user['user_name'] == '':
    message = 'Name cannot be empty'
elif user['email'] == '':
    message = 'email cannot be empty'
elif user['user_pass'] == '':
    message = 'Password cannot be empty'
elif not is_user_name_unique:
    message = 'User with the name {} already exists'.format(user['user_name'])
elif not is_user_email_unique:
    message = 'User with the email {} already exists'.format(user['email'])

if not message:
    user_pass = UserPass(user['user_name'], user['user_pass'])
    password_hash = user_pass.hash_password()
    sql_statement = '''insert into users(name, email, password, is_active, is_admin)
                        values(?,?,?, True, False);'''
    db.execute(sql_statement, [user['user_name'], user['email'], password_hash])
    db.commit()
    flash('User {} created'.format(user['user_name']))
    return redirect(url_for('users'))
else:
    flash('Correct error: {}'.format(message))
    return render_template('new_user.html', active_menu='users', user=user)

# to add to menu.html

<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"
    data-bs-toggle="dropdown" aria-expanded="false">
    Users
  </a>
  <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
    <li><a class="dropdown-item" href="{{ url_for('users') }}">Users</a></li>
    <li><a class="dropdown-item" href="{{ url_for('new_user') }}">New user</a></li>
  </ul>
</li>

# new_user.html

{% extends "base.html" %}
{% block content %}
<div class="container">
  <div class="row mb-3">
    <div class="col-5 h1">New user</div>
  </div>
  <form id="user_form" action="{{ url_for('new_user') }}" method="POST">

    <div class="row mb-3">
      <div class="col-1 col-form-label"><label for="user_name">User name</label></div>
      <div class="col-4">
        <input type="text" id="user_name" name="user_name" value="{{ user.user_name }}"
class="form-control"><br>
      </div>
    </div>
    <div class="row mb-3">
      <div class="col-1 col-form-label"><label for="email">Email</label></div>
      <div class="col-4">
        <input type="text" id="email" name="email" value="{{ user.email }}" class="form-
control"><br>
      </div>
    </div>
    <div class="row mb-3">
      <div class="col-1 col-form-label"><label for="user_pass">Password</label></div>
      <div class="col-4">
        <input type="password" id="user_pass" name="user_pass" class="form-control"><br>
      </div>
    </div>
    <div class="row mb-3">
      <div class="col-1"></div>
      <div class="col-2"><input type="submit" value="Send" class="btn btn-primary"></div>
    </div>
  </form>
</div>
{% endblock %}

```

## Lista użytkowników

### Notatka

- Budując połączenia odnośnikami między stronami aplikacji można, zależnie od sytuacji, korzystać z metody GET i przekazywania argumentu w definicji route, albo budować formularz, który niezbędne dane przekaże jako elementy formularza
- Zależnie od planowanego zachowania aplikacji, w przypadku otrzymania niepoprawnych parametrów, można zgłosić taką sytuację użytkownikowi albo obsłużyć ją „po cichu”. Dotyczy to zwłaszcza „niby błędów” n[. w postaci: usunięcie nieistniejącego już rekordu
- Wielokrotne korzystanie z tego samego kodu, to rzecz pożądana! Powtarzalny kod powinien być przenoszony do bibliotek funkcji, makr, klas itp.

### Laboratorium

1. Dodaj do swojej hotelowej aplikacji kod pozwalający na wyświetlenie listy użytkowników oraz usuwanie użytkownika. Kod znajdziesz w odpowiedziach.

### Sprawdź się!

1. Uzasadnij, dlaczego czasami aplikacja powinna informować użytkownika o wykonywanych czynnościach.
2. Uzasadnij, dlaczego czasami aplikacja powinna ukrywać przed użytkownikiem wykonywane czynności

### Propozycja rozwiązania

```
# users.html - the most important points only

{% block content %}
<!-- Modal -->
<div class="modal fade" id="confirmDeleteModal" tabindex="-1" aria-labelledby="exampleModalLabel"
aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">This entry will be deleted:</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
      </div>
      <div class="modal-body" id="idDeleteModalBody">
        <div>
          <div class="modal-footer">
            <form action="" method="GET" id="confirmDeleteForm">
              <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
              <button type="submit" class="btn btn-danger">Delete</button>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function () {
  // For A Delete Record Popup
  // This function is applied to all elements with class ending with ".delete-confirm"
  $(''.delete-confirm').click(function () {
    // get attributes of the found element
    var desc = $(this).attr('data-desc');
  });
});
</script>
```

```

        var url = $(this).attr('data-url');
        // the #... designates id of an element - change the text in the modal window
        $('#idDeleteModalBody').text(desc);
        $('#confirmDeleteForm').attr("action", url);
    });
});
</script>
<div class="container">
    <table class="table">
        <thead>
            <tr>
                <th scope="col">#</th>
                <th scope="col">User name</th>
                <th scope="col">Email</th>
                <th scope="col">Is active</th>
                <th scope="col">Is admin</th>
                <th scope="col">Actions</th>
            </tr>
        </thead>
        <tbody>
            {% for user in users %}
            <tr>
                <th scope="row">{{ user.id }}</th>
                <td>{{ user.name }}</td>
                <td>{{ user.email }}</td>
                <td></td>
                <td></td>
                <td>
                    <a href="{{ url_for('edit_user', user_name=user.name) }}"
                      class="btn btn-success btn-sm" role="button">Edit...</a>
                    <a type="button" class="btn btn-danger btn-sm delete-confirm"
                      data-bs-toggle="modal" data-bs-target="#confirmDeleteModal"
                      data-desc="{{ 'Delete user {}'.format(user.name) }}"
                      data-url="{{ url_for('delete_user', user_name=user.name) }}">
                        Delete
                    </a>
                </td>
            </tr>
            {%endfor%}
        </tbody>
    </table>
</div>
{% endblock %}

```

# app.py - only the most important functions:

```

@app.route('/users')
def users():
    db = get_db()
    sql_command = 'select id, name, email, is_admin, is_active from users;'
    cur = db.execute(sql_command)
    users = cur.fetchall()

    return render_template('users.html', active_menu='users', users=users)

@app.route('/user_delete/<user_name>')
def delete_user(user_name):
    if not 'user' in session:
        return redirect(url_for('login'))
    login = session['user']

    db = get_db()
    sql_statement = "delete from users where name = ? and name <> ?"
    db.execute(sql_statement, [user_name, login])
    db.commit()

    return redirect(url_for('users'))

```



## Edycja użytkownika

### Notatka

- To logika aplikacji decyduje o tym, jakie pola edycyjne są dostępne na formularzu
- Jako programista musisz „spodziewać się niespodziewanego”. Intuicyjnie, dane które przed chwilą zostały pobrane z bazy danych powinny się w tej bazie danych znajdować. Biorąc jednak pod uwagę, że systemy webowe mogą być wykorzystywane równocześnie przez wielu użytkowników wcale tak nie musi być. Dlatego warto sprawdzać, czy obiekty odczytywane z bazy danych nie mają wartości None.
- Na szczęście tego rodzaju błędy jak np. usunięcie nieistniejącego użytkownika można często zignorować lub tylko poinformować użytkownika o braku danych

### Laboratorium

1. Dodaj do aplikacji hotelowej kod pozwalający na edycję użytkownika. Skorzystaj z dołączonego kodu lub stwórz swój własny.

### Sprawdź się!

1. Jakie zalety i jakie wady ma obsługa formularza zademonstrowana w tej lekcji? Jaki problem może powstać, jeśli formularz prezentuje 100 pól i każde z nich będzie aktualizowane w bazie danych przy pomocy osobnego polecenia UPDATE?

### Propozycja rozwiązania

```
# app.py - only modified function
@app.route('/edit_user/<user_name>', methods=['GET', 'POST'])
def edit_user(user_name):
    db = get_db()
    cur = db.execute('select name, email from users where name = ?', [user_name])
    user = cur.fetchone()
    message = None

    if user == None:
        flash('No such user')
        return redirect(url_for('users'))

    if request.method == 'GET':
        return render_template('edit_user.html', active_menu='users', user=user)
    else:
        new_email = '' if 'email' not in request.form else request.form["email"]
        new_password = '' if 'user_pass' not in request.form else request.form['user_pass']

        if new_email != user['email']:
            sql_statement = "update users set email = ? where name = ?"
            db.execute(sql_statement, [new_email, user_name])
            db.commit()
            flash('Email was changed')

        if new_password != '':
            user_pass = UserPass(user_name, new_password)
            sql_statement = "update users set password = ? where name = ?"
            db.execute(sql_statement, [user_pass.hash_password(), user_name])
            db.commit()
            flash('Password was changed')

    return redirect(url_for('users'))
```

```

# edit_user.html

{% extends "base.html" %}

{% block title %}
    Edit user
{% endblock %}

{% block content %}
<div class="container">
    <div class="row mb-3">
        <div class="col-5 h1">Edit user {{ user.name }}</div>
    </div>
    <form id="user_form" action="{{ url_for('edit_user', user_name=user.name) }}" method="POST">

        <div class="row mb-3">
            <div class="col-1 col-form-label"><label for="email">Email</label></div>
            <div class="col-4">
                <input type="text" id="email" name="email" value="{{ user.email }}" class="form-
control"><br>
            </div>
        </div>

        <div class="row mb-3">
            <div class="col-1 col-form-label"><label for="user_pass">Password</label></div>
            <div class="col-4">
                <input type="password" id="user_pass" name="user_pass" class="form-control"><br>
            </div>

            <div class="row mb-3">
                <div class="col-1"></div>
                <div class="col-2"><input type="submit" value="Send" class="btn btn-primary"></div>
            </div>

    </form>
</div>
{% endblock %}

```

## Edycja uprawnień

### Notatka

- Edycja danych może być wykonywana nie tylko w oparciu o dane wprowadzane jawnie w formularzu
- Modyfikacja danych może być inicjowana przez dowolne akcje, w tym kliknięcie odnośników
- Aplikacje powinny minimalizować liczbę połączeń z bazą danych. Dzięki temu oszczędzasz zasoby serwerów obsługujących aplikację i bazę danych oraz przyspieszasz działanie aplikacji

### Laboratorium

1. Dodaj do aplikacji hotelowej funkcjonalność pozwalającą na nadawanie uprawnień w aplikacji. Odpowiedni kod znajdziesz w propozycjach rozwiązań

### Sprawdź się!

1. Jakie symbole można umieszczać w stronach generowanych we Flask (i nie tylko)
2. Jaka sztuczka pozwala zmieniać 0 na 1 i 1 na 0?
3. Co jest wadą łączenia wielu funkcjonalności w jednej funkcji?

### Propozycja rozwiązania

# app.py – only one function:

```
@app.route('/user_status_change/<action>/<user_name>')
def user_status_change(action, user_name):
    if not 'user' in session:
        return redirect(url_for('login'))
    login = session['user']

    db = get_db()

    if action == 'active':
        db.execute("""update users set is_active = (is_active + 1) % 2
                      where name = ? and name <> ?""",
                    [user_name, login])
        db.commit()
    elif action == 'admin':
        db.execute("""update users set is_admin = (is_admin + 1) % 2
                      where name = ? and name <> ?""",
                    [user_name, login])
        db.commit()

    return redirect(url_for('users'))
```

```
# users.html - missing part of the table
```

```
<td>
  <a href="{ url_for('user_status_change', action='active', user_name=user.name)
  }}">
    {% if user.is_active %}
      &check;
    {% else %}
      &#x25a2;
    {% endif %}
  </a>
</td>
<td>
  <a href="{ url_for('user_status_change', action='admin', user_name=user.name)
  }}">
    {% if user.is_admin %}
      &check;
    {% else %}
      &#x25a2;
    {% endif %}
  </a>
</td>
```

## Implementacja uprawnień w aplikacji

### Notatka

- Każda z metod powinna w jakiś sposób sprawdzać, czy jest uruchamiana w ramach zarejestrowanej sesji użytkownika
- W zależności od implementowanej logiki biznesowej, funkcje te mogą również sprawdzać poziom zdefiniowanych dla użytkownika uprawnień
- W przypadku braku uprawnień użytkownik może być przekierowywany do strony, która pozwala mu na zalogowanie się na odpowiednim koncie
- Menu powinno pokazywać tylko opcje dostępne dla użytkownika z uwzględnieniem jego uprawnień. W tym celu można wykorzystać warunkowe instrukcje Jinja

### Laboratorium

1. Dodaj do klasy UserPass nowe właściwości (is\_valid, is\_admin) oraz metodę
2. Zaimplementuj w metodach aplikacji sprawdzanie uprawnień
3. Dostosuj menu do uprawnień użytkownika

### Sprawdź się!

1. Dlaczego należy sprawdzać uprawnienia w funkcjach, które dla danego użytkownika są niedostępne w menu?
2. Jakie inne niż pokazane w filmie funkcjonalności związane z kontem użytkownika mogły by być zaimplementowane w aplikacji pozwalającej pracownikom hotelowym na zgłaszanie awarii?

### Propozycja rozwiązania

```
# app.py – properties and method for UserPass class:
class UserPass:
    def __init__(self, user='', password=''):
        self.user = user
        self.password = password
        self.email = ''
        self.is_valid = False
        self.is_admin = False

    def get_user_info(self):
        db = get_db()
        sql_statement = 'select name, email, is_active, is_admin from users where name=?'
        cur = db.execute(sql_statement, [self.user])
        db_user = cur.fetchone()

        if db_user == None:
            self.is_valid = False
            self.is_admin = False
            self.email = ''
        elif db_user['is_active']!=1:
            self.is_valid = False
            self.is_admin = False
            self.email = db_user['email']
        else:
            self.is_valid = True
            self.is_admin = db_user['is_admin']
            self.email = db_user['email']
```

```

# app.py - code to add to functions - non admin access:
    login = UserPass(session.get('user'))
    login.get_user_info()
    if not login.is_valid:
        return redirect(url_for('login'))

# app.py - code to add to functions - admin access
    login = UserPass(session.get('user'))
    login.get_user_info()
    if not login.is_valid or not login.is_admin:
        return redirect(url_for('login'))

# menu.html - code to show/hide different menu options (only 2 examples):
    {% if login.is_valid: %}
    <li class="nav-item">
        <a class="nav-link" {{ 'active' if active_menu=='notification' }}" href="{{
url_for('notification') }}">New notification</a>
    </li>
    {% endif %}
    <li class="nav-item">
        <a class="nav-link" {{ 'active' if active_menu=='about' }}" href="{{ url_for('about') }}"
tabindex="-1">About this app</a>
    </li>
    {% if login.is_valid and login.is_admin: %}
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"
        data-bs-toggle="dropdown" aria-expanded="false">
            Users
        </a>
        <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item" href="{{ url_for('users') }}">Users</a></li>
            <li><a class="dropdown-item" href="{{ url_for('new_user') }}">New user</a></li>
        </ul>
    </li>
    {% endif %}

```

## Projekt – 03 – Aplikacja CRUD

Pod pojęciem aplikacji CRUD rozumiemy aplikację, która pozwala:

- **CR**eat - Dodawać nowe dane
- **U**pdate – Modyfikować dane
- **D**ele - Usuwać dane

Na tym etapie kursu, jesteś już w stanie stworzyć aplikację pracującą wg tego scenariusza. Nie kłamie!

Jeśli więc tylko masz ochotę, to zbuduj kolejny program, które zaimplementuje scenariusz CRUD. Oto kilka pomysłów:

- Cennik w kwiaciarni
- Inwentaryzacja domowego sprzętu technicznego
- Ewidencja rachunków za media
- Spis rzeczy jakie chcesz zrobić przed ukończeniem XX urodzin
- Spis kart gwarancyjnych domowych sprzętów
- ...

Nie narzucam, żadnej dodatkowej formy takiej aplikacji. Warto byłoby w niej uwzględnić te elementy, które do tej pory poznaliśmy.

Może myślisz, że to mało kreatywne, bo podobne aplikacje już pisaliśmy wcześniej? To sprawdź 😊  
Każdy błąd, który popełnisz i którego znaczenia od razu nie odgadniesz, to nowe doświadczenie. Nie na darmo mówi się, że „mistrzowie popełnili więcej błędów, niż początkujący adepci podjęli prób”.

Możesz też wpaść na pomysł przerobienia aktualnej aplikacji na nową. W sumie też można... W takim przypadku liczyć się z wieloma błędami wynikającymi z typowego kopiuj/wklej

Pusta strona wstawiona celowo



## Flask SQLAlchemy

### Notatka

- SQLAlchemy to moduł, który pozwala programiście skupić się na kodzie aplikacji, a nie na kodzie pracującym z bazą danych. Instalację wykonasz uruchamiając:

```
pip install flask-sqlalchemy
```

- Flask-SQLAlchemy to moduł, który ułatwia korzystanie z SQLAlchemy z poziomu Flask
- Flask SQLAlchemy znajduje się w module Flask-SQLAlchemy
- Do korzystania z baz danych, nierzadko trzeba dodatkowo instalować inne moduły pythonowe oraz sterowniki dedykowane dla określonej bazy danych
- Aby korzystać z Flask SQLAlchemy zaimportuj odpowiedni moduł:

```
from flask_sqlalchemy import SQLAlchemy
```

- Flask SQLAlchemy wymaga parametrów, które można umieścić w pliku, a potem go załadować

```
app.config.from_pyfile('config.cfg')
```

- Aby korzystać z SQLAlchemy, trzeba utworzyć instancję SQLAlchemy

```
db = SQLAlchemy(app)
```

- Definiując klasy dziedziczące z db.Model automatycznie tworzysz definicje tabel

```
class Vendor(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    name = db.Column(db.String(50))  
    priority = db.Column(db.Integer)  
    active = db.Column(db.Boolean)
```

- Tworzenie table odbywa się po wykonaniu polecenia db.create\_all()

```
db.create_all()
```

### Laboratorium

1. Skonfiguruj nowy projekt.
2. zainstaluj w nowym środowisku wirtualnym moduły
  - a. Flask
  - b. Flask-SQLAlchemy
3. No korytarzach ekskluzywnego hotelu znajdują się antyczne rzeźby i obrazy. Twoim zadaniem jest stworzenie aplikacji, która pozwoli przechowywać dane o autorach i ich dziełach. Wprawdzie nie napiszemy tu całej aplikacji, ale tym scenariuszem posłużymy się w celu ćwiczenia pracy z SQLAlchemy
4. Stwórz prosty prototyp aplikacji Flask korzystającej z SQLAlchemy
5. Utwórz klasę Author i powiązaną tabelę author o kolumnach:
  - a. Id – liczba całkowita, klucz podstawowy
  - b. Name – imię i nazwisko/pseudo autora – napis 50 znaków

- c. Special – pole logiczne informujące o tym, czy ten autor jest jakoś powiązany z naszym hotelem (np. bywał tu, albo przynajmniej zamieszkiwał okolice)

### Sprawdź się!

1. Wymień kilka zalet i wad korzystania z SQLAlchemy
2. Co jeszcze należy zainstalować na komputerze, aby korzystać z baz danych?
3. Z jakiej klasy powinny dziedziczyć klasy opisujące tabele SQLAlchemy?
4. Jaka metoda odpowiada za fizyczne utworzenie tabel w bazie danych przez SQLAlchemy?

### Propozycja rozwiązania

```
#app.py
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config.from_pyfile('config.cfg')
db = SQLAlchemy(app)

class Author(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    special = db.Column(db.Boolean)

@app.route('/')
def index():
    db.create_all()

    return '''Hello Flask-SQLAlchemy'''

if __name__ == '__main__':
    app.run()

# config.cfg

SQLALCHEMY_DATABASE_URI='sqlite:///c:\\data\\museum.db'
SQLALCHEMY_TRACK_MODIFICATIONS=False
```

## Dodawanie i pobieranie danych z bazy danych

### Notatka

- Aby dodać do tabeli nowy rekord, utwórz po prostu obiekt będący instancją klasy odpowiadającej danej tabeli. Następnie dodaj ten obiekt do sesji i zapisz zmiany poleceniem commit:

```
v1 = Vendor(id=1, name='Microsoft', discount=0, active=True)
db.session.add(v1)
db.session.commit()
```

- Aby odczytać wszystkie dane z tabeli można posłużyć się poleceniem query.all()

```
vendors = Vendor.query.all()
```

- Dodając do definicji klasy metodę repr(self) możesz samodzielnie zdefiniować tekst, jaki jest zwracany podczas konwersji obiektu do tekstu:

```
def __repr__(self):
    return 'Vendor: {}'.format(self.id, self.name)
```

- Aby w wygodny sposób testować działanie SQLAlchemy możesz z app.py zaimportować odpowiednie obiekty i pracować z nimi w sesji Pythona

```
from app import app, Vendor, db
```

### Laboratorium

- Do klasy Author dodaj metodę \_\_repr\_\_ zwracającą napis złożony z id, name i special
- Otwórz sesję Pythona i zaimportuj app, Author i db z app
- Utwórz obiekty:
  - id=1, name='Salvador Dali', special=False
  - id=2, name='Pablo Picasso', special=False
  - id=3, name='Paul Cezane', special=True
- Dodaj te obiekty do sesji i zapisz je do bazy danych
- Pobierz wszystkich aktorów z bazy danych

### Sprawdź się!

- Jak dodaje się rekord do bazy danych z wykorzystaniem SQLAlchemy?
- Jak pobiera się wszystkie rekordy?
- Do czego służy metoda \_\_repr\_\_?
- Jak testować działanie funkcji i obiektów w app.py bez uruchamiania aplikacji Flask?

## Propozycja rozwiązania

# app.py – class definition

```
class Author(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    special = db.Column(db.Boolean)

    def __repr__(self):
        return '<id: {}, name: {}, special: {}>'.format(self.id, self.name,
self.special)
```

# in python session:

```
>>> from app import app, db, Author
>>> dali = Author(id=1, name='Salvador Dali', special=False)
>>> db.session.add(dali)
>>> picasso = Author(id=2, name='Pablo Picasso', special=False)
>>> db.session.add(picasso)
>>> cezane = Author(id=3, name='Paul Cezane', special=True)
>>> db.session.add(cezane)
>>> db.session.commit()
>>> Author.query.all()
[<id: 1, name: Salvador Dali, special: False>, <id: 2, name: Pablo Picasso, special:
False>, <id: 3, name: Paul Cezane, special: True>]
>>>
```

## Filtrowanie danych

### Notatka

- **Vendor.query.filter\_by(discount=5).all()** – pobranie tych Vendor dla których discount wynosi 5. Kończąc wywołanie metodą all zwracana jest lista wszystkich pasujących obiektów, a po zakończeniu przez first() tylko jeden obiekt. Gdy brak pasujących obiektów zwracane jest None. Polecenie nie radzi sobie z niektórymi warunkami, a stosowane operatory są „niepythonowe”
- **Vendor.query.filter(Vendor.discount==5).all()** – działanie j.w. ale bardziej elastyczna składnia. Obiekt query i obiekt zwracany przez filter() są typu BaseQuery, co pozwala na składanie zapytania z kolejnych segmentów. Szukając wartości None odnajdziesz rekordy z wartością NULL
- **Vendor.query.filter(Vendor.name.like('%s%')).all()** – znajdź tych Vendor, gdzie w name występuje s. Warunek przeciwny uzyskuje się przez dodanie znaku ~ (tylda) przed warunkiem w filter()
- **Vendor.query.filter(Vendor.id.in\_([1,2,3])).all()** – znajdź tych Vendor, gdzie id wynosi 1, 2 lub 3
- Jeśli w metodzie filter po przecinku dodasz kolejny warunek, to są one łączone operatorem logicznym and, dokładnie tak samo jak w przypadku składania kilku wywołań filter()
- **Vendor.query.filter(db.and\_(Vendor.id > 1, Vendor.id < 6)).all()** – łączenie warunków przez and
- **Vendor.query.filter(db.or\_(Vendor.id == 1, Vendor.id >= 4)).all()** – łączenie warunków przez or

### Laboratorium

1. W sesji Pythona załaduj z app.py obiekty app, db i Author
2. Dodaj do tabeli author rekordy:
  - a. id=4, name='Cloud Monet',special=False
  - b. id=5, name='Andy Warhol'
  - c. id=6, name='Frida Kahlo'
3. Wyświetl wszystkie rekordy
4. Napisz zapytania, które zwrócą:
  - a. Autorów z identyfikatorami 1 lub 3 lub 5
  - b. Autorów, którzy nie mają określonej wartości w polu special
  - c. Autorów, którzy mają określoną wartość w polu special
  - d. Autorów, którzy w polu name mają literkę „u”
  - e. Autorów, którzy w polu name mają literkę „w” i nie mają określonej wartości w polu special
  - f. Autorów, którzy w polu name mają literkę „w” i mają znaną wartość w polu special
  - g. Autorów, którzy w polu name nie mają literki „w” i nie mają określonej wartości w polu special
  - h. Autorów, którzy w polu name nie mają literki „w” i mają znaną wartość w polu special
  - i. Autorów, którzy w polu name mają literkę „w” lub „u”

## Sprawdź się!

1. Jakiego typu jest `Vendor.query` ?
2. Co jest zwracane, jeśli warunki zdefiniowane w `filter()` nie pasują do żadnego rekordu?
3. Jak zaprzeczyć warunek zdefiniowany w `filter()` ?
4. Przypomnij sobie 3 metody na budowanie wyrażeń filtrujących AND.

## Propozycja rozwiązania

```
# in python session:
>>> from app import app, db, Authors
>>> monet = Author(id=4, name='Cloud Monet', special=False)
>>> db.session.add(monet)
>>> warhol = Authors(id=5, name='Andy warhol')
>>> db.session.add(warhol)
>>> kahlo = Author(id=6, name='Frida Kahlo')
>>> db.session.add(kahlo)
>>> db.session.commit()

>>> Author.query.all()
>>> Author.query.filter(Author.id.in_([1,3,5])).all()
>>> Author.query.filter(Author.special == None).all()
>>> Author.query.filter(Author.special != None).all()
>>> Author.query.filter(Author.name.like('%u%')).all()
>>> Author.query.filter(db.and_(Author.name.like('%w%'), Author.special==None)).all()
>>> Author.query.filter(db.or_(Author.name.like('%w%'), Author.special==None)).all()
>>> Author.query.filter(db.and_(~Author.name.like('%w%'), Author.special==None)).all()
>>> Author.query.filter(db.and_(~Author.name.like('%w%'), Author.special!=None)).all()
>>> Author.query.filter(db.or_(Author.name.like('%w%'), Author.name.like('%u%'))).all()
```

## Sortowanie, zliczanie i ograniczanie liczby rekordów

### Notatka

- `Vendor.query.order_by(Vendor.name).all()` - zwraca dane posortowane w kolejności wg nazwy
- `Vendor.query.filter(Vendor.id > 3).order_by(Vendor.name).all()` – j.w. ale zbiór danych jest dodatkowo filtrowany
- `Vendor.query.order_by(Vendor.discount.desc()).all()` – sortowanie danych w odwróconej kolejności
- `Vendor.query.order_by(Vendor.active, Vendor.name).all()` – sortowanie wg dwóch pól na raz
- `Vendor.query.order_by(Vendor.discount.desc()).limit(3).all()` – zwracane są tylko 3 pierwsze rekordy posortowane pod względem malejącej wartości w kolumnie discount
- `Vendor.query.order_by(Vendor.discount.desc()).offset(10).limit(5).all()` – po posortowaniu danych wg malejącej zniżki i po opuszczeniu pierwszych 10-ciu rekordów wyświetl następne 5
- `Vendor.query.count()` – wyznaczy liczbę rekordów zwracanych przez zapytanie

### Laboratorium

1. Posortuj autorów wg nazwy
2. Posortuj autorów wg nazwy malejąco
3. Posortuj autorów wg nazwy, ale opuść tych, którzy nie mają znanej wartości w kolumnie special
4. Wyświetl 5-ciu pierwszych autorów posortowanych wg identyfikatora id
5. Wyświetl kolejną porcję 5-ciu rekordów po opuszczeniu 5, sortując wg id
6. Wyznacz liczbę rekordów w Author
7. Wyznacz liczbę rekordów w Author, gdzie wartość w kolumnie special jest nieznana
8. Wyznacz liczbę rekordów w Author, gdzie wartość w kolumnie special jest znana

### Sprawdź się!

1. Jeśliby poskładać z następujących słów polecenie SQLAlchemy, to w jakiej występowałyby one kolejności: limit, offset query, order by, filter
2. Jakie metody mogą kończyć „łańcuch” poskładanych funkcji w wyrażeniu SQLAlchemy?

## Propozycja rozwiązania

```
Author.query.order_by(Author.name).all()
Author.query.order_by(Author.name.desc()).all()
Author.query.filter(Author.special != None).order_by(Author.name).all()
Author.query.order_by(Author.id).limit(5).all()
Author.query.order_by(Author.id).offset(5).limit(5).all()
Author.query.count()
Author.query.filter(Author.special == None).count()
Author.query.filter(Author.special != None).count()
```



## Dodawanie i usuwanie danych

### Notatka

- Aby zmodyfikować rekord
  - Pobierz go do pamięci wykonując odpowiednie polecenie query
  - Wykonaj modyfikację w pamięci, zmieniając właściwości obiektu
  - Wyślij modyfikację do bazy danych korzystając z `db.session.commit()`

```
logi = Vendor.query.filter(Vendor.id == 6).first()
logi.discount = 10
db.session.commit()
```

- Aby usunąć rekord:
  - Pobierz go do pamięci wykonując odpowiednie polecenie query
  - Usuń rekord z sesji
  - Wyślij polecenie do bazy danych korzystając z `db.session.commit()`

```
logi = Vendor.query.filter(Vendor.id == 6).first()
db.session.delete(logi)
db.session.commit()
```

- Modyfikacja wielu rekordów na raz jest możliwa z poziomu pythona, ale biorąc pod uwagę wydajność takiej operacji, lepiej wykonać ją wykorzystując odpowiednie polecenie UPDATE na bazie danych bezpośrednio lub za pośrednictwem innych metod udostępnionych w SQLAlchemy

### Laboratorium

1. Pobierz rekord dla Pablo Picasso
2. Zmień wartość pola `special` na `True`
3. Pobierz rekord dla Paul Cezane
4. Zmień wartość pola `special` na `False`
5. Zapisz zmiany do bazy danych i sprawdź aktualne wartości
6. Pobierz rekord dla Andy Warhol
7. Usuń ten rekord i sprawdź czy rekord rzeczywiście został usunięty

### Sprawdź się!

1. Masz zamiar usunąć rekord. Wykonujesz polecenie `db.session.delete`, ale rekord ciągle jest w bazie danych. O czym najprawdopodobniej zapominasz?
2. Tabela ma 100 tys. rekordów. Chcesz je wszystkie zmodyfikować. Uzasadnij, dlaczego prezentowana na lekcji metoda nie będzie idealnym rozwiązaniem.

## Propozycja rozwiązania

```
Author.query.all()

picasso = Author.query.filter(Author.name == 'Pablo Picasso').first()
picasso.special = True

cezane = Author.query.filter(Author.name == 'Paul Cezane').first()
cezane.special = False

db.session.commit()
Author.query.all()

warhol = Author.query.filter(Author.name == 'Andy Warhol').first()
db.session.delete(warhol)
db.session.commit()

Author.query.all()
```

## Relacyjne bazy danych

### Notatka

- Relacje wiążą między sobą tabele
- W tabeli, którą chcesz powiązać z inną tabelą należy zdefiniować kolumnę, jako klucz obcy (tutaj vendor\_id w klasie Product)
- Aby w wygodny sposób odwoływać się z obiektów jednej klasy do drugiej tworzy się tzw. Backreference

```
class Vendor(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    discount = db.Column(db.Integer)
    active = db.Column(db.Boolean)

    products = db.relationship('Product', backref='vendor', lazy='dynamic')

class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    vendor_id = db.Column(db.Integer, db.ForeignKey('vendor.id'))
```

- Dodając powiązany rekord można to robić obiektowo:

```
apple = Vendor.query.filter(Vendor.name=='APPLE').first()
prod_a1 = Product(id=301, name='IPHONE', vendor=apple)
```

- Można łatwo zobaczyć powiązane ze sobą rekordy, np. produkty jednego dostawcy lub informacje o dostawcy konkretnego produktu:

```
apple.products.all()
prod_a1.vendor
```

### Laboratorium

1. Do modelu z poprzedniej lekcji dodaj klasę ArtWork o następujących polach:
  - a. id, identyfikator rekordu, liczba całkowita, klucz podstawowy
  - b. name, nazwa dzieła, napis maksymalnie 50 znaków
  - c. author\_id, pole wskazujące na autora, liczba całkowita, klucz obcy do authors.id
2. Korzystając ze sprytnych metod SQLAlchemy dodaj rekordy:
  - a. 11, "The Truth of Nature" – autorstwa Cloud Monet
  - b. 12, "Boquet of Sunflowers" –Cloud Monet
  - c. 21, "Girafe en Feu" – Salvador Dali
  - d. 22, "Sant Anthony" – Salvador Dali
3. Wyświetl obrazy Salvadora Dali
4. Wyświetl informację o autorze obrazu z identyfikatorem 21
5. Zmień autora obrazu 12 na Salvadora Dali (jeśli burzy to Twój spokój ducha, to nie commituj tej zmiany) 😊

### Sprawdź się!

1. Jaka jest przewaga używania modelu obiektowego przy dostępie do danych zapisanych w tabelach relacyjnych?
2. Czy model obiektowy pozwala zupełnie wyeliminować potrzebę tworzenia klucza obcego w bazie danych?

### Propozycja rozwiązania

#### Solution placeholder

```
from app import db, app, Author, ArtWork
db.create_all()

monet = Author.query.filter(Author.name=='Cloud Monet').first()
nature = ArtWork(id=11, name='The Truth of Nature', author=monet)
sunflowers = ArtWork(id=12, name='Bouquet of Sunflowers', author=monet)

dali = Author.query.filter(Author.name=='Salvador Dali').first()
girafe = ArtWork(id=21, name='Girafe En Feu', author=dali)
anthony = ArtWork(id=22, name='Sant Anthony', author=dali)

db.session.add(nature)
db.session.add(sunflowers)
db.session.add(girafe)
db.session.add(anthony)
db.session.commit()

dali.artwork.all()

ArtWork.query.filter(ArtWork.id == 21).first().author

ArtWork.query.filter(ArtWork.id == 12).first().author = dali
db.session.commit()
```

## Porównanie metod pracy z bazą danych

### Notatka

- Zależnie od tego, czy chcesz wykorzystać SQLAlchemy czy tradycyjny i bezpośredni dostęp do bazy danych musisz zaimportować inne moduły. Jeśli korzystasz z dedykowanych baz danych i tak nie minie cię instalacja i konfiguracja sterowników
- SQLAlchemy wymaga zbudowania modelu obiektowego mapowanego do struktur bazy danych, zaś tradycyjny model wykorzystuje dedykowane procedury na budowanie połączenia z bazą
- Jednolinijkowy i stosunkowo prosty kod do pobrania rekordów z bazy danych, w podejściu tradycyjnym zazwyczaj wiąże się ze zbudowaniem połączenia, zbudowaniem zapytania, wykonania zapytania i dopiero pobrania danych do zmiennych
- Pobrane do pamięci rekordy łatwiej jest wykorzystywać, gdy są one obiektami – to zaleta SQLAlchemy
- SQLAlchemy daje też możliwość łatwego wypełnienia bazy danych strukturami tabel. W podejściu tradycyjnym trzeba uruchomić skrypt SQL
- W tradycyjnym SQL można wykonywać „sztuczki”, które polegają na takim skonstruowaniu polecenia SQL, że w szczególnym przypadku nie zostaną zmodyfikowane żadne rekordy. W przypadku SQLAlchemy, rekord trzeba tak czy siak pobrać. Po stwierdzeniu, że pasujących danych nie ma, należy sprawdzić, czy otrzymany obiekt to pełnowartościowy rekord, czy może None. To skutek przesunięcia logiki biznesowej aplikacji do kodu aplikacji.

### Laboratorium

1. Jeśli masz ochotę, to wróć ponownie do hotelowej mini aplikacji do zgłoszeń. Aktualnie ta aplikacja korzysta z kodu SQL w tradycyjny sposób. Zmień to i wykorzystaj SQLAlchemy
  - a. Pamiętaj o zaimportowaniu wymaganych modułów
  - b. Stworzeniu klas odpowiadających tabelom w bazie danych
  - c. Modyfikacji funkcji init, która w razie potrzeby utworzy tabele w bazie danych
  - d. Przeglądzie całego kodu i zamianie instrukcji wyszukiwanych i pobierających dane, jak również instrukcji wstawiających, modyfikujących i usuwających rekordy
  - e. Jeśli istnieje możliwość, że wyrażenia przesyłane w filter mogą spowodować, że zwrócony zostanie pusty zbiór danych, to przed dalszym wykorzystaniem tych danych, sprawdź czy nie masz do czynienia z obiektem None
  - f. Jak to przy refaktoryzacji gotowej aplikacji bywa, przygotuj się na liczne usterki. Testy tak przebudowanej aplikacji są krytyczne!
2. Na zakończenie porównaj kod aplikacji przed i po zmianach

### Sprawdź się!

1. Z punktu widzenia łatwości programowania, co wydaje się lepsze? Podejście tradycyjne, czy SQLAlchemy?
2. Z punktu widzenia wydajności, co wydaje się lepsze? Podejście tradycyjne, czy SQLAlchemy?
3. Czy krótszy kod jest zawsze lepszy?

## Propozycja rozwiązania

```
# some snippets to reuse in your solution

# old - get user information -----
db = get_db()
sql_statement = 'select id, name, email, password, is_active, is_admin from users where name=?'
cur = db.execute(sql_statement, [self.user])
user_record = cur.fetchone()

# new
user_record = User.query.filter(User.name == self.user).first()

# old - get all users -----
db = get_db()
sql_command = 'select id, name, email, is_admin, is_active from users;'
cur = db.execute(sql_command)
users = cur.fetchall()

# new
users = User.query.all()

# old - update is_active for a user, but not for a current user -----
db.execute("""update users set is_active = (is_active + 1) % 2
           where name = ? and name <> ?""",
           [user_name, login.user])

# new
user = User.query.filter(User.name == user_name, User.name != login.user).first()
if user:
    user.is_active = (user.is_active + 1) % 2
    db.session.commit()

# old - update email -----
sql_statement = "update users set email = ? where name = ?"
db.execute(sql_statement, [new_email, user_name])
db.commit()

# new
user.email = new_email
db.session.commit()

# old - delete a user -----
db = get_db()
sql_statement = "delete from users where name = ? and name <> ?"
db.execute(sql_statement, [user_name, login.user])
db.commit()

# new
user = User.query.filter(User.name==user_name, User.name != login.user).first()
if user:
    db.session.delete(user)
    db.session.commit()

# old - add a new user -----
sql_statement = '''insert into users(name, email, password, is_active, is_admin)
                  values(?,?,?, True, False);'''
db.execute(sql_statement, [user['user_name'], user['email'], password_hash])
db.commit()

# new
new_user = User(name=user['user_name'], email=user['email'], password=password_hash,
                is_active=True, is_admin=False)
db.session.add(new_user)
db.session.commit()
```

## Flask-WTF – instalacja i pierwszy formularz

### Notatka

- Moduł WTForms i Flask-WTF pozwalają tworzyć kontrolki HTML za pomocą kodu Pythona
- Flask-WTF zainstalujesz za pomocą polecenia pip. Automatycznie zainstaluje się WTForms

### Pip install Flask-WTF

- Z flask\_wtf należy zaimportować FlaskForm. Zależnie od pól planowanego formularza należy jeszcze zaimportować odpowiednie klasy z wtforms:

```
from flask_wtf import FlaskForm
from wtforms import StringField, IntegerField, BooleanField
```

- Zamiast definiować formularz definiuje się klasę

```
class BookForm(FlaskForm):
    title = StringField('Book title')
    amount = IntegerField('Amount')
    available = BooleanField('Available')
```

- Na formularzu zamiast kodu html umieszcza się wyrażenia w postaci:

```
{{ form.title.label }}      {{ form.title }}
```

- W funkcji widoku za pomocą funkcji validate\_on\_submit można zdecydować o przyjęciu danych

### Laboratorium

1. Zdarza Ci się jeździć pociągiem? Zrobimy formularz do rejestrowania opóźnień 😊
2. Przygotuj klasę TrainInfo z polami:
  - a. Train\_number – pole do wprowadzenia napisu
  - b. Is\_delayed – pole logiczne mówiące o tym, że pociąg był opóźniony
  - c. Delay\_minutes – pole na wprowadzenie wielkości opóźnienia pociągu
  - d. Delay\_reason – lista rozwijana (SelectField) z wartościami do wyboru:  
choices=['None', 'Weather', 'Failure', 'Other']
3. Funkcja widoku, po przyjęciu parametrów ma je po prostu wyświetlić

### Sprawdź się!

1. Jaki problem pomoże rozwiązać Flask-WTF i WTForms?
2. Jak definiuje się formularz?
3. Jakie typy pól mogą się znajdować w formularzu?
4. Jaka funkcja sprawdza, czy dane formularza są poprawne?

## Propozycja rozwiązania

```
# app.py
from flask import Flask, render_template, url_for
from flask_wtf import FlaskForm
from wtforms import StringField, IntegerField, BooleanField, SelectField

app = Flask(__name__)
app.config['SECRET_KEY'] = 'ACompl1cat3dText.'

class TrainInfo(FlaskForm):
    train_number = StringField('Train number')
    is_delayed = BooleanField('Is delayed')
    delay_minutes = IntegerField('Delay in minutes')
    delay_reason = SelectField('Delay reason', choices=['None', 'Weather', 'Failure',
'other'])

@app.route('/', methods=['POST', 'GET'])
def index():
    form = TrainInfo()
    if form.validate_on_submit():
        return f'''<H1>Hello</H1>
        <ul>
            <li>{form.train_number.label}: {form.train_number.data}</li>
            <li>{form.is_delayed.label}: {form.is_delayed.data}</li>
            <li>{form.delay_minutes.label}: {form.delay_minutes.data}</li>
            <li>{form.delay_reason.label}: {form.delay_reason.data}</li>
        </ul>'''
    return render_template('index.html', form=form)

if __name__ == '__main__':
    app.run()

# index.html

<form method="POST" action="{{ url_for('index') }}">
    {{ form.csrf_token }}
    {{ form.train_number.label }} {{ form.train_number }} <br/>
    {{ form.is_delayed.label }} {{ form.is_delayed }} <br/>
    {{ form.delay_minutes.label }} {{ form.delay_minutes }} <br/>
    {{ form.delay_reason.label }} {{ form.delay_reason }} <br/>
    <input type="submit" value="GO!">
</form>
```



## Sprawdzanie poprawności danych (validators)

### Notatka

- Walidatory pozwalają na weryfikację danych wprowadzonych przez użytkownika, samo sprawdzenie wykonuje się zazwyczaj funkcją `validate_on_submit()`, która dodatkowo pozwala rozstrzygnąć, czy funkcja pracuje w metodzie POST:

```
if form.validate_on_submit():
```

- Jednym z pól podlegającym weryfikacji jest `csrf_token` pozwalający na stwierdzenie czy formularz jest rzeczywiście wysyłany ze strony wygenerowanej wcześniej przez aplikację. Pole to dodaje się w szablonie Jinja:

```
{{ form.csrf_token }}
```

- Poszczególne pola mogą mieć przypisane własne, bardziej wyspecjalizowane walidatory, np. `DataRequired`, `Email`, `EqualTo`, `IPAddress`, `Length`, `NumberRange`, `Regexp`, `URL` itp.
- Aby korzystać z walidatorów zaimportuj je:

```
from wtforms.validators import DataRequired, Length, ValidationError
```

- Deklaracja pól klasy definiującej formularz powinna zawierać parametr będący listą walidatorów

```
title = StringField('Book title', validators=[DataRequired("Enter book title"),  
                                             Length(min=5, max=50, message="The title must have 5-50 charactes")],
```

- Weryfikacja formularza odbywa się dwupoziomowo: w przeglądarce (co sprawia, że strona jest „user friendly”, ale łatwo ją obejść) i w aplikacji (mniej „user friendly”, ale pewniejsza)
- Można tworzyć własne walidatory, które implementują logikę budowanej aplikacji:

```
class BookForm(FlaskForm):  
    def validate_amount_even(form, field):  
        if field.data % 2 != 1:  
            raise ValidationError("This number must be even!")  
  
    amount = IntegerField('Amount', validators=[DataRequired(message="Enter amount"),  
                                              validate_amount_even], default=101)
```

### Laboratorium

1. Do formularza rejestrującego opóźnienia pociągów dodaj:
  - a. Dla pola `train_number` walidator `DataRequired`
  - b. Dla pola `delay_minutes` walidator `NumberRange` (z parametrem `min=0`)
2. Dodaj custom validator dla `train_number` określający, że numer pociągu ma się zaczynać od 2 liter.
3. Zadbaj o to, aby zdefiniowane komunikaty o błędach były wyświetlane użytkownikowi. Ponieważ ten kod się powtarza, rozważ zbudowanie odpowiedniego makra Jinja

## Sprawdź się!

1. Gdzie odbywa się sprawdzenie poprawności danych wprowadzonych przez użytkownika?
2. Czy można ufać walidacji wykonywanej po stronie przeglądarki?
3. Co to jest CSRF token i do czego on służy?

## Propozycja rozwiązania

```
# app.py - only fragments:
from wtforms.validators import DataRequired, NumberRange, ValidationError
...
class TrainInfo(FlaskForm):
    def start_with2letters(form, field):
        if not (len(field.data) > 2 and field.data[0:2].isalpha()):
            raise ValidationError('Train number must start with 2 letters')

    train_number = StringField('Train number',
                               validators=[DataRequired('Enter train number'), start_with2letters])
    is_delayed = BooleanField('Is delayed')
    delay_minutes = IntegerField('Delay in minutes', validators=[DataRequired('Enter delay'),
                                                                NumberRange(min=0, message='The delay must be a number >= 0')])
    delay_reason = SelectField('Delay reason', choices=['None', 'Weather', 'Failure', 'Other'])

# macros.html
{% macro show_validation_results(field) %}
<ul>
    {% for error in field.errors %}
        <li>{{ error }}</li>
    {% endfor %}
</ul>
{% endmacro %}

# index.html
{% from "macros.html" import show_validation_results %}
<form method="POST" action="{{ url_for('index') }}">
    {{ form.csrf_token }}
    {{ form.train_number.label }} {{ form.train_number }} <br/>
    {{ show_validation_results(form.train_number) }}
    {{ form.is_delayed.label }} {{ form.is_delayed }} <br/>
    {{ form.delay_minutes.label }} {{ form.delay_minutes }} <br/>
    {{ show_validation_results(form.delay_minutes) }}
    {{ form.delay_reason.label }} {{ form.delay_reason }} <br/>

    <input type="submit" value="GO!">
</form>
```

## Zapisywanie danych z formularzy w obiektach

### Notatka

- Klasa definiująca formularz może przypisywać polom formularza domyślne wartości:

```
class BookForm(FlaskForm):  
    title = StringField('Book title', default='Unknown')
```

- Aby jawnie zmienić wartości prezentowane w formularzu można je inicjować „na piechotę”:

```
form = BookForm(title='European Kings', amount=99, available=True)
```

- Jeśli w programie masz już obiekt z danymi i jeżeli pola formularza nazywają się tak samo, jak pola właściwości w tabelach, to można inicjować formularz tym obiektem:

```
class Book:  
    def __init__(self, title, amount, available):  
        self.title = title  
        self.amount = amount  
        self.available = available  
  
book = Book()  
form = BookForm(obj=book)
```

- Gdy dane formularza są przyjmowane w funkcji widoku, można łatwo przepisać pasujące pola formularza do obiektu klasy z danymi korzystając z funkcji `populate_obj`:

```
book = Book()  
form.populate_obj(book)
```

### Laboratorium

1. Do środowiska wirtualnego doinstaluj Flask-SQLAlchemy.
2. Utwórz definicję klasy korzystającej z Flask-SQLAlchemy odpowiadającej za ewidencję opóźnień pociągów. Do definicji klasy dodaj kolumnę `id`, która będzie kluczem podstawowym, pozostałe pola weź takie, jak w definicji klasy odpowiadającej za formularz.
3. Dodaj do aplikacji z poprzedniego LAB, funkcjonalność pozwalającą w wygodny sposób pobrać dane z formularza i zapisać je jako nowy rekord w bazie danych.

### Sprawdź się!

1. Jakie warunki musi spełniać definicja klasy i definicja formularza, aby w wygodny sposób wypełniać ten formularz danymi z obiektu?
2. Jakie znasz metody na wypełnienie pól formularza określonymi wartościami?

## Propozycja rozwiązania

```
# app.py - only selected fragments:
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SECRET_KEY'] = 'ACompl1cat3dText.'
app.config['SQLALCHEMY_DATABASE_URI']='sqlite:///d:\\Flask\\trains.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS']=False
db = SQLAlchemy(app)

class TrainDelay(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    train_number = db.Column(db.String(30))
    is_delayed = db.Column(db.Boolean)
    delay_minutes = db.Column(db.Integer)
    delay_reason = db.Column(db.String(100))

    def __repr__(self):
        return f"{self.train_number} - {self.delay_minutes}"

db.create_all()

@app.route('/', methods=['POST', 'GET'])
def index():
    form = TrainInfo()
    if form.validate_on_submit():
        train_delay = TrainDelay()
        form.populate_obj(train_delay)
        db.session.add(train_delay)
        db.session.commit()

        return f'''<H1>Hello</H1>
            <ul>
                <li>{form.train_number.label}: {form.train_number.data}</li>
                <li>{form.is_delayed.label}: {form.is_delayed.data}</li>
                <li>{form.delay_minutes.label}: {form.delay_minutes.data}</li>
                <li>{form.delay_reason.label}: {form.delay_reason.data}</li>
            </ul>
            Following record was added to the table in database: {train_delay}
            '''

    return render_template('index.html', form=form)
```

## Dziedziczenie z klasy formularza

### Notatka

- Jeśli do utworzenia są dwa podobne formularze, to budując nowy formularz możesz odziedziczyć z klasy pierwszego formularza:

```
class BookFormEmail(BookForm):  
    email = StringField("e-mail", validators=[Email()])
```

- Jeśli w formularzu potomnym trzeba zrezygnować z jakiegoś pola, to można je usunąć:

```
del form.available
```

- Szablon Jinja, może prezentować pola formularza dynamicznie, zależnie od tego, czy na formularzu jakieś pola są dostępne czy nie:

```
{% if form.email %}  
    {{ form.email.label }}    {{ form.email }} <br/>  
    <ul>  
        {% for error in form.email.errors %}  
            {{ error }}  
        {% endfor %}  
    </ul>  
{% endif %}
```

### Laboratorium

1. W tym LAB utworzysz nowy formularz i tabelę pozwalającą na zapisywanie informacji o opóźnieniu pociągu na określonej stacji (pociąg może powiększać lub zmniejszać opóźnienie)
2. Utwórz nową klasę TrainDelayOnStation dziedziczącą z db.Model, która będzie zawierać te same pola, co TrainDelay i dodatkowo jeszcze 50-cio znakowe pole station, które posłuży do zapisywania na jakiej stacji odnotowano opóźnienie (tutaj nie stosuj dziedziczenia)
3. Utwórz klasę formularza dziedziczącą z TrainInfo i nazwij ją TrainInfoOnStation. Dodaj do niej pole station z walidatorem DataRequired
4. Dodaj route /delay\_on\_station i stowarzyszoną funkcję widoku. Ciało funkcji może być skopiowane z poprzedniej funkcji ze zmianami:
  - a. Pracuj z obiektem typu TrainDelayOnStation i TrainInfoOnStation
  - b. W pliku szablonu spraw, by część formularza związana z polem station była wyświetlana tylko jeśli obiekt form ma właściwość station
  - c. Uwaga: w szablonie w polu <form...> nie podawaj atrybutu action. W takim przypadku mogą z niego korzystać dwie niezależne funkcje widoku, a formularz jest odsyłany do funkcji, która go wygenerowała

### Sprawdź się!

1. Jaki dodatkowy pakiet jest potrzebny do wykonania walidacji adresu email?
2. Czy budowanie dynamicznego kodu, który zachowa się różnie, w zależności od wartości innych zmiennych i parametrów w kodzie, to dobra praktyka czy zła?
3. Jaka funkcjonalność szablonów Jinja pozwala na dynamiczne pokazywanie części formularza?

## Propozycja rozwiązania

```
# app.py - only important fragments:

class TrainDelayOnStation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    train_number = db.Column(db.String(30))
    is_delayed = db.Column(db.Boolean)
    delay_minutes = db.Column(db.Integer)
    delay_reason = db.Column(db.String(100))
    station = db.Column(db.String(50))

    def __repr__(self):
        return f'{self.station} - {self.train_number} - {self.delay_minutes}'

class TrainInfoOnStation(TrainInfo):
    station = StringField('Station', validators=[DataRequired('Enter station name')])

@app.route('/delay_on_station', methods=['POST', 'GET'])
def delay_on_station():
    form = TrainInfoOnStation()
    if form.validate_on_submit():
        train_delay_on_station = TrainDelayOnStation()
        form.populate_obj(train_delay_on_station)
        db.session.add(train_delay_on_station)
        db.session.commit()

        return f'''<H1>Hello</H1>
        <ul>
            <li>{form.train_number.label}: {form.train_number.data}</li>
            <li>{form.is_delayed.label}: {form.is_delayed.data}</li>
            <li>{form.delay_minutes.label}: {form.delay_minutes.data}</li>
            <li>{form.delay_reason.label}: {form.delay_reason.data}</li>
            <li>{form.station.label}: {form.station.data}</li>
        </ul>
        Following record was added to the table in database: {train_delay_on_station}'''

    return render_template('index.html', form=form)

# index.html

{% from "macros.html" import show_validation_results %}

<form method="POST" >>
    {{ form.csrf_token }}
    {% if form.station %}
        {{ form.station.label }} {{ form.station }} <br/>
        {{ show_validation_results(form.station) }}
    {% endif %}
    {{ form.train_number.label }} {{ form.train_number }} <br/>
    {{ show_validation_results(form.train_number) }}
    {{ form.is_delayed.label }} {{ form.is_delayed }} <br/>
    {{ form.delay_minutes.label }} {{ form.delay_minutes }} <br/>
    {{ show_validation_results(form.delay_minutes) }}
    {{ form.delay_reason.label }} {{ form.delay_reason }} <br/>

    <input type="submit" value="GO!">
</form>
```

## Korzystanie z kontrolek WTForms

### Notatka

- WTForms udostępnia wiele popularnych kontrolek HTML, które dzięki modułowi Flask-WTF można wykorzystywać podczas budowania formularzy webowych
- Aby skorzystać z wybranych kontrolek, należy je zaimportować:

```
from flask_wtf.file import FileField, FileRequired, FileAllowed
```

- W klasie definiującej formularz można następnie wykorzystać kontrolkę i walidatory:

```
cover = FileField("Book cover", validators=[FileRequired(),  
                                           FileAllowed(['jpg', 'png'], "Sorry, only png and jpg")])
```

- Przyjmując np. Plik wysłany przez użytkownika, zazwyczaj zapisuje się go na serwerze:

```
f = form.cover.data  
filename = secure_filename(f.filename)  
f.save(os.path.join(app.root_path, 'static', 'covers', filename))
```

- Aby taki plik wyświetlić można skonstruować znacznik img i umieścić go na stronie:

```

```

- Przy formularzu wysyłającym plik istotne jest, aby zdefiniować atrybut enctype:

```
<form method="POST" action="{{ url_for('index') }}" enctype="multipart/form-data" >
```

### Laboratorium

1. W formularzu dotyczącym opóźnień pociągów zamień listę rozwijaną pozwalającą na definiowanie przyczyny opóźnienia, na pole typu radio.
  - a. Zaimportuj odpowiednią klasę z WTForms
  - b. Zmień definicję pola w klasie formularza, dodając wartość domyślną wskazującą na 'None'
2. W formularzu Jinja, pole delay\_reason możesz zostawić tak jak jest lub jeśli masz ochotę wyświetlać w sposób opisany w dokumentacji:

```
{% for option in form.delay_reason %}  
  <tr>  
    <td>{{ option }}</td>  
    <td>{{ option.label }}</td>  
  </tr>  
{% endfor %}
```

### Sprawdź się!

1. Jakie typy danych można przyjmować w aplikacji z wykorzystaniem WTForms?
2. Jakie walidatory mogą być używane z kontrolką do wysyłania plików?

## Propozycja rozwiązania

# app.py - only changed fragments:

```
from wtforms import StringField, IntegerField, BooleanField, SelectField, RadioField
```

```
class TrainInfo(FlaskForm):
```

```
    def start_with2letters(form, field):
        if not (len(field.data) > 2 and field.data[0:2].isalpha()):
            raise ValidationError('Train number must start with 2 letters')

    train_number = StringField('Train number',
                               validators=[DataRequired('Enter train number'), start_with2letters])
    is_delayed = BooleanField('Is delayed')
    delay_minutes = IntegerField('Delay in minutes',
                                 validators=[DataRequired('Enter delay'),
                                             NumberRange(min=0, message='The delay must be a number >= 0')])
    #delay_reason = SelectField('Delay reason', choices=['None', 'weather', 'Failure', 'Other'])
    delay_reason = RadioField('Delay reason', choices=['None', 'weather', 'Failure', 'Other'],
                              default='None')
```

# index.html - full content

```
{% from "macros.html" import show_validation_results %}
```

```
<form method="POST" %}>
```

```
    {{ form.csrf_token }}
    {% if form.station %}
        {{ form.station.label }} {{ form.station }} <br/>
        {{ show_validation_results(form.station) }}
    {% endif %}
    {{ form.train_number.label }} {{ form.train_number }} <br/>
    {{ show_validation_results(form.train_number) }}
    {{ form.is_delayed.label }} {{ form.is_delayed }} <br/>
    {{ form.delay_minutes.label }} {{ form.delay_minutes }} <br/>
    {{ show_validation_results(form.delay_minutes) }}
    <!--{{ form.delay_reason.label }} {{ form.delay_reason }} <br/> -->
    {% for option in form.delay_reason %}
        <tr>
            <td>{{ option }}</td>
            <td>{{ option.label }}</td>
        </tr>
    {% endfor %}</br>
```

```
    <input type="submit" value="GO!">
</form>
```



## Kontrolki HTML5

### Notatka

- Kontrolki HTML5 pochodzące w WTForms są nowszą i przyjaźniejszą dla użytkownika formą wprowadzania danych
- Tworząc interfejs aplikacji, pamiętaj, że nazwy klas w WTForms dla HTML i HTML5 są takie same – zdecyduj się, które chcesz wykorzystywać i zaimportuj właściwe moduły, np. zamiast

```
from wtforms import DateField
```

użyj

```
from wtforms.fields.html5 import DateField
```

- Kontrolki mogą też mieć różne zestawy akceptowanych parametrów np. zamiast:

```
DateField('offer date', format='%Y,%m-%d')
```

napisz

```
DateField('offer date')
```

### Laboratorium

1. W tym LAB utworzysz nowy formularz i tabelę pozwalającą na zapisywanie informacji o opóźnieniu pociągu określonego dnia na określonej stacji (czyli część czynności jest podobna do jednego z wcześniejszych LAB, gdzie dodawaliśmy pole station).
2. Utwórz nową klasę TrainDelayOnStationDay dziedziczącą z db.Model, która będzie zawierać te same pola, co TrainDelay i dodatkowo jeszcze 50-cio znakowe pole station i pole day typu Date. (nie stosuj dziedziczenia)
3. Do klasy formularza TrainInfoStation dodaj pole day
4. Zmień route skojarzone z j route delay\_on\_station. Funkcja ta ma pracować z klasą TrainDelayOnStationDay
5. W pliku szablonu spraw, by część formularza związana z polem day była wyświetlana tylko jeśli obiekt form ma właściwość day

### Sprawdź się!

1. Jak sądzisz (o tym nie było mowy na lekcji), z jakiego powodu można by się decydować na wykorzystywanie jednak tylko kontrolki HTML zamiast HTML5?
2. Jakiego rodzaju kontrolki HTML5 są dostępne?

## Propozycja rozwiązania

#app.py - only fragments:

```
class TrainDelayOnStationDay(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    train_number = db.Column(db.String(30))
    is_delayed = db.Column(db.Boolean)
    delay_minutes = db.Column(db.Integer)
    delay_reason = db.Column(db.String(100))
    station = db.Column(db.String(50))
    day = db.Column(db.Date)

    def __repr__(self):
        return f"{self.station} - {self.train_number} - {self.delay_minutes}"

class TrainInfo(FlaskForm):
    def start_with2letters(form, field):
        if not (len(field.data) > 2 and field.data[0:2].isalpha()):
            raise ValidationError('Train number must start with 2 letters')

    train_number = StringField('Train number',
                               validators=[DataRequired('Enter train number'), start_with2letters])
    is_delayed = BooleanField('Is delayed')
    delay_minutes = IntegerField('Delay in minutes', validators=[DataRequired('Enter delay'),
                                                                NumberRange(min=0, message='The delay must be a number >= 0')])
    delay_reason = RadioField('Delay reason', choices=['None', 'Weather', 'Failure', 'Other'],
                              default='None')

class TrainInfoOnStation(TrainInfo):
    station = StringField('Station', validators=[DataRequired('Enter station name')])
    day = DateField('Day', validators=[DataRequired('Enter date')], default=date.today())

@app.route('/delay_on_station', methods=['POST', 'GET'])
def delay_on_station():
    form = TrainInfoOnStation()
    if form.validate_on_submit():
        train_delay_on_station_day = TrainDelayOnStationDay()
        form.populate_obj(train_delay_on_station_day)
        db.session.add(train_delay_on_station_day)
        db.session.commit()

        return f'''<H1>Hello</H1>
        <ul>
            <li>{form.train_number.label}: {form.train_number.data}</li>
            <li>{form.is_delayed.label}: {form.is_delayed.data}</li>
            <li>{form.delay_minutes.label}: {form.delay_minutes.data}</li>
            <li>{form.delay_reason.label}: {form.delay_reason.data}</li>
            <li>{form.station.label}: {form.station.data}</li>
            <li>{form.day.label}: {form.day.data}</li>
        </ul>
        Following record was added to the db: {train_delay_on_station_day}
        '''

    return render_template('index.html', form=form)
```

#index.html - only added part:

```
{% if form.day %}
    {{ form.day.label }} {{ form.day }} <br/>
    {{ show_validation_results(form.day) }}
{% endif %}
```

## Flask Login – testowa aplikacja

### Notatka

- Niekiedy mechanizm logowania do aplikacji jest dodawany dopiero po stworzeniu szkieletu funkcji. Taki scenariusz rozważamy w najbliższym module

### Laboratorium

- Do aplikacji rejestrującej opóźnienia pociągów dodaj route do funkcji login i logout. Dodaj również formularz logowania. Dodaj też klasę User pozwalającą na przechowywanie w bazie danych informacji o użytkowniku oraz klasę LoginForm odpowiadającą za klasę widoku dla route login (skorzystaj z kodu z rozwiązań)

### Sprawdź się!

- Jeśli formularz nie ma atrybutu action, to gdzie zostaną wysłane dane wprowadzone przez użytkownika?

### Propozycja rozwiązania

```
# app.py – only changed/added fragments

from flask import Flask, render_template, url_for, request, redirect
from flask_wtf import FlaskForm
from wtforms import BooleanField, StringField, SelectField, RadioField, PasswordField
from wtforms.fields.html5 import IntegerField, DateField
from wtforms.validators import DataRequired, NumberRange, ValidationError
from flask_sqlalchemy import SQLAlchemy
from datetime import date
import hashlib
import binascii

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), unique=True)
    password = db.Column(db.String(100))
    first_name = db.Column(db.String(50))
    last_name = db.Column(db.String(50))

    def __repr__(self):
        return ('User: {}, {}'.format(self.name))

    def get_hashed_password(password):
        """Hash a password for storing."""
        # the value generated using os.urandom(60)
        os_urandom_static = b"ID_\x12p:\x8d\xe7&\xcb\xfb=H1\xc1\x16\xac\xe5B\x7d\x6j\xe3i\x11\xbe\xaa\x05\xccc\xc2\xe8K\xcf\xfb\xac\x9bFy(\xfbn.\xe9\xcd\xdd'\xdf~vm\xae\xf2\x93wD\x04"
        salt = hashlib.sha256(os_urandom_static).hexdigest().encode('ascii')
        pwdbash = hashlib.pbkdf2_hmac('sha512', password.encode('utf-8'), salt, 100000)
        pwdbash = binascii.hexlify(pwdbash)
        return (salt + pwdbash).decode('ascii')

    def verify_password(stored_password_hash, provided_password):
        """Verify a stored password against one provided by user"""
        salt = stored_password_hash[:64]
        stored_password = stored_password_hash[64:]
        pwdbash = hashlib.pbkdf2_hmac('sha512', provided_password.encode('utf-8'), salt.encode('ascii'), 100000)
        pwdbash = binascii.hexlify(pwdbash).decode('ascii')
        return pwdbash == stored_password

class LoginForm(FlaskForm):
    name = StringField('User name')
    password = PasswordField('Password')
    remember = BooleanField('Remember me')
```

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    return render_template('login.html', form=form)

@app.route('/logout')
def logout():
    return '<h1>You are logged out</h1>'

# login.html - full
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wquIj61tLrc4wSX0szH/Ev+nYRRuWlo1flfl"
crossorigin="anonymous">
    <title>Logon</title>
  </head>
  <body>

    {% for message in get_flashed_messages() %}
      <div class="alert alert-warning alert-dismissible fade show" role="alert">
        <strong>{{ message }}</strong>
        <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>
      </div>
    {% endfor %}

    <!--Grid row-->
    <div class="row d-flex justify-content-center">
      <!--Grid column-->
      <div class="col-md-4">

        <form method="POST">
          {{ form.csrf_token }}
          <div class="form-row align-items-center">
            <div class="col-auto my-1">
              <label class="sr-only" for="inlineFormInputName">{{ form.name.label
}}</label>
              {{ form.name(class_="form-control") }}
            </div>
            <div class="col-auto my-1">
              <label class="sr-only" for="inlineFormInputGroupUsername">{{
form.password.label }}</label>
              {{ form.password(class_="form-control") }}
            </div>
            <div class="col-auto my-1">
              <div class="form-check">
                {{ form.remember(class_="form-check-input") }}
                <label class="form-check-label" for="autoSizingCheck2">
                  {{ form.remember.label }}
                </label>
              </div>
            </div>
            <div class="col-auto my-1">
              <button type="submit" class="btn btn-primary">Login</button>
            </div>
          </div>
        </form>
      </div>
    <!--Grid column-->
  </div>
  <!--Grid row-->

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/js/bootstrap.bundle.min.js" integrity="sha384-
b5kHyXgcpbJ30/tY9U17kgkf1S0CWuKcCD38l8YkeH8z8QjE0Gmw1gyU5S9FonJ0"
crossorigin="anonymous"></script>

  </body>
</html>

```

## Flask-Login instalacja i wykorzystanie

### Notatka

- Instalacja Flask-Login może być wykonana komendą

```
pip install flask-login
```

- Najczęściej importowane obiekty z flask\_login to:
  - LoginManager – odpowiada za funkcjonalność modułu
  - UserMixin – klasa rodzicielska dla klasy z informacjami o użytkownikach
  - login\_user, logout\_user – funkcje do logowania i wylogowania użytkownika
  - login\_required – dekorator dla funkcji, do których dostęp ma być ograniczany
  - current\_user – obiekt przechowujący dane bieżącego użytkownika
- LoginManager wymaga stworzenia instancji:

```
login_manager = LoginManager(app)
```

- Klasa przechowująca informacje o użytkownikach powinna dziedziczyć z UserMixin:

```
class User(db.Model, UserMixin):
```

- Funkcja oznaczona dekoratorem user\_loader powinna ładować informacje o użytkowniku w oparciu o id użytkownika:

```
@login_manager.user_loader
def load_user(id):
    return User.query.filter(User.id == id).first()
```

- W aplikacji, po przetestowaniu użytkownika loguje się go funkcją login\_user(user\_obj). Kiedy użytkownik wybiera polecenie „Wyloguj”, wystarczy wywołać logout\_user()
- Funkcje z ograniczonym dostępem dekoruje się przez @login\_required

### Laboratorium

1. Dodaj do aplikacji rejestrującej opóźnienia pociągów możliwość zalogowania się:
  - a. Zainstaluj Flask-Login, zimportuj przydatne obiekty z flask\_login
  - b. Oprogramuj funkcję init, która stworzy rekord dla użytkownika admin
  - c. Oprogramuj funkcje login (po sprawdzeniu hasła zaloguj użytkownika) i logout
  - d. Zabezpiecz formularz dodawania rekordów, tak aby działał tylko dla zalogowanych użytkowników i przetestuj działanie funkcji będąc zalogowanym i wylogowanym

### Sprawdź się!

1. Do czego służy UserMixin?
2. Jakie warunki musi spełniać klasa User przechowująca informacje o użytkownikach?
3. Jakie zadanie spełnia funkcja oznaczona dekoratorem user\_loader?

## Propozycja rozwiązania

```
#app.py - only fragments

from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required,
current_user

login_manager = LoginManager(app)

class User(db.Model, UserMixin):
    # [...]

@login_manager.user_loader
def load_user(id):
    return User.query.filter(User.id == id).first()

@app.route('/init')
def init():
    db.create_all()

    admin = User.query.filter(User.name=='admin').first()
    if admin == None:
        admin = User(id=1, name='admin', password=User.get_hashed_password('Passw0rd'),
                     first_name='King', last_name='Kong')
        db.session.add(admin)
        db.session.commit()

    return '<h1>Initial configuration done!</h1>'

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

    if form.validate_on_submit():
        user = User.query.filter(User.name == form.name.data).first()
        if user != None and User.verify_password(user.password, form.password.data):
            login_user(user)

            next = request.args.get('next')
            if next and is_safe_url(next):
                return redirect(next)
            else:
                return '<h1>You are authenticated!</h1>'

    return render_template('login.html', form=form)

@app.route('/logout')
def logout():
    logout_user()
    return '<h1>You are logged out</h1>'

@app.route('/delay_on_station', methods=['POST', 'GET'])
@login_required
def delay_on_station():
    # [...]
```

## Przekierowanie użytkownika do strony logowania

### Notatka

- Dobrym zwyczajem jest przekierowanie niezalogowanego użytkownika do strony logowania, jeśli próbuje sięgnąć do strony dostępnej tylko dla zalogowanych użytkowników
- Ustawienie poniższych parametrów LoginManagera, spowoduje przekierowanie użytkownika do funkcji 'login' jeśli dochodzi do połączenia do zastrzeżonej strony. Dodatkowo wyświetlany jest komunikat z parametru login\_message

```
login_manager.login_view = 'login'  
login_manager.login_message = 'First, please log in using this form:'
```

- Po zalogowaniu użytkownik może być przeniesiony na stronę wskazywaną przez parametr query string o nazwie next. Podczas przekierowywania użytkownika należy zwracać uwagę czy next wskazuje na bezpieczny adres:

```
next = request.args.get('next')  
if next and is_safe_url(next):  
    return redirect(next)
```

### Laboratorium

1. Dodaj omówioną w tej lekcji funkcjonalność do swojej aplikacji:
  - a. Kiedy użytkownik jest niezalogowany i próbuje zarejestrować opóźnienie pociągu, to
  - b. Przenieś go na stronę logowania, a potem
  - c. Odeślij z powrotem na stronę z formularzem do opisu opóźnień pociągów
2. Zabezpiecz aplikację przed przekierowaniem na adres zewnętrzny

### Sprawdź się!

1. Jakie niebezpieczeństwa niesie za sobą niekontrolowane przekierowywanie użytkownika w oparciu o parametry przekazywane w Query String?
2. Jakie jest znaczenie login\_view i login\_message?

## Propozycja rozwiązania

```
from urllib.parse import urlparse, urljoin

login_manager.login_view = 'login'
login_manager.login_message = 'First, please log in using this form:'

def is_safe_url(target):
    ref_url = urlparse(request.host_url)
    test_url = urljoin(request.host_url, target)
    return test_url.scheme in ('http', 'https') and \
        ref_url.netloc == test_url.netloc

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter(User.name == form.name.data).first()
        if user != None and User.verify_password(user.password, form.password.data):
            login_user(user)

            next = request.args.get('next')
            if next and is_safe_url(next):
                return redirect(next)
            else:
                return '<h1>You are authenticated!</h1>'
    return render_template('login.html', form=form)
```



## Wymuszenie ponownego logowania

### Notatka

- Sesja i informacja o zalogowanym użytkowniku jest zapisywana w postaci cookie
- Zazwyczaj po zamknięciu przeglądarki ciasteczka sesji czyszczą się
- Jeśli sesja ma przetrwać zamknięcie przeglądarki, to logując użytkownika użyj `remember=True`:

```
login_user(user, remember=True)
```

- Jeśli klasa `User` nie posiada właściwości `id`, to należy zdefiniować funkcję `get_id` w klasie `User`, tak aby funkcja ta zwracała obiekt użytkownika mając na wejściu wartość klucza podstawowego klasy/tabeli `user`:

```
def get_id(self):  
    return self.name
```

- Aby przed uruchomieniem funkcji wymusić odświeżenie logowania użytkownika:
  - Zamiportuj dekorator `fresh_login_required`
  - Udekoruj funkcję dekoratorem `@fresh_login_required`
  - Aby automatycznie przenieść użytkownika na stronę zalogowania i wyświetlić odpowiedni komunikat zmień wartość zmiennych:

```
login_manager.refresh_view = 'login'  
login_manager.needs_refresh_message = 'You need to log on again'
```

UWAGA: Działanie omówionych tu funkcji może być różne na różnych przeglądarkach. Jeśli więc Twoje testy nie wychodzą, to spróbuj zmienić przeglądarkę.

### Laboratorium

1. Do programu rejestrującego opóźnienia pociągów dodaj opcję „Remember me”
2. Przetestuj działanie aplikacji, po zamknięciu przeglądarki i ponownym połączeniu do aplikacji.  
Jeśli widzisz pewne problemy, spróbuj również innej przeglądarki

### Sprawdź się!

1. Gdzie jest przechowywana informacja o tym, że użytkownik jest zalogowany?
2. Co powoduje opcja `remember` w poleceniu `login_user`?
3. Jakie są zalety i wady korzystania z modułów?

## Propozycja rozwiązania

```
# app.py

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

    if form.validate_on_submit():
        user = User.query.filter(User.name == form.name.data).first()
        if user != None and User.verify_password(user.password, form.password.data):
            login_user(user, remember=form.remember.data)

            next = request.args.get('next')
            if next and is_safe_url(next):
                return redirect(next)
            else:
                return '<h1>You are authenticated!</h1>'

    return render_template('login.html', form=form)
```

## Projekt – 04 – Aplikacja CRUD – drugie podejście

W poprzednim projekcie tworzyłeś aplikację CRUD (o ile pisałeś). Obecnie jednak, znając SQLAlchemy, WTF, Flask-Login, można by tą aplikację napisać zupełnie inaczej!

- Czy tworzenie zapytań SQL w postaci „sklejanego” tekstu było wygodne? Chyba nie... SQLAlchemy może ten problem rozwiązać
- Czy budowanie formularzy w czystym HTML to najwygodniejszy sposób pracy? Chyba nie... WTForms pozwoli budować formularze łatwiej!
- Czy przetwarzanie w programie niesprawdzonych danych wprowadzanych przez użytkownika bezpośrednio w przeglądarce jest na pewno bezpieczne? Chyba nie... Dlatego dodaj do pól walidatory
- Czy samodzielne wbudowywanie logowania i uprawnień w aplikacji jest eleganckie? Chyba nie... Dlatego rozważ zamianę tej funkcjonalności na Flask-Login.

## Co dalej?

### Notatka

- Korzystaj z dokumentacji – żaden kurs ani artykuł lepiej nie opisze funkcjonalności niż dobra (nudna) dokumentacja
- Korzystaj z blogów
  - Patrz z czego korzystają inni programiści
  - Patrz gdzie się uczą – jakie artykuły, książki, kursy polecają
  - Patrz nad jakimi projektami pracują
  - Przeglądaj ich repozytoria na Git-Hub

### Laboratorium

1. Poświęć chwilę na wyszukanie i dodanie do ulubionych
  - a. Kilku stron z dokumentacją prezentowanych na kursie modułów
  - b. Kilku artykułów recenzujących moduły
  - c. Kilku programistów, którzy dzielą się tym, jak uczą się Flaska
2. Rozważ samodzielne opublikowanie podobnych informacji na LinkedIn, Facebooku itp.
3. Mówili Ci w szkole, że pisanie ściąg jest złe? Kłamali. Dosyć popularne są tzw. Cheat Sheets zawierające krótkie informacje o języku i jego konstrukcjach. Niegłupim pomysłem jest ściągnięcie odpowiednich ściąg, wydrukowanie i zawieszenie nad biurkiem. Serwis <https://cheatography.com/> pozwala nawet tworzyć samodzielnie takie ściągi i udostępniać je innym. Rozważ zbudowanie takiej ściągi. Pochwal się swoim dziełem w komentarzach do tego kursu – szybko zdobędziesz fanów!

### Sprawdź się!

1. Do czego służą wymienione poniżej moduły?
  - a. Flask Admin
  - b. Flask User
  - c. Flask Security
  - d. Flask Babel
  - e. Flask Uploads
  - f. Flask Migrate
  - g. Flask Mail
2. Czy korzystanie z modułów to ZAWSZE najlepsza możliwa praktyka?

## Dodatek: Typowe błędy

### Flask is not recognized

flask : The term 'flask' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.

Nie aktywowałeś środowiska wirtualnego lub w środowisku nie zainstalowałeś Flask'a?

### Could not locate a Flask application

Error: Could not locate a Flask application. You did not provide the "FLASK\_APP" environment variable, and a "wsgi.py" or "app.py" module was not found in the current directory.

Główny program aplikacji powinien mieć nazwę app.py i musi się znajdować w katalogu bieżącym albo musi być ustawiona zmienna środowiskowa FLASK\_APP wskazująca na nazwę pliku programu

### Strona się nie odświeża

Po zmianie kodu, przeładowanie strony nadal pokazuje starą zawartość

Upewnij się, że zapisałeś kod 😊

Upewnij się, że jesteś w trybie debug (ustaw zmienną środowiskową FLASK\_DEBUG na 1 przed uruchomieniem Flask'a)

Zatrzymaj Flask i uruchom go ponownie (zmiana kodu nie została poprawnie wykryta przez Flask)

### The requested URL was not found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

Odwołujesz się do nieistniejącej route. Sprawdź czy nie ma literówek oraz czy dynamicznie przekazywane parametry są rzeczywiście zdefiniowane i czy mają właściwy typ. Być może route definiuje parametr określonego typu, a przeglądarka przesyła adres z wartością, która do tego typu nie pasuje

### TypeError unexpected keyword

TypeError: cantor() got an unexpected keyword argument 'currency1'

Nazwy parametrów w definicji dynamicznej route różnią się od parametrów w przypisanej do tej route funkcji.

### TypeError missing required positional arguments

TypeError: cantor() missing 2 required positional arguments: 'amount' and 'currency'

Dokładnie sprawdź, czy nazwy parametrów w definicji route i nazwy parametrów w funkcji przypisanej do tej route są takie same. Zrezygnuj, albo przemyśl wykorzystanie argumentów `*args`, `**kwargs`

### Method not allowed

Method Not Allowed. The method is not allowed for the requested URL.

W dekoratorze `app.route` zapomniałeś dodać metody przesyłania żądania. Dodaj do dekoratora `@app.route` argument `methods`)

### werkzeug.routing.BuildError, could not build URL

werkzeug.routing.BuildError: Could not build url for endpoint '/exchange'. Did you mean 'exchange' instead?

Prawdopodobnie korzystając z funkcji `url_for` odwołujesz się do nazwy route (`/exchange`), zamiast do nazwy funkcji, która tą route obsługuje (`exchange`). Pamiętaj: `url_for` wymaga przekazywania nazwy funkcji, a nie route!

### Runtime error, the session is unavailable

RuntimeError: The session is unavailable because no secret key was set. Set the secret\_key on the application to something unique and secret.

Korzystasz z funkcji, która wymaga zdefiniowania tzw. Secret key. Taką funkcjonalnością może być funkcja flask() lub korzystanie z cookies. Dodaj do kodu instrukcję definiującą SECRET\_KEY:

```
app.config['SECRET_KEY'] = 'SomethingHardToGuess'
```

Spróbuj też!

