

# Sprawozdanie z kursu Bazy Danych 2

Tymoteusz Frankiewicz 241255

Filip Mykieta 229900

Jan Wawruszczak 241344

24 stycznia 2020

## Etap 1 - wymagania

### Proponowany temat:

**WhatsDown** - ewidencja cmentarna

### Opis aplikacji

Aplikacja obsługuje ewidencję cmentarną i ma formę wyszukiwarki. Zakłada trzy role: gość, użytkownik (zakład pogrzebowy), administrator. W bazie danych, z której korzysta aplikacja, gromadzone są dane o zmarłych (personalia, data zgonu, strój trumienny) oraz szczegóły ich pogrzebów (takie jak cmentarz, kapelan, świątynia).

### Wymagania funkcjonalne

1. **Gość**, którym jest każdy internauta odwiedzający stronę:

- (a) może wyszukiwać dane zmarłego, podając na przykład jego nazwisko; dane zmarłego są automatycznie uzupełnione o szczegóły jego pogrzebu,
- (b) może filtrować niejednoznaczne wyniki wyszukiwania, na przykład wielu zmarłych o tym samym nazwisku,
- (c) może rozszerzyć dowolne wyszukiwanie o podobne wyszukiwania, na przykład szukać zmarłych pochowanych w tym samym stroju lub tego samego dnia co konkretny znaleziony zmarły,
- (d) może sprawdzić udostępnione statystyki na tematy związane z ewidencją cmentarną, na przykład średni koszt pogrzebu.

2. **Użytkownik**, którym jest każdy zakład pogrzebowy korzystający z aplikacji:

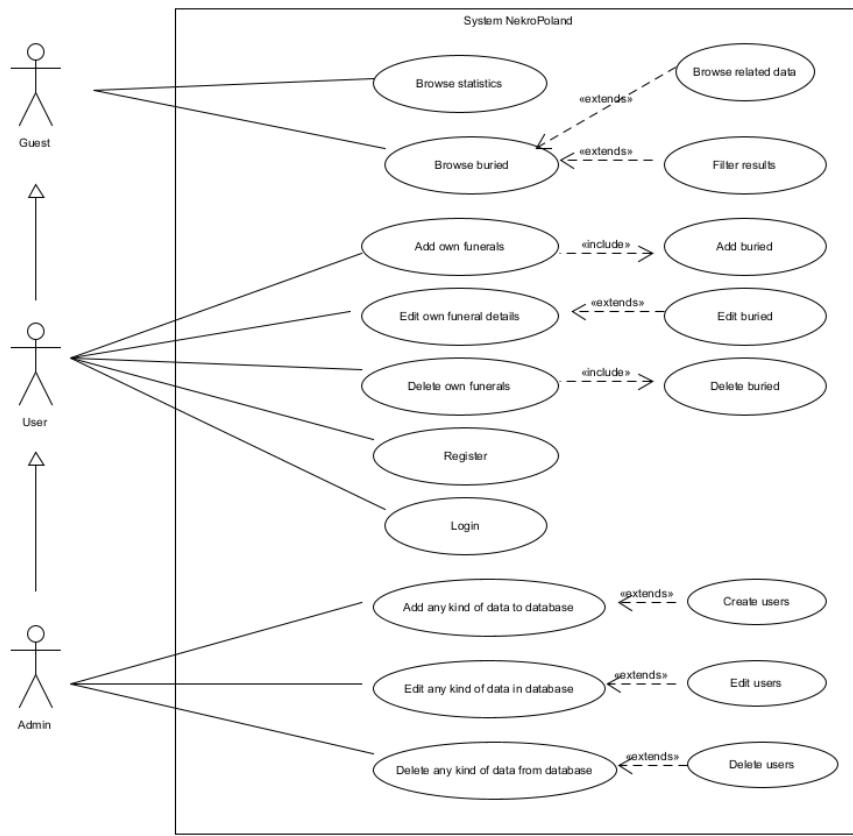
- (a) rejestruje się na stronie przez formularz,
- (b) ma do dyspozycji wszystkie funkcjonalności udostępnione gościowi,
- (c) może logować się na swoje konto,
- (d) może w panelu użytkownika edytować swoje dane,
- (e) może w panelu użytkownika dodawać informacje o pogrzebach, które sam zorganizował,
- (f) może w panelu użytkownika dodawać dane zmarłych, których pochował w zorganizowanych przez siebie pogrzebach,
- (g) może w panelu użytkownika edytować informacje o pogrzebach, które sam zorganizował,

- (h) może w panelu użytkownika edytować dane zmarłych, których pochował w zorganizowanych przez siebie pogrzebach,
- (i) może w panelu użytkownika usuwać informacje o pogrzebach, które sam zorganizował,
- (j) może w panelu użytkownika usuwać dane zmarłych, których pochował w zorganizowanych przez siebie pogrzebach,

3. **Administrator**, którym jest każdy zarządcą całej aplikacji:

- (a) ma do dyspozycji połączone wszystkie funkcjonalności udostępnione każdemu z pojedynczych użytkowników, w tym także rejestruje się na stronie i może logować się na swoje konto,
- (b) może w panelu administratora dodawać dowolne dane w bazie, takie jak szczegóły cmentarzy, strojów czy nagrobków,
- (c) może w panelu administratora utworzyć nowego użytkownika poprzez odpowiedni wpis do bazy,
- (d) może w panelu administratora edytować dowolne dane w bazie, takie jak szczegóły cmentarzy, strojów czy nagrobków,
- (e) może w panelu administratora edytować dane istniejącego użytkownika,
- (f) może w panelu administratora usuwać dowolne dane w bazie, takie jak szczegóły cmentarzy, strojów czy nagrobków,
- (g) może w panelu administratora usunąć istniejącego użytkownika poprzez odpowiedni wypis z bazy.

## Diagram przypadków użycia



Rysunek 1: Diagram przypadków użycia

## Wymagania niefunkcjonalne

### 1. Ogólne:

- forma: aplikacja webowa,
- preferowany system jako środowisko działania aplikacji: Linux,
- aplikacja obsługuje kodowanie polskich znaków, czytelnych na różnych platformach,
- aplikacja powinna korzystać głównie z technologii open source.

### 2. Wymagania:

- nowoczesna przeglądarka np. Google Chrome lub Mozilla Firefox,
- nieblokowanie w przeglądarce skryptów JS

**3. Bezpieczeństwo:**

- (a) różne poziomy dostępu do aplikacji, zatem niemożliwe będzie dostanie się do strony admina przez gościa lub użytkownika,
- (b) hashowanie haseł,
- (c) wykorzystanie tajnego, unikatowego kodu, umożliwiającego rejestrację administratorską,
- (d) limit czasu sesji,
- (e) zabezpieczenie przed atakiem typu SQL Injection,
- (f) zabezpieczenie przed atakiem typu Cross Site Scripting,
- (g) zabezpieczenie przed atakiem typu Cross-Site Request Forgery,
- (h) niepozostawienie hasła do bazy danych w publicznym repozytorium

## Etap 2

### 1. Architektura

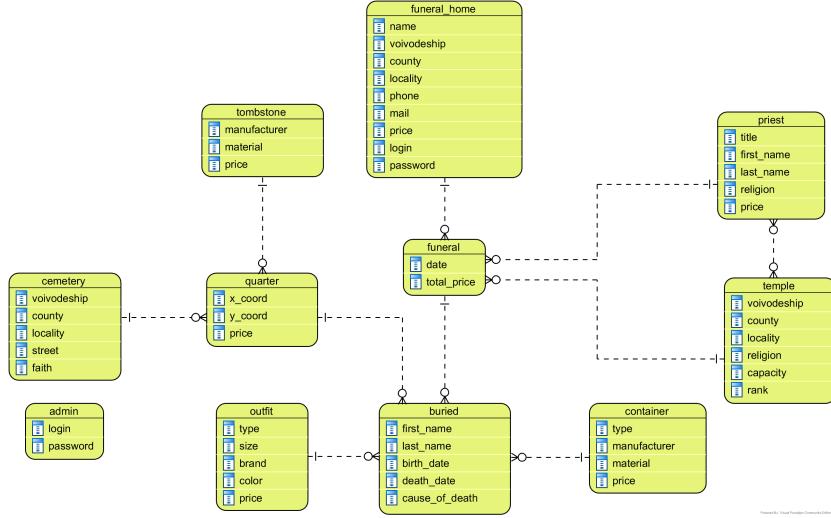
Aplikacja będzie napisana zgodnie ze wzorcem MVT (Model View Template). Oznacza to podział aplikacji na następujące części:

- (a) Model - jako że używamy mapowania obiektowo-relacyjnego (ORM), modelem danych będą klasy, których pola odpowiadają kolumnom tabel w bazie danych. Pole klasy zawiera zarówno informacje o typie kolumny w bazie danych, jak i o typie relacji z inną tabelą,
- (b) View - jest to warstwa logiki (operacje na bazie danych, obliczenia, obsługa danych otrzymanych przez formularz HTML, etc) aplikacji. Każdy widok reprezentuje to, co się dzieje pod określonym adresem URL. Widoki zwracają dane, które mają zostać wyrenderowane w szablonie HTML,
- (c) Template - Plik HTML ze wstawkami w postaci danych zwróconych przez widok. Użycie szablonów likwiduje problem redundancji kodu HTML (np. możliwe wydzielenie stopki do osobnego pliku i ładowanie jej przez proste polecenie include). Możliwe jest stylizowanie HTML przez CSS.

### 2. Technologie

- (a) Python3 - interpretowany język programowania. Preferowana jest wersja 3.6.7 i nowsze,
- (b) Flask - microframework Pythona, służący do pisania aplikacji webowych. Sam framework dostarcza minimalną funkcjonalność, którą można rozszerzyć przez instalację odpowiednich dependencies (np. służących do bezpiecznej obsługi formularzy, panelu administratora czy do mapowania obiektowo-relacyjnego),
- (c) MySQL - otwartoźródłowa relacyjna baza danych,
- (d) SQLAlchemy - ORM dedykowany do Flaska. Korzystanie z ORM nie dość, że ułatwia korzystanie z bazy danych (np. nie ma konieczności zmiany zapytania SQL przy zmianie struktury tabeli w bazie danych), to czyni pracę w grupie znacznie wygodniejszą. Każda zmiana bazy danych zapisywana jest w migracji - pliku Pythona. Wtedy druga osoba korzystająca z bazy danych może ją zaaplikować jedną komendą, zamiast zmieniać strukturę bazy danych instrukcjami DDL,
- (e) Jinja2 - szablony HTML umożliwiające prezentację danych zwróconych przez widok,
- (f) venv - wirtualne środowisko Pythona pozwalające na odseparowanie dependencies będących za-instalowanych globalnie od tych potrzebnych w projekcie,
- (g) git i zdalne repozytorium GitHub - narzędzia konieczne do pracy w zespole. Pozwalają na zapisywanie postępu zmian i jednoczesną pracę nad różnymi funkcjonalnościami,
- (h) inne dependencies, które w zależności od napotkanych wyzwań w implementacji mogą okazać się pomocne.

### 3. Diagram encji



Rysunek 2: Diagram encji

### 4. Transakcje

Każdy zakład pogrzebowy prowadzi ze swojego konta ewidencję pogrzebów, które zorganizował. Transakcja pogrzebowa polega w pierwszej kolejności na dodaniu rekordu do tabeli *funeral*. Pola rekordu uzupełnia sam zakład (wybierając wartości z *combo boxów*), wyjątkiem jest tu przypisanie zakładu organizującego pogrzeb do wprowadzanego rekordu, które następuje automatycznie - poprzez pobranie danych z sesji logowania. Drugim wyjątkiem jest pole *total\_price* - tutaj przy wprowadzeniu nowego pogrzebu automatycznie wstawiamy wartość pola *price* pobraną z powiązanego z pogrzebem rekordu z tabeli *priest*. Żadne pole w tabeli *funeral* nie może przyjąć wartości *NULL*.

Kolejnym etapem transakcji pogrzebowej jest dodanie pochowanych do tabeli *buried*. Wartość *NULL* mogą przyjmować pola *birth\_date*, *death\_date* oraz *cause\_of\_death*. W szczególności oznacza to, że pola *first\_name* oraz *last\_name* nie mogą przyjmować wartości *NULL*. Taka kolejność kroków (pogrzeb → pochowany) wynika z tego, że nie dopuszcza się istnienia w bazie pochowanego bez przypisanego pogrzebu. Poza tym w miarę dodawania pochowanych (dopuszczamy kilku chowanych przy okazji jednego pogrzebu) pole *total\_price* pogrzebu jest powiększane o kolejne wartości pochodzące z odpowiednich pól *price* z tabel będących w relacji z pochowanym. Kwaterna, w której jest umieszczany pochowany, musi być wolna - brak nagrobka na kwarterze (klucz obcy o wartości *NULL*) oznacza, że można tam jeszcze kogoś umieścić. Po pochowaniu w kwarterze jednej czy też kilku osób przypisanie do tego miejsca nagrobka kończy transakcję dodawania pogrzebu i zmarłych.

Zakład edytujący (własne) rekordy w tabelach *funeral* lub *buried* czy też własne dane w tabeli *funeral\_homes* ma do tego pełne prawo (może to być na przykład korekta wcześniejszej pomyłki). System nie dopuści oczywiście do tego, by jako klucza obcego użyć indeksu nieistniejącego w skojarzonej tabeli.

Przyjmujemy zasadę, że zakład usuwający własny rekord z tabeli *funeral* tym samym usuwa wszystkich powiązanych z danym pogrzebem pochowanych z tabeli *buried*. Natomiast usuwanie pochowanych nie wpływa na powiązany rekord pogrzebu do momentu usunięcia ostatniego pochowanego, wtedy usuwany jest też rekord z tabeli *funeral* - odpowiada to sytuacji, gdy na przykład następuje likwidacja grobu. Jeśli kwaterna cmentarna zostanie bez pochowanych, należy zaznaczyć to w bazie poprzez wpisanie wartości *NULL* w pole nagrobka tej kwatery - jest ona wtedy ponownie wolna.

Administrator praktycznie nie ingeruje w dane o pogrzebach i zmarłych, które są przechowywane w bazie przez zakłady pogrzebowe, choć ma teoretycznie do tego prawo, na przykład gdy zakład dopuści się jakiegoś przekłamania. Znacznie częściej dodaje i usuwa ofertę w tabelach powiązanych z pochowanym, taką jak stroje czy pojemniki, czy też w tabelach powiązanych z pogrzebem, taką jak pary ksiądz - świątynia. Są to transakcje jednoetapowe, wyjątkiem są działania na tabeli *cemetery*. Po dodaniu nowego cmentarza należy uzupełnić tabelę *quarter* o kwatery na tym cmentarzu. Transakcja usuwania rekordu, który jest kluczem obcym dla rekordów w innych tabelach, jest przeprowadzana zgodnie z zasadą przyjętą w rozdziale *Zapewnienie spójności*.

## 5. Zapewnienie spójności

Spójność danych zapewniona jest przez ORM. W polach klasy będącej modelem można zdefiniować zachowanie, które będzie stosowane w przypadkach, kiedy na przykład rekord z powiązanej tabeli, do którego próbujemy się odwołać, nie istnieje.

Generalna zasada, jaką przyjmujemy, jest taka, że jeśli usuwany jest jakiś rekord, który jest kluczem obcym dla innych, to nie usuwamy tych rekordów lawinowo. Jeśli rekord z powiązanej tabeli nie zostanie znaleziony, to użytkownik zobaczy dane pole jako niezdefiniowane (np. po bankructwie zakładu pogrzebowego jego pochówki będą wyświetlane z zakładem nieznanym). Pozwoli to zachować jak najwięcej danych dla gości szukających pochowanych (informacja pozbawiona domu pogrzebowego będzie nieco wybrakowana, ale zostanie reszta, potencjalnie interesujących dla szukającego, faktów).

Przewidziane są odstępstwa od powyższej zasady. Usuniecie cmentarza (np. w skutek sprzedaży terenu cmentarza deweloperowi) zapoczątkuje reakcję łańcuchową usuwania kwater, pogrzebów i zmarłych. Podobne zachowanie wiąże się z usunięciem kwatery czy pogrzebu.

## Etap 3 i 4

Zgodnie z założeniami, podczas realizacji skorzystano ze wzorca **Model View Template**. Poniżej w krótki sposób zostało opisane zastosowanie poszczególnych składowych wzorca.

### Baza danych - Model

Podczas realizowania bazy danych wykorzystano mapowanie obiektowo-relacyjne (ORM SQLAlchemy), w celu zapisywania aktualnej struktury bazy danych wykorzystano migracje (Flask-migrate). Aby wytworzyć tabelę należało napisać odpowiednią klasę w Python’ie.

Listing 1: Klasa w Python’ie realizująca tabelę pochowanego

```
1 @whooshee.register_model('first_name', 'last_name', 'cause_of_death')
2 class Buried(db.Model):
3     # connected with outfit o-t-m
4     # connected with container o-t-m
5     # connected with quarter o-t-m
6
7     # attributes
8     __tablename__ = 'buried'
9     id = db.Column(db.Integer, primary_key=True, autoincrement=True, nullable=False)
10    first_name = db.Column(db.Text, nullable=False)
11    last_name = db.Column(db.Text, nullable=False)
12    birth_date = db.Column(db.Date(), nullable=True)
13    death_date = db.Column(db.Date(), nullable=True)
14    cause_of_death = db.Column(db.Text, nullable=True)
15
16    # foreign keys
17    outfit_id = db.Column(db.Integer, ForeignKey('outfit.id'), nullable=False)
18    container_id = db.Column(db.Integer, ForeignKey('container.id'), nullable=False)
19    quarter_id = db.Column(db.Integer, ForeignKey('quarter.id'), nullable=False)
20    funeral_id = db.Column(db.Integer, ForeignKey('funeral.id'), nullable=False)
21
22    # relationships
23    quarter = relationship("Quarter", back_populates="buried", single_parent=True)
24    funeral = relationship("Funeral", back_populates="buried", single_parent=True, cascade="all, delete-orphan",
25                           passive_deletes=True)
26    container = relationship("Container")
27    outfit = relationship("Outfit")
28
29    def __repr__(self):
30        return f'Pochowany {self.first_name} {self.last_name}, ur. {self.birth_date}, zm. {self.death_date}'
31
32    def __getitem__(self, field):
33        return self.__dict__[field]
```

Rozwiązań okazało się łatwe w zastosowaniu i pozwoliło na relatywnie prostą implementację struktury bazy. Jedynymi trudnościami było stworzenie relacji *many-to-many*, jednakże ostatecznie problem udało się rozwiązać poprzez znalezienie i zmodyfikowanie rozwiązania proponowanego w dokumentacji.

Należało też zadbać o integralność bazy, co na poziomie administratora zostało zrealizowane na gruncie modeli (np. przez wymaganie klucza obcego lub odpowiednie event’y).

Listing 2: Event usuwający pochowanych których pogrzeb został usunięty

```
1 # usuwanie pogrzebu bez pochowanych
2 @event.listens_for(Buried, 'before_delete')
3 def delete_reference(mapper, connection, target):
4     # after_flush used for consistent results
5     @event.listens_for(Session, 'after_flush', once=True)
6     def receive_after_flush(session, context):
7         # if this buried was last in funeral
8         if not target.funeral.buried:
9             session.delete(target.funeral)
```

Rozwiązań okazało się efektywne na poziomie panelu administratora - panel użytkownika wymagał osobnego zapewnienia integralności realizowanej przez niektóre event’y. Przykładowo, należało aktualizować cenę pogrzebu, co było niewykonalne na poziomie modeli, koniecznym więc było napisanie rozwiązań, które zrealizują to w inny sposób. Przykładowo, aktualizacja ceny pogrzebu powinna odbywać się podczas dodawania/usuwania/edykcji pochowanego - kod aplikacji korzystający z obiektów będących odzwierciedleniem rekordu z bazy zmienia ich atrybuty, po czym taki zedytowany obiekt zmienia rekord w bazie. Dzięki temu poprawnie aktualizuje się cena zarówno na poziomie administratora, jak i na poziomie użytkownika.

Listing 3: Aktualizacja całkowitej ceny pogrzebu w przypadku usunięcia jednego z pochowanych

```

1 # change funeral total price when buried is deleted
2     funeral = Funeral.query.filter_by(id=buried_to_edit.funeral_id).first()
3     funeral.total_price = funeral.total_price - buried_to_edit.container.price - buried_to_edit.outfit.price -
4     buried_to_edit.quarter.price
5
6 #(...)

7     funeral.total_price = funeral.total_price + buried_to_edit.container.price + buried_to_edit.outfit.price +
     buried_to_edit.quarter.price

```

Na poziomie modeli została także zrealizowana oferowana przez aplikację możliwość wyszukiwania w bazie danych rekordów, które zawierają konkretną frazę. Posłużyła do tego biblioteka *Whoosh*, która odpowiada za indeksowanie treści wprowadzanych rekordów, tak by można je było wydajnie przeszukiwać. W projekcie funkcjonalność ta została wprowadzona za pośrednictwem pakietu *Flask-Whooshee*. Funkcja *register\_model* poprzedzona dekoratorem *@whooshee* (widoczna na szczycie listingu 1) zapewnia, że treść atrybutów, które są jej argumentami, będzie indeksowana, to znaczy będzie możliwe przeszukiwanie określonych tam kolumn tabeli, by dopasować zadaną frazę. W aplikacji podawanie frazy do odnalezienia zaimplementowane jest w jednym z widoków.

## Widoki - View

Za logikę z obranej architekturze aplikacji odpowiadają **widoki**. Są to funkcje Pythona, poprzedzone dekoratorem *@app.route*. Definiowane w niej jest zachowanie serwera dla danego URL, a zwracany jest szablon Jinja2 (HTML z dodatkowymi znacznikami), z danymi które zostały w jakiś sposób przetworzone lub pozyowane z bazy danych.

Działanie widoku prezentuje poniższy listing. Najpierw pobierane są dane z formularza, a następnie następuje sprawdzenie czy użytkownik lub administrator o danej nazwie już istnieją. Jeśli istnieją, wyświetlna jest na stronie powiadomienie, a w przeciwnym wypadku tworzony jest nowy użytkownik, który zapisywany jest w bazie danych. Powodzenie operacji powoduje utworzenie nowego użytkownika i zapisanie go w bazie danych.

Listing 4: Widok odpowiedzialny za rejestracje użytkownika

```

1 @app.route('/signup-user', methods=['GET', 'POST'])
2 def signup_user():
3     form = RegisterUserForm()
4     if form.validate_on_submit():
5         hashed_password = generate_password_hash(form.password.data, method='sha256')
6
7         if Administrator.query.filter_by(login=form.username.data).first() or FuneralHome.query.filter_by(
8             login=form.username.data).first():
9             flash('This user already exists')
10        else:
11            new_user = FuneralHome(login=form.username.data, password=hashed_password, name=form.name.data,
12                                    voivodeship=form.voivodeship.data, county=form.county.data,
13                                    locality=form.locality.data,
14                                    phone=form.phone.data, price=form.price.data)
15            db.session.add(new_user)
16            db.session.commit()
17            flash('New funeral agency created')
18
19    return render_template('signup-user.html', form=form)

```

Jak wspomniano wcześniej w widoku został wykorzystany formularz. Formularze, podobnie jak modele, tworzone są w klasie Pythona. Ich przewaga nad prostym formularzem jest taka, że posiadają już zaimplementowaną walidację i są odporne na wiele podatności (np. CSRF, XSS). Warto wspomnieć, że zadano także o poufność danych wprowadzonych danych. Wdrożony projekt korzysta z protokołu HTTPS, który szyfruje dane podczas komunikacji klient-serwer.

Listing 5: Formularz odpowiedzialny za rejestracje administratora

```

1 class RegisterAdminForm(FlaskForm):
2     username = StringField('username', validators=[InputRequired()], render_kw={"placeholder": "Username"})
3     password = PasswordField('password', validators=[InputRequired(), EqualTo('confirm',
4                                                               message='Passwords must match')], render_kw={"placeholder": "Password"})
5     confirm = PasswordField('Repeat Password', render_kw={"placeholder": "Repeat password"})
6     special_key = PasswordField('admin code', validators=[InputRequired()], render_kw={"placeholder": "Special key"})
7

```

## Frontend - Template

Podczas implementacji, zgodnie z założeniami, skorzystano z silnika szablonów Jinja2. Najważniejsze cechy szablonów, które zostały wykorzystane w projekcie:

1. korzystanie z pętli i if'ów,
2. możliwość wyodrębnienia powtarzających się elementów i umieszczenie ich w osobnych plikach

Poniżej przykład zastosowania pętli w szablonie. Z widoku została przekazana lista nazw kolumn i lista słowników (każdy słownik przechowuje dane jednego rekordu).

Listing 6: Szablon Jinja2 generujący tabelę

```
1 <table>
2   <tr>
3     {% for column_name in column_names %}
4       <th>{{ column_name }}</th>
5     {% endfor %}
6   </tr>
7   {% for record in query_records %}
8     <tr>
9       {% for column_name in column_names %}
10         <td>{{ record[column_name] }}</td>
11       {% endfor %}
12     </tr>
13   {% endfor %}
14 </table>
```

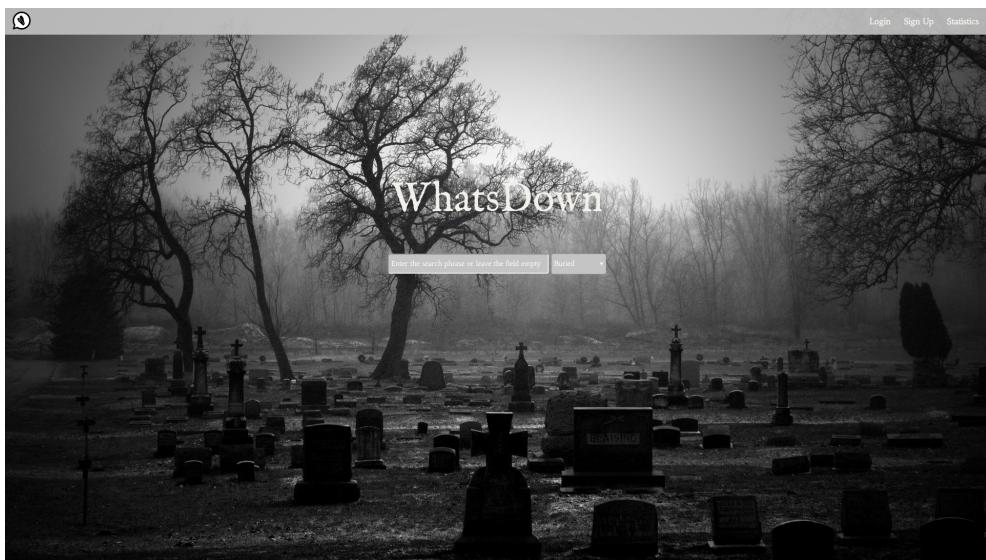
## Wygląd strony internetowej

### Strona główna

Aplikacja została wdrożona na serwer, dzięki czemu może zostać otworzona w przeglądarce pod warunkiem dostępu do Internetu. Projekt każdej z podstron został wykonany w internetowym edytorze **Figma**, a sama strona w JS, HTML i CSS (szablony Jinja2). Strona jest responsywna, dzięki czemu zapewnia wysokiej jakości UX zarówno dla użytkownika korzystającego ze smartfona, jak i komputera stacjonarnego.

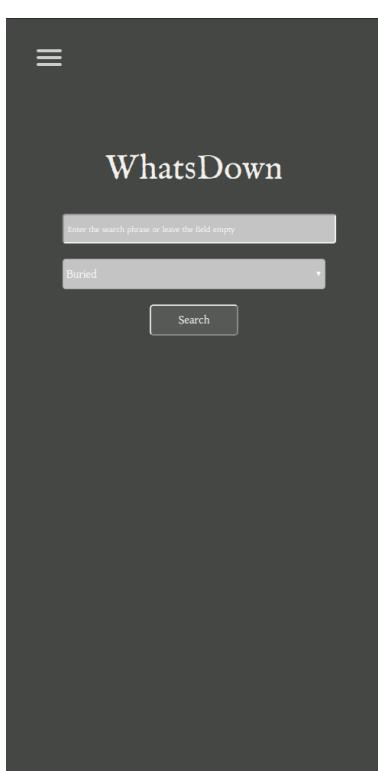
Na stronie głównej istnieje możliwość wyszukania interesujących nas pogrzebów ze względu na konkretny jego element oraz zalogowania się, rejestracji i uzyskania statystyk.

W combo box'ie po prawej stronie możemy wybrać, w jakiej kategorii chcemy przeprowadzić wyszukiwanie. Natomiast w polu tekstowym po lewej stronie możemy wprowadzić szukaną frazę (pokażą się te rekordy z wybranej kategorii, które w którymkolwiek swoim polu zawierają frazę jako całość treści pola lub jej ciągły fragment, wielkość liter nie ma tu znaczenia); pole tekstowe można także zostawić puste (pokażą się wszystkie rekordy z danej kategorii).

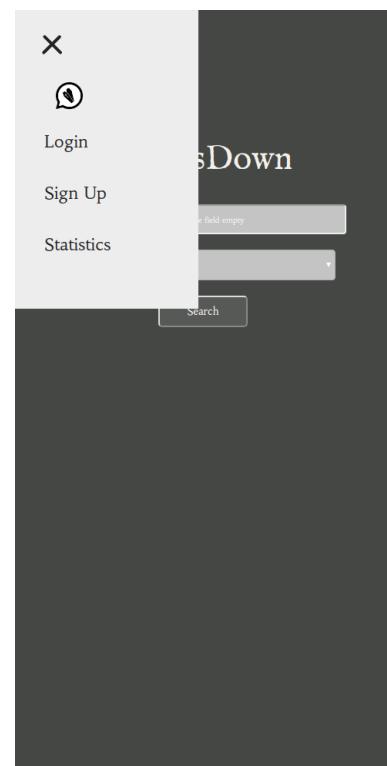


Rysunek 3: Whatsdown - strona główna

Dla komfortu użytkowników urządzeń przenośnych, style zostały dobrane tak, aby strona działała jak najszybciej. Zdecydowano o zastąpieniu obrazka z wersji desktopowej ciemnym tłem, zaś górnego navbar'u - rozwijanym menu typu hamburger. Podobnie postąpiono we wszystkich pozostałych podstronach.



Rysunek 4: Strona główna dla smartfonów



Rysunek 5: Strona główna dla smartfonów - rozwijane menu w stylu *hamburger*

FIRST NAME	LAST NAME	BIRTH DATE	DEATH DATE	CAUSE OF DEATH	OUTFIT	CONTAINER	QUARTER	FUNERAL
Maciej	Marczak	1938-01-06	2019-07-31	utonięcie	Garnitur marki Lancerto, rozmiar M, kolor czarny	Trumna marki Polanstronk, wykonany z dąb	Kwatra na cmentarzu w Wrocław przy ulicy Grabiszyńska, rząd 29, kolumna 50	Pogrzeb z 2019-08-06
Happy	Weasley	1980-09-17	2010-01-06	utonięcie	Garnitur marki Vistula, rozmiar L, kolor czarny	Urna marki Debest, wykonany z metal	Kwatra na cmentarzu w Wrocław przy ulicy Grabiszyńska, rząd 3, kolumna 5	Pogrzeb z 2010-01-11
Hermenegilda	Hababdibaldo	1938-01-06	2020-01-20	zawal serca	Suknia marki Versace, rozmiar M, kolor czarny	Urna marki Debest, wykonany z metal	Kwatra na cmentarzu w Wrocław przy ulicy Grabiszyńska, rząd 15, kolumna 16	Pogrzeb z 2020-01-24

Rysunek 6: Whatsdown - wyniki wyszukiwania w kategorii *Buried* w wersji desktopowej

FIRST NAME	Maciej
LAST NAME	Marczak
BIRTH DATE	1988-01-21
DEATH DATE	2019-07-31
CAUSE OF DEATH	utonięcie
OUTFIT	Garnitur marki Vistula, rozmiar L, kolor czarny
CONTAINER	Trumna marki Polanstronk, wykonany z dąb
QUARTER	Kwatra na cmentarzu w Wrocław przy ulicy Grabiszyńska, rząd 3, kolumna 5
FUNERAL	Pogrzeb z 2019-08-03

Rysunek 7: Whatsdown - wyniki wyszukiwania w kategorii *Buried* w wersji mobilnej

W polu tekstowym widocznym na górze strony wyszukiwania można przeprowadzić filtrowanie wyników, to znaczy w wypadku, gdy zwróconych rekordów będzie zbyt wiele, można ograniczyć ich liczbę, wpisując w pole frazę filtrującą. Na ekranie zostaną tylko te rekordy, które w treści któregoś ze swych atrybutów zawierają jako spójny fragment daną frazę.

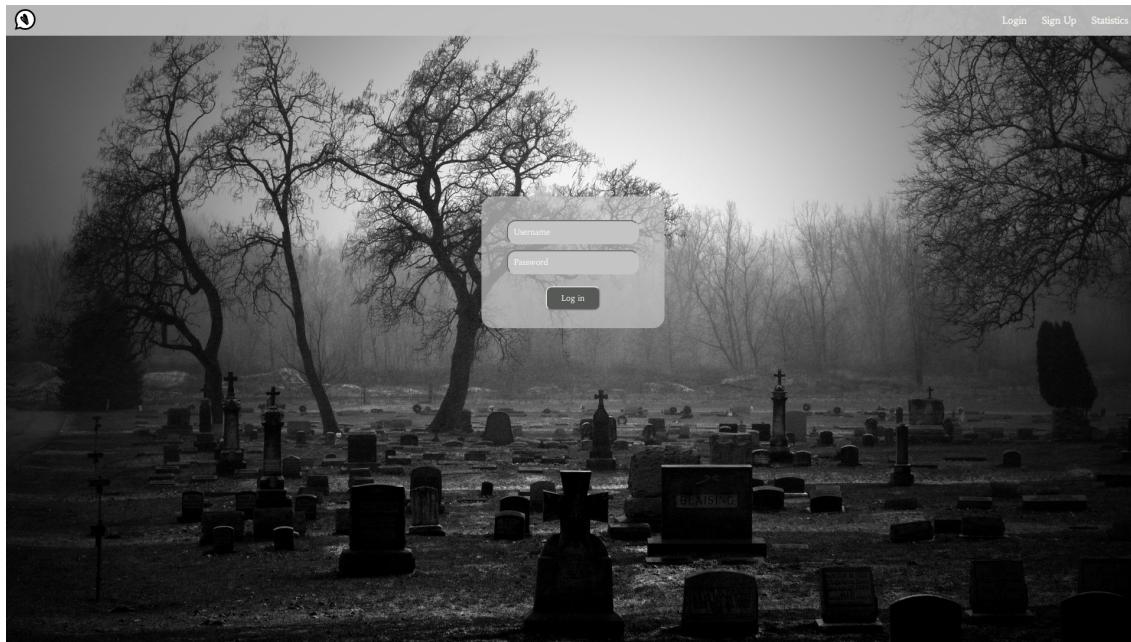
The screenshot shows a table with columns: FIRST NAME, LAST NAME, BIRTH DATE, DEATH DATE, CAUSE OF DEATH, OUTFIT, CONTAINER, QUARTER, and FUNERAL. There are two rows of data. A search bar at the top right says "Enter a phrase to filter" and a "Filter" button is next to it.

FIRST NAME	LAST NAME	BIRTH DATE	DEATH DATE	CAUSE OF DEATH	OUTFIT	CONTAINER	QUARTER	FUNERAL
Happy	Weasley	1980-09-17	2010-01-06	utonięcie	Garnitur marki Vistula, rozmiar L, kolor czarny	Urna marki Debest, wykonany z metal	Kwatra na cmentarzu w Wrocław przy ulicy Grabiszynska, rząd 3, kolumna 5	Pogrzeb z 2010-01-11
Hermenegilda	Habadzibadło	1938-01-06	2020-01-20	zawał serca	Suknia marki Versace, rozmiar M, kolor czarny	Urna marki Debest, wykonany z metal	Kwatra na cmentarzu w Wrocław przy ulicy Grabiszynska, rząd 15, kolumna 16	Pogrzeb z 2020-01-24

Rysunek 8: Whatsdown - wyniki filtrowania w kategorii *Buried* (frazą *Debest*)

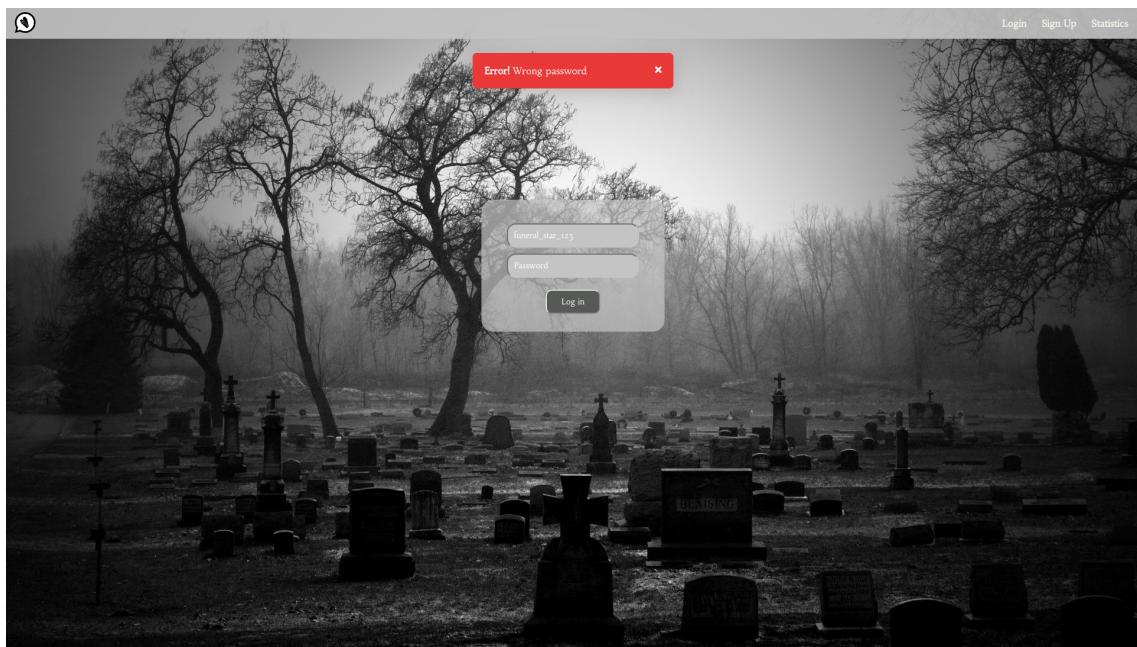
### Panel logowania i rejestracji

Panel logowania i rejestracji zostały wykonane w podobnej koncepcji i pozwalają za pomocą formularza utworzyć konto lub zalogować się na poprzednio utworzone.



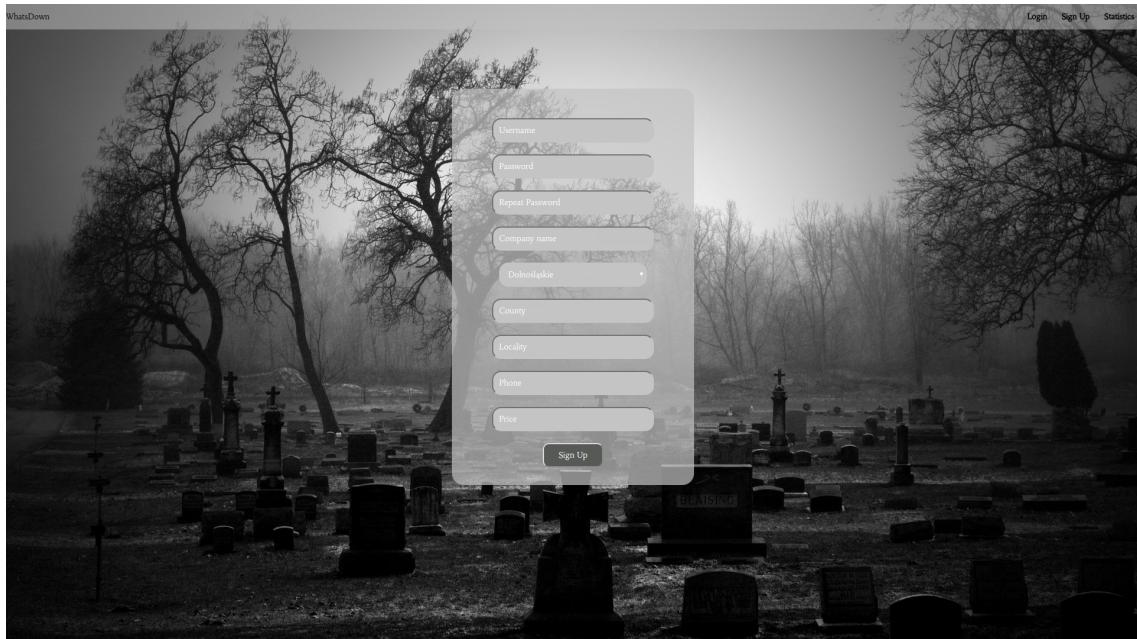
Rysunek 9: Whatsdown - panel logowania

W przypadku powodzenia użytkownik zyska możliwość dostępu do swojego panelu, zaś w przypadku niepowodzenia, zostanie wyświetlona wiadomość informująca o nieprawidłowym haśle lub nieistniejącym użytkowniku.

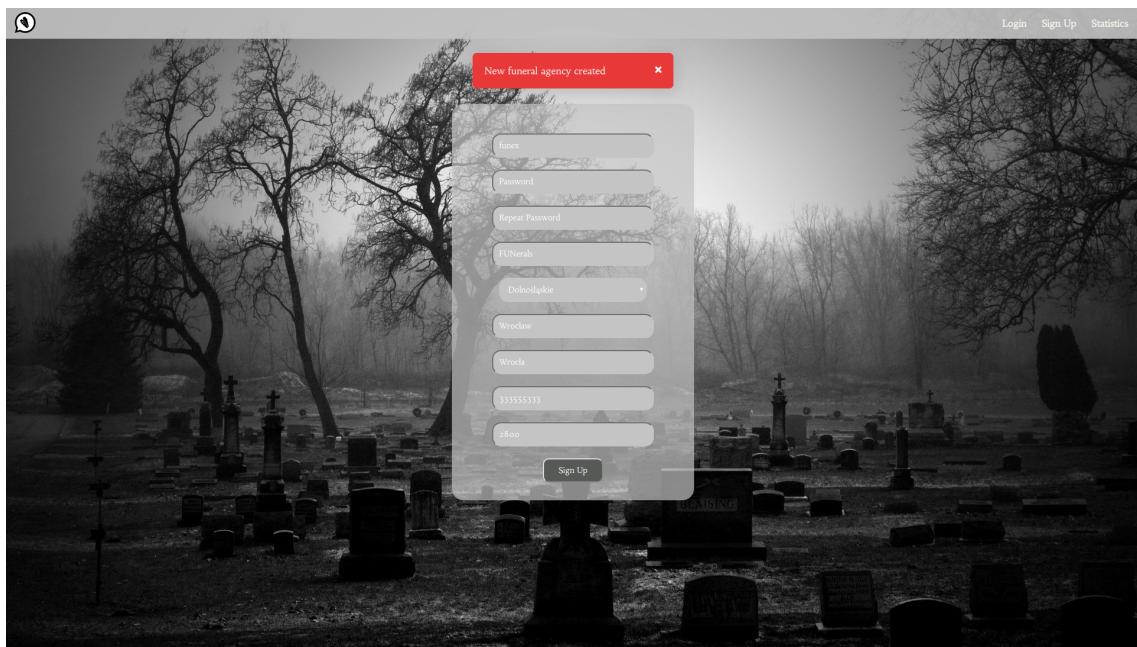


Rysunek 10: Whatsdown - zalogowanie nie powiodło się - błędne hasło

Formularz rejestracji działa na zasadzie analogicznej do formularza logowania. W przypadku powodzenia, zostanie wyświetlona wiadomość, a w bazie danych zostanie utworzony dany użytkownik, wraz z zahaszowanym hasłem.

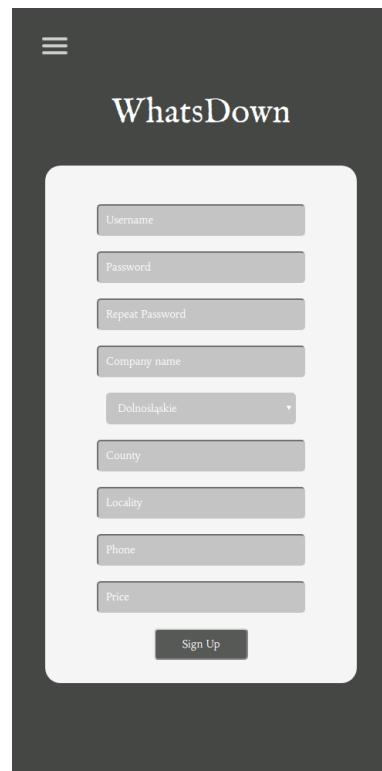


Rysunek 11: Whatsdown - panel rejestracji



Rysunek 12: Whatsdown - udana rejestracja

Wersja panelu rejestracji dla urządzeń mobilnych również została przygotowana.



Rysunek 13: Whatsdown - rejestracja - wersja dla użytkowników urządzeń mobilnych

## Statystyki Whatsdown

Zgodnie z założeniem, utworzona została podstrona, na której można było przeglądać udostępnione statystki.



Rysunek 14: Whatsdown - statystki

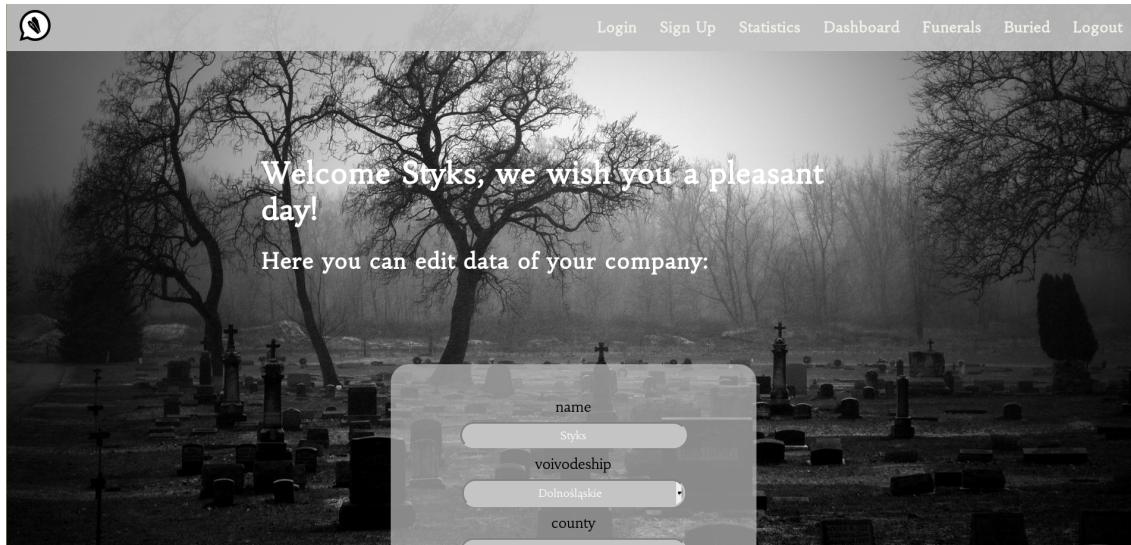
Wersja strony wyświetlającej statystki dla urządzeń mobilnych.



Rysunek 15: Whatsdown - statystki dla wersji mobilnej

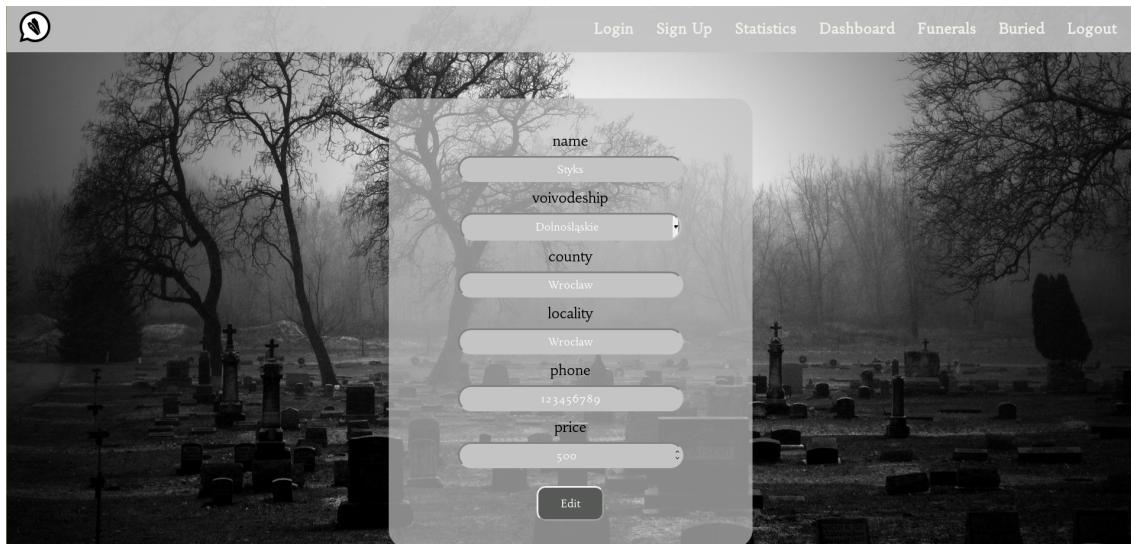
## Panel użytkownika

Każdy użytkownik, którym jest zarejestrowany zakład pogrzebowy, może po zalogowaniu się uzyskać dostęp do panelu użytkownika. Zgodnie z założeniami zapewnia on zakładowi pogrzebowemu możliwość dodawania, usuwania i ewentualnej edycji swoich danych oraz danych o pochowanych i pogrzebach.



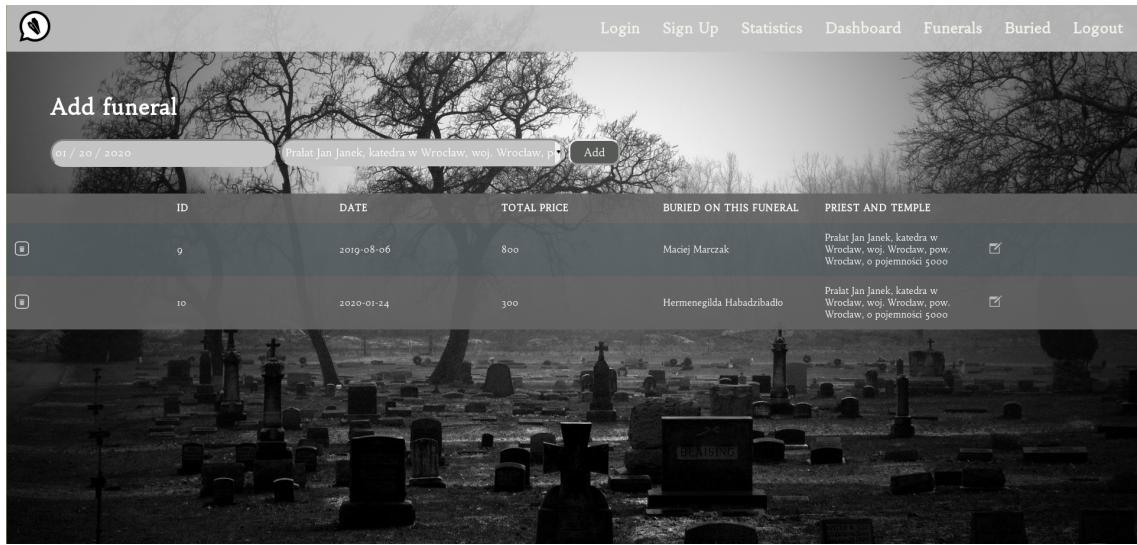
Rysunek 16: Whatsdown - panel użytkownika po zalogowaniu (*dashboard*)

W *dashboardzie* zakład w każdej chwili może edytować swoje dane.

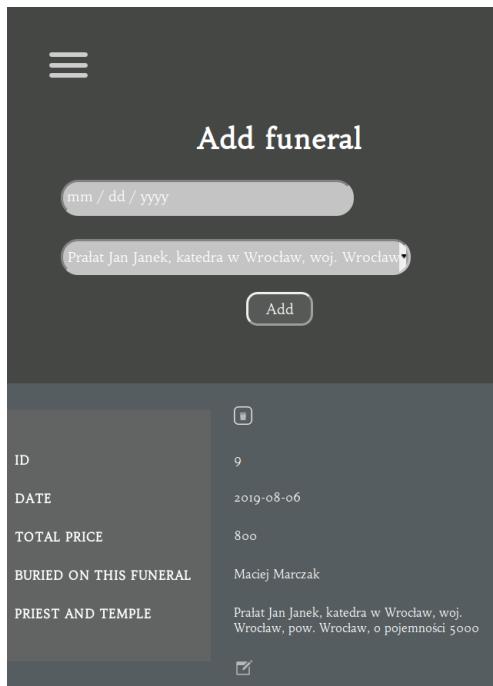


Rysunek 17: Whatsdown - formularz edycji danych użytkownika

Na kolejnych zrzutach ekranu zostanie zademonstrowany proces dodawania do bazy nowego pogrzebu oraz pochowanych wtedy osób. Pierwszym krokiem jest wejście w zakładkę *Funerals*.

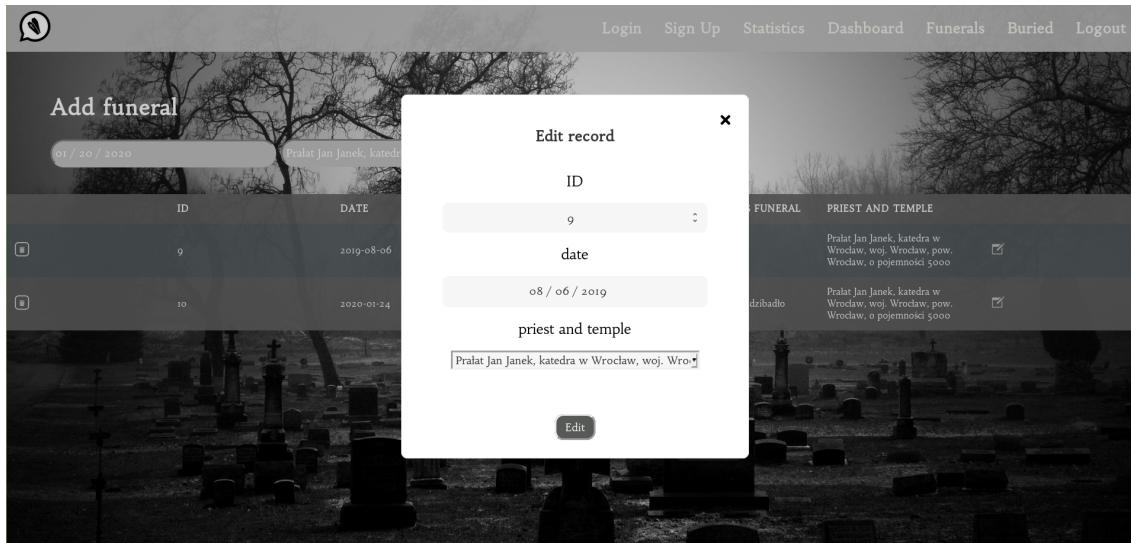


Rysunek 18: Whatsdown - formularz dodawania nowego pogrzebu w wersji desktopowej



Rysunek 19: Whatsdown - formularz dodawania nowego pogrzebu w wersji mobilnej

W dolnej części okna dostępna jest lista wszystkich pogrzebów, które przeprowadził zakład i wprowadził do bazy. Każdy z tych rekordów można usunąć, a także edytować.



Rysunek 20: Whatsdown - formularz edycji istniejącego pogrzebu

W łatwy sposób można w górnej części strony wybrać datę pogrzebu oraz odprawiającego go kapłana. Kliknięcie *add* od razu spowoduje dodanie pogrzebu o określonych atrybutach do bazy oraz przekierowanie na podstronę *Buried*.



Rysunek 21: Whatsdown - formularz dodawania pochowanego do pogrzebu

W dolnej części okna widzimy listę pochowanych przez dany zakład i umieszczonych w bazie (podobnie jak w przypadku pogrzebów z możliwością usunięcia i edycji). Natomiast w górnej części możemy wprowadzić do formularzy dane pochowanego, po czym umieścić go w konkretnej kwaterze, klikając *Add*. Na ewentualność zbiorowego pochówku dajemy możliwość pochowania kilku osób w tej samej kwaterze – wystarczy wprowadzić dane nowego pochowanego i znów zatwierdzić przyciskiem. Natomiast formularz *End burial...* pozwala po zakończeniu wszystkich pochówków ustawić na danej kwaterze nagrobek – w ten sposób nie będzie już ona dostępna dla innych pochowanych. Jest to forma zabezpieczenia przed błędny przypisaniem

pochowanemu kwaterom w przyszłości, a także ogranicza liczbę kwater, jakie system podpowiada użytkownikowi jako dostępne. Aby pochować kogoś w kwaterze z już ustawionym wcześniej nagrobkiem, użytkownik powinien skontaktować się z administratorem.

## Panel administratora

Ta część aplikacji umożliwia zarządzanie całą bazą danych. Uprawniony użytkownik może tworzyć nowe cmentarze, kwatery, dodawać nowe stroje etc. Zasadniczo, z panelu można edytować dowolne dane w bazie. Do panelu admina można uzyskać dostęp przez wcześniejszą rejestrację w ukrytym widoku. Aby się zarejestrować jako admin należy także dysponować tajnym kodem.

The screenshot shows a web-based administrative interface for a database. At the top, there is a navigation bar with several tabs: Admin, Home, Cemetery, Quarter, Outfit, Container, Tombstone, Priest, Temple, Priest Temple, Funeral, Buried, Administrator, and Funeral Home. The 'Temple' tab is currently selected, highlighted in blue. Below the navigation bar, there is a toolbar with three buttons: 'List (2)', 'Create', and 'With selected'. The 'List (2)' button is also highlighted in blue. The main content area displays a table with two rows of data. The columns are labeled: Voivodeship, County, Locality, Religion, Capacity, and Rank. The first row shows 'Dolnośląskie' as the Voivodeship, 'Wrocław' as the County, 'Wrocław' as the Locality, 'Katalizm' as the Religion, '600' as the Capacity, and 'Katedra' as the Rank. The second row shows 'Dolnośląskie' as the Voivodeship, 'Wrocław' as the County, 'Wrocław' as the Locality, 'świeckość' as the Religion, '50' as the Capacity, and 'dom pożegnań' as the Rank. Each row has a small edit icon next to the first column.

	Voivodeship	County	Locality	Religion	Capacity	Rank
<input type="checkbox"/>	Dolnośląskie	Wrocław	Wrocław	Katalizm	600	Katedra
<input type="checkbox"/>	Dolnośląskie	Wrocław	Wrocław	świeckość	50	dom pożegnań

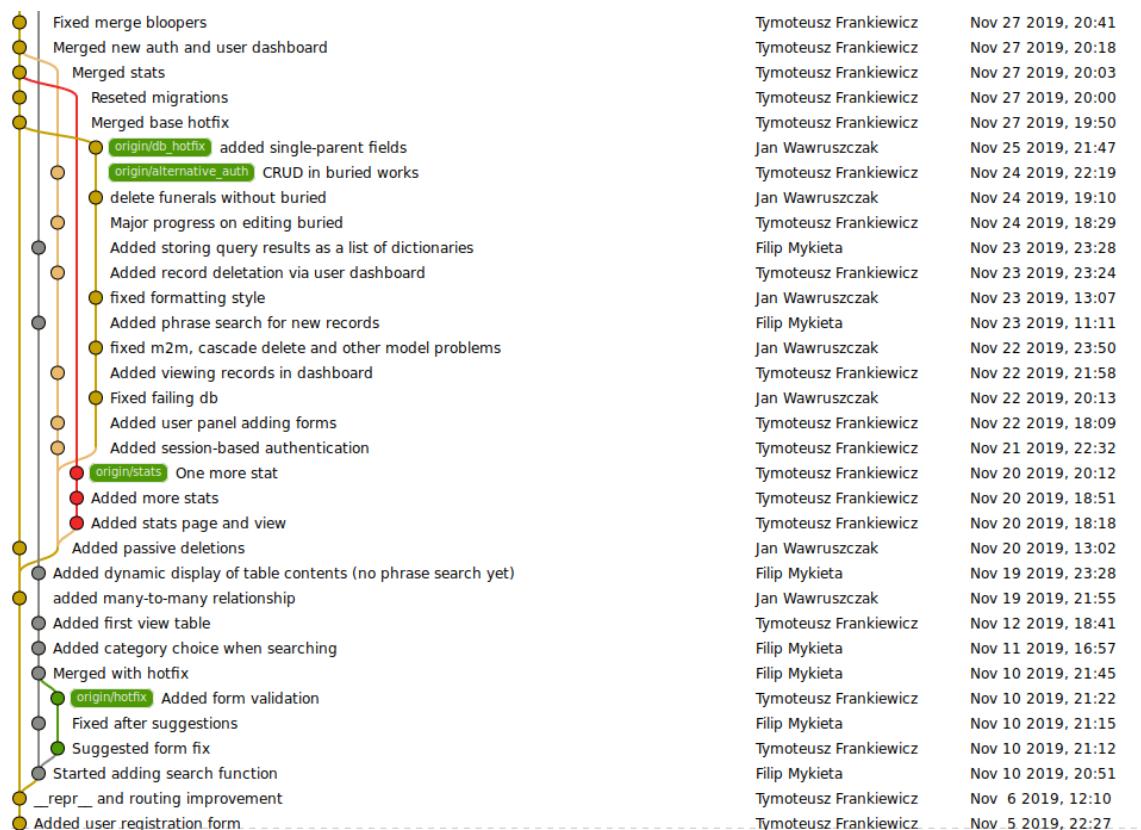
Rysunek 22: Whatsdown - panel administratora

## Podsumowanie i wnioski

Projekt okazał się ciekawym doświadczeniem, które pozwoliło zobaczyć każdy aspekt związany z tworzeniem oprogramowania, zarówno część „backendową” jak i „frontendową”. Dzięki temu każdy z nas potrafi lepiej określić, w którym z tych działów chciałby się rozwijać. Interesujące okazało się to, jak odpowiednie zgranie zespołu, dobry podział prac, rozsądne terminy i dobry dobór technologii wpływają na szybkość tworzenia aplikacji. Ze względu na dobre zgranie grupy udało się uniknąć opóźnień i w sympatycznej atmosferze powstała zadowalająca nas aplikacja.

Problemy, jakie napotkaliśmy, wiązały się głównie z początkową nieznajomością działania niektórych rozwiązań, z których korzystaliśmy - część rzeczy należało dostosować do naszych potrzeb, natomiast ze względu na fakt, że nie wszyscy mieli z nimi do czynienia wiązało się to z problemami działania w sposób inny, niż oczekiwany. Co za tym idzie, sporo czasu zostało poświęcone na uczenie się od podstaw i poprawki, a nie tylko na efektywne pisanie kodu.

Nieocenioną pomocą okazał się system kontroli wersji **git** wraz ze zdalnym repozytorium GitHub. Dzięki takiemu rozwiązaniu cały zespół mógł jednocześnie wprowadzać zmiany w kodzie.



Rysunek 23: Git - część historii projektu

Narzędzia, które wykorzystywaliśmy, cechowały się dobrą dokumentacją (w szczególności należy pochwalić dokumentację SQLAlchemy i Flask'a), dzięki czemu dużo problemów można było rozwiązać bez zbędnego poszukiwania odpowiedzi na forach.

Przy tworzeniu części wizualnej aplikacji istotnym był fakt, że przed rozpoczęciem pisania kodu w JS + HTML + CSS wykonaliśmy odpowiednie projekty w internetowym edytorze stron, aby móc to później wykorzystać przy implementacjach.

Problematyczna była kwestia bezpieczeństwa - posiadana wiedza czasami nie była wystarczająca i dla

pewnych wrażliwych elementów zakładamy, że automatyczne rozwiązania framework'ów poniekąd rozwiązują problemy z bezpieczeństwem bez ingerencji programisty. Mimo to, udało się manualnie wdrożyć kilka zabezpieczeń przed najpopularniejszymi atakami.

Ze względu na to, że projekt po prostu nam się bardzo spodobał, zdecydowaliśmy się wdrożyć go na wykupiony serwer - będzie to dobry element portfolio, którym można się pochwalić na rozmowie kwalifikacyjnej.

Wszyscy uznajemy ten projekt za dobre doświadczenie, którego efekt już jest zadowalający, a nie wykluczamy jego poprawiania w przyszłości.

Projekt można zobaczyć na witrynie [whatsdown.tscode.net](https://whatsdown.tscode.net).