Kamil Matuła
Piotr Maślanka

28.05.2013

Opis kodu

Projekt 'równodzień'
obsługa systemu bibliotecznego

Projekt z odległej perspektywy

Podstawowymi elementami, znajdującymi się w głównym katalogu projektu, są:

- folder **books**: aplikacja obsługująca książki i ich egzemplarze
- folder **media**: zawiera arkusze stylów CSS do formatowania HTML
- folder **people:** aplikacja obsługująca czytelników i bibliotekarzy
- folder **rentings**: aplikacja obsługująca wypożyczenia i zwroty
- folder **templates**: zawiera szablony HTML do prezentacji danych
- plik **manage.py**: plik administracyjny Django
- plik **settings.py**: konfiguracja projektu
- plik **urls.py**: informuje którą funkcję należy wywołać w zależności od URL
- plik **wsgi.py**: plik administracyjny Django

Aplikacja books

Aplikacja books zajmuje się obsługą książek oraz ich egzemplarzy. Zawierają się w niej definicje tabel danych, charakterystycznych dla nich zachowań oraz formularzy i kontrolerów - czyli funkcji które obsługują żądania użytkownika. Tak więc widać że elementy te są bardziej pogrupowane tematycznie niż po swoim typie.

Modele aplikacji books

Dwa modele zamieszczone w pliku **books/models.py** odpowiadają za encje **książka** i **egzemplarz** (odpowiednio Book i BookInstance). Poniżej podano w jaki sposób pola encji odpowiadają polom modeli:

Book

Encja ***Book*** określa informacje o książkach dostępnych w systemie.
Zawiera następujące pola:

- - ***idBook*** – klucz podstawowy, określa identyfikator książki, pole liczbowe całkowite, unikalne;
- - ***author*** – określa nazwę autora książki, pole znakowe, maksymalna dł. – 40;
- - ***title*** – określa tytuł książki, pole znakowe, maksymalna dł. – 40;
- - ***year*** – określa rok wydania książki, pole liczbowe całkowite;
- - ***isbn*** – określa ISBN danej książki, pole znakowe, maksymalna dł. – 13, unikalne, sprawdzana suma kontrolna podczas wprowadzania numeru ISBN co zapobiega wprowadzeniu niepoprawnej wartości;
- - ***publish_place*** – określa miejsce wydania książki, pole znakowe, maksymalna dł. - 40;
- - ***year_pl*** – określa rok wydania polskiego, pole znakowe, maksymalna dł. – 40;
- - ***language*** – określa język książki, pole znakowe, maksymalna dł. – 20;
- - ***translation*** – określa autora przekładu książki, maksymalna dł. – 40;

BookInstance

Encja ***BookInstance*** określa informacje o egzemplarzach danej książki dostępnej systemie.
Zawiera następujące pola:

- - ***idBookInstance*** – klucz podstawowy, określa identyfikator danego egzemplarza, pole liczbowe całkowite, unikalne;
- ***book*** – klucz obcy, określa identyfikator modelu książki z tabeli *Book;*
- - ***code*** – pole znakowe, określa numer lub kod kreskowy książki, maksymalna dł. - 14, unikalne;
- - ***is_damaged*** – pole logiczne, określa czy książka jest uszkodzona;
- - ***is_lost*** – pole logiczne, określa czy książka jest zagubiona;

Kontrolery aplikacji books

Czynnościami które obsługuje aplikacja books są:

- dodawanie, edycja i usuwanie książek
- dodawanie, edycja i usuwanie egzemplarzy

Dokonywane jest to przez funkcje kontrolerów, pogrupowane tematycznie. Rezydują one w **books/instances.py** oraz **books/books.py**. Ze względu na ich ścisłe podobieństwo, omówiony zostanie tylko **books/books.py**. Podobnież w aplikacji **people** występuje taki rozkład obiektów i funkcji, dlatego również on zostanie pobieżnie omówiony.

Plik ten zawiera:

- obiekt **BookForm** dziedziczący po **forms.ModelForm** na podstawie modelu **Book**, stanowiącym formularz do dodawania/edycji.
- funkcję-kontroler *add_book()*, zajmującą się dodawaniem książki do bazy
- funkcję-kontroler *edit_book()*, zajmującą się edycją książki w bazie
- funkcję-kontroler *delete_book()*, zajmującą się usunięciem książki z bazy

BookForm jest obiektem formularza, wygenerowanym automatycznie przez Django na podstawie modelu *Book*. Jedyną modyfikacją w stosunku do tej generacji jest wprowadzenie dodatkowej walidacji – sprawdzanie poprawności numeru ISBN. Odbywa się to – po usunięciu pauz – dzięki wykorzystaniu biblioteki *pyisbn*, która dokonuje sprawdzenia. Jeśli numer jest poprawny, metoda zwraca ISBN, a jeśli nie – wyjątek błędu walidacji formularza, który następnie obsłuży Django wyświetlając użytkownikowi stosowną wiadomość.

Funkcja-kontroler *add_book()* ma za zadanie dodać książkę. Nie potrzebuje kontekstu – dlatego jedyne co przyjmuje to parametr *request*, w którym zapisane są dane które potencjalnie użytkownik mógł chcieć wysłać. Funkcja ta zasadniczo obsługuje dwa scenariusze:

1. **Wywołanie bez danych lub wywołanie z błędnymi danymi.**
   W takim wypadku funkcja ma wyświetlić formularz z pustymi polami – lub z polami wypełnionymi danymi użytkownika i informacjami o błędzie. Zauważmy że *render_to_response()*, o którym wiemy że jego zadaniem jest wyświetlenie treści zostanie osiągnięte tylko jeśli:
   a) Metodą wywołania nie jest *POST* (czyli użytkownik chce wyświetlić formularz)
   b) Metodą wywołania jest *POST* (użytkownik wypełnił formularz), ale dane są złe (czyli nie jest prawdą że `form.is_valid()`).

Jeśli tak jest, to zostanie wyświetlony szablon formularza poprzez *render_to_response*.

2. **Wywołanie z poprawnymi danymi.**
   W takim wypadku funkcja utworzy i zapisze w bazie danych odpowiedni rekord (dokonuje tego `form.save()` – ponieważ formularz oparty jest o model, Django wie jak go utworzyć i wypełnić), a następnie przekieruje pod adres dostępu „do edycji" nowo utworzonego modelu (funkcja *redirect* – zwracająca *HttpResponse* które nie wyświetla treści, a przekierowuje).

Funkcja-kontroler *edit_book()* zachowuje się bardzo podobnie, ale jej celem jest edycja istniejącego rekordu. Dlatego funkcja ta przyjmuje dodatkowy argument, którym jest ISBN książki. Aby upewnić się że dany wpis istnieje, wywoływana jest funkcja *get_object_or_404* (która albo zwróci żądany obiekt, albo przerwie skrypt z błędem HTTP 404). Drobną modyfikacją jest podawanie przez parametr *instancje* konkretnego obiektu typu Book – dzięki temu Django wie, że nie chodzi o stworzenie nowego obiektu, a jedynie o manipulację obiektem istniejącym. W tym scenariuszu, gdy zapis odbędzie się poprawnie – czyli dane z formularza będą poprawne – nie nastąpi przekierowanie, a jedynie dalszy ciąg wykonania skryptu; tj. zostanie wyświetlony formularz wraz z „wiadomością" że dokonano zapisu. Obsługę wiadomości realizuje podsystem *messages* Django.

Funkcja-kontroler *delete_book()* jest nadzwyczaj prosta. Podobnie jak *edit_book()* operuje na konkretnej książce, więc przyjmuje jej ISBN. Jednak tuż po uzyskaniu dostępu do obiektu za pomocą *get_object_or_404()* konkretny rekord zostaje skasowany[1].

W pliku **books/instances.py** zawarte jest:

- Obiekt **BookInstanceForm**, stanowiący formularz do dodawania/edycji egzemplarza
- Funkcja-kontroler *add_instance()*
- Funkcja-kontroler *edit_instance()*
- Funkcja-kontroler *delete_instance()*

Obiekt **BookInstanceForm** ma dodatkowy parametr w klauzuli *class Meta*, którego nie miał **BookForm**. Dodatkowy atrybut *exclude* informuje nas o tym, że nie chcemy aby pole *book* egzemplarza było wyświetlane. Będzie się to odbywać dlatego, że w konkretnej książce jest opcja dodania egzemplarza, i jasne jest do której książki ma być on przypisany – użytkownik nie powinien być dodatkowo obarczony obowiązkiem wybrania książki.

Funkcja-kontroler *add_instance()* różnią się od swoich odpowiednika tym że automatycznie wypełnia pole *book* przed zapisem do bazy (ze względu na oczywisty fakt, że użytkownik nie może wybrać egzemplarza). Ponieważ ta funkcja wywoływana jest z parametrem ISBN książki, można to ustalić.

[1] Bazodanowy podsystem Django zapewnia że wszystkie rekordy które są powiązane z kasowanym obiektem także zostaną skasowane.

Funkcja-kontroler *delete_instance()* zachowuje się tak jak swój odpowiednik, jednak zapisuje do zmiennej ISBN przed skasowaniem egzemplarza, żeby przekserować użytkownika do strony książki odpowiedniej kasowanemu.

Aplikacja people

Zadaniem aplikacji *people* jest:

- Dodawanie, edycja i usuwanie bibliotekarzy
- Dodawanie, edycja i usuwanie czytelników

Modele aplikacji people

Modele rezydujące w **people/models.py** odpowiadają za encje czytelnika i bibliotekarza. Poniżej podano w jaki sposób pola encji odpowiadają polom modeli:

Reader

Encja ***Reader*** określa informacje o czytelnikach dostępnych w systemie.
Zawiera następujące pola:

- **- *number*** – określa numer czytelnika, pole dziesiętne, maksymalnie 4. cyfrowe, unikalne;
- - ***pesel*** – określa PESEL bibliotekarza, pole znakowe, maksymalna dł. - 50, sprawdzana suma kontrolna podczas wprowadzania numeru PESEL co zapobiega wpowadzaniu niepoprawnej wartości;
- - ***name*** – określa imię czytelnika, pole znakowe, maksymalna dł. - 50;
- - ***surname*** – określa nazwisko czytelnika, pole znakowe, maksymalna dł. - 50;
- - ***phone*** – określa nr telefonu czytelnika, pole znakowe, maksymalna dł. - 30;

Librarian

Encja ***Librarian*** określa informacje o bibliotekarzach dostępnych w systemie.
Zawiera następujące pola:

- **- *number*** – klucz podstawowy, określa numer bibliotekarza, pole dziesiętne, maksymalnie 4. cyfrowe, unikalne;
- - ***name*** – określa imię bibliotekarza, pole znakowe, maksymalna dł. - 50;
- - ***surname*** – określa nazwisko bibliotekarza, pole znakowe, maksymalna dł. - 50;
- - ***pesel*** – określa PESEL bibliotekarza, pole znakowe, maksymalna dł. - 50, sprawdzana suma kontrolna podczas wprowadzania numeru PESEL co zapobiega wpowadzaniu niepoprawnej wartości;

Kontrolery aplikacji people

Aplikacja *people* ma łącznie dwa formularze i sześć kontrolerów, opisanych poniżej:

- Obiekt **LibrarianForm**, stanowiący formularz do dodawania i edycji bibliotekarzy
- Obiekt **ReaderForm**, stanowiący formularz do dodawania i edycji czytelników
- Funkcja-kontroler *add_librarian()*, obsługująca dodawanie bibliotekarzy
- Funkcja-kontroler *edit_librarian()*, obsługująca edycję bibliotekarzy

- Funkcja-kontroler *delete_librarian()*, obsługująca usuwanie bibliotekarzy
- Funkcja-kontroler *add_reader()*, obsługująca dodawanie czytelników
- Funkcja-kontroler *edit_reader()*, obsługująca edycję czytelników
- Funkcja-kontroler *delete_reader()*, obsługująca kasowanie czytelników

Jedyną ciekawostką i różnicą tych elementów od swoich odpowiedników w aplikacji *books*, jest fakt iż formularze mają „przedefiniowane" pole PESEL. Uczyniono to z tego względu, iż baza PostgreSQL nie ma natywnie typu PESEL, a Django posiada takie pole formularza. Django zastąpi więc pole PESEL zdefiniowanym w modelu formularza, czego skutkiem ubocznym będzie to że użytkownik będzie musiał wprowadzić poprawny PESEL.

Aplikacja rentings

Aplikacja **rentings** zasadniczo różni się od dwóch poprzednich aplikacji. Jej zadaniem jest udostępnianie funkcji wypożyczenia książki i jej zwrotu. W tym celu definiuje ona jeden model, dwa formularze i dwa kontroleru.

Modele aplikacji rentings

Model rezydujący w **rentings/models.py** odpowiadają za encje wypożyczenia. Poniżej podano w jaki sposób pola encji odpowiadają polom modeli:

BookRent

Encja **BookRent** określa informacje o aktualnych wypożyczeniach zarejestrowanych w systemie. Zawiera następujące pola:

- **- *bookinstance*** – klucz obcy, określa identyfikator wypożyczonego egzemplarza z tabeli *Instance;*
- - ***who*** – klucz obcy, określa identyfokator obsługującego bibliotekarza z tabeli *Librarian;*
- - ***whom*** – klucz obcy, określa numer czytelnika z tabeli *Reader*, który wypożycza egzemplarz;
- - ***when_rented*** – określa kiedy wypożyczono dany egzemplarz, pole daty;
- - ***official_due*** – określa oficjalną date oddania egzemplarza, pole daty;
- - ***real_due*** – określa rzeczywistą date oddania egzemplarza, pole daty;

Kontrolery aplikacji rentings

W pliku **rentings/views.py** znajdują się obiekty:

- Obiekt **BookRentForm**, stanowiący formularz wypożyczenia książki.
- Obiekt **BookReturnForm**, stanowiący formularz zwrotu książki.
- Funkcja-kontroler *register_return()*, obsługująca zwrot książki.
- Funkcja-kontroler *register_rent()*, obsługująca wypożyczenie książki.

Obiekt **BookRentForm** oparty jest o model **BookRent**. Istotnie, ponieważ nowy rekord wypożyczeń odpowiada faktowi wypożyczenia książki takie podejście jest zasadne. Pominięto w nim pola egzemplarza i rzeczywistej daty zwrotu, w pierwszym przypadku dlatego, że ta informacja dana jest kontrolerowi, a w drugim – ponieważ przy wypożyczaniu nie jest nam

znana rzeczywista data zwrotu. Ponieważ nie jest ustalona w kontrolerze, przyjmie wartość *NULL*.

Obiekt **BookReturnForm** nie jest formularzem opartym na modelu – jest zwykłym formularzem. Zawiera on tylko trzy pola – rzeczywista data zwrotu oraz dwie flagi – czyli egzemplarz został zagubiony oraz czy został zniszczony. Ze względu na to że nie jest to formularz oparty o model, dane które są w nim zawarte będą po prostu udostępnione kontrolerowi po skutecznej weryfikacji.

Funkcja-kontroler *register_rent()* odpowiada za rejestrację wypożyczenia. Jest stosunkowo podobna do strony dodawania egzemplarza książki. Dodatkową modyfikacją jest wstępnie wypełnione pole „oficjalnej daty zwrotu", tj. data wypożyczenia plus dwa tygodnie. Wartość ta przekazywana jest do konstruktora formularza. Po pomyślnym wypożyczeniu użytkownik przekierowywany jest do strony egzemplarza.

Funkcja-kontroler *register_return()* odpowiada za rejestrację zwrotu książki. Ponieważ jej kontekstem jest egzemplarz książki, tak więc wiadomo czy jest wypożyczony, oraz który rekord wypożyczeń odpowiada temuż wypożyczeniu. Najpierw sprawdzane jest czy książka nie już wypożyczona – ze względu na to że nie można zwrócić niewypożyczonej książki. Dalej kontroler jest dość podobny do *register_rent()*, choć posługuje się formularzem nieprzypisanym do modelu. W celu zarejestrowania zwrotu wykonywana jest metoda *close()* odpowiedniego wypożyczenia, uzyskiwana wywołaniem *get_renting()* na egzemplarzu do zwrotu.