Architektury i technologie systemów internetowych – dokumentacja techniczna „ATSI Event Shop”

Kornel Samociuk 311619

Spis treści

Rozdział 1. Zakres projektu 3
Rozdział 2. Projektowanie systemu 4
Rozdział 3. Architektura systemu 4
Moduł core 5
Plik models.py 5
Plik admin.py 5
Plik views.py 5
Plik context_processor.py 6
Plik urls.py 6
Plik forms.py 6
Moduł userauths 6
Plik models.py 6
Plik admin.py 6
Plik views.py 6
Plik urls.py 6
Plik forms.py 7
Function.js 7
Folder templates 7
Szablon partials 7
Szablony modułu core 7
Szablony modułu userauths 8
Rozdział 4. Bibliografia 8

Rozdział 1. Zakres projektu

W ramach przedmiotu należało zrealizować projekt witryny internetowej obsługującej zakup biletów na wydarzenia kulturalne. System ten musiał obsługiwać następujące role:

- Administrator,
- Użytkownik końcowy zalogowany,
- Użytkownik końcowy niezalogowany,

oraz następujące wymagania:

- Administrator powinien móc wprowadzać do sprzedaży pule biletów na określone wydarzenie kulturalne.
- Administrator może definiować listę kategorii wydarzeń.
- Każda pula biletów na określone wydarzenie kulturalne jest definiowana przez następujący minimalny zestaw parametrów: nazwa wydarzenia, kategoria wydarzenia, szczegółowy opis wydarzenia, miejsce wydarzenia, data wydarzenia, liczba dostępnych biletów, cena pojedynczego biletu, daty od-do dostępności biletów do sprzedaży - uwaga: daty sprzedaży muszą być przed datą wydarzenia.
- Użytkownik końcowy widzi w witrynie listę aktualnie sprzedawanych pul biletów z informacją, czy nie jest ona już wyczerpana. Informacja o dostępności jest automatycznie aktualizowana na bieżąco.
- Użytkownik końcowy może wyświetlić szczegóły każdego wydarzenia.
- Użytkownik końcowy może zakupić bilety klikając przycisk 'Kup'. Przycisk ten jest dostępny tylko dla pul biletów niewyczerpanych, zarówno na stronie z widokiem listy wydarzeń, jak i na stronie przedstawiającej szczegóły wydarzenia. Dla wyczerpanych pul biletów widoczna jest tylko informacja o braku biletów.
- Po wybraniu przycisku Kup użytkownik może wprowadzić swoje dane oraz liczbę kupowanych biletów (domyślnie 1, ma możliwość ich zmiany). Dane użytkownika zalogowanego wypełniane są automatycznie. Użytkownik niezalogowany w procesie zakupu powinien być zachęcany do zarejestrowania/zalogowania się do witryny. Wraz ze zmianą liczby biletów Użytkownik widzi ich sumaryczną cenę. Zakup jest dokonywany poprzez kliknięcie przycisku 'Potwierdzam zakup'.
- Po potwierdzeniu, jeżeli zakup został zrealizowany poprawnie (np. w międzyczasie bilety nie zostały wykupione) to prezentowana jest informacja podsumowująca, w przeciwnym wypadku informacja o przyczynie nieskutecznego zakupu.
- Zalogowany użytkownik powinien mieć możliwość zobaczenia historii zakupów, tj. wszystkich biletów, które zakupił poprzez witrynę.

Rozdział 2. Projektowanie systemu

Na podstawie założeń projektu postanowiono na utworzenie aplikacji „ATSI Event Shop”, wykorzystując do tego język programowania $Python$ w wersji 3.10 [B1], framework $Django$ [B2] oraz bazę danych $SQLite$ [B3].

Powodami przez które zdecydowano się na te technologie była prostota instalacji i obsługi tych technologii oraz stosunkowo nieskomplikowane wymagania funkcjonale projektowanego systemu. Dodatkowym atutem, który wzięto pod uwagę podczas wyboru metody wykonania projektu było obsługiwanie przez $Django$ szablonów, które wykorzystuje się do klarownego oddzielania warstwy logiki aplikacji od warstwy prezentacji co pozwoliło by zapewnić wysoką przejrzystość kodu projektowanego systemu.

Rozdział 3. Architektura systemu

Systemy oparte o $Django$ składają się z wielu modułów, gdzie każdy z nich zawiera swoje własne pliki deklarujące logikę poszczególnych elementów systemu.

Aplikacja „ATSI Event Shop” składa się z dwóch takich modułów:

- $core$ – odpowiada za logikę działania sklepu,
- $userauths$ – autentykację użytkowników i tworzenie im profili w systemie.

Gdzie każdy z nich zawiera następujący zestaw plików:

- $admin.py$ – służący do konfigurowania panelu admina czyli wbudowanego interfejsu do zarządzania danymi obsługiwanego przez bibliotekę $Jazzmin$ [B4],
- $forms.py$ – służący do tworzenia formularzy dzięki którym $Django$ waliduje dane wprowadzane na stronie,
- $models.py$ – służący do definicji struktury danych w bazie danych, czyli jakie tabele i pola mają się w niej pojawić,
- $urls.py$ – służący do mapowania ścieżek URL na konkretne widoki,
- $views.py$ – służący do przetwarzanie żądań i przygotowanie odpowiedzi konkretnych widoków na stronie.

Ponadto system zawiera jeszcze istotny plik $settings.py$, folder $templates$ oraz plik $function.js$:

- $settings.py$ – plik konfiguracyjny projektu definiujący np. z jakich zewnętrznych aplikacji korzysta system oraz do jakiej strefy czasowej mają myć dostosowane dane godzinowe,
- $templates$ – folder przechowujący szablony $HTML$,
- $function.js$ – plik JavaScript obsługujące interakcje użytkownika na stronie korzystając z $Ajax$ i $jQuery$ [B5].

Moduł core

Plik models.py

Plik ten odpowiada za zdefiniowanie struktury danych dotyczących funkcjonowania sklepu w bazie danych. Niektóre z modeli mają odpowiednie metody do zarządzania wprowadzanymi danymi, np. obliczania zniżek, aktualizowania stanu magazynowego czy przechowywania obrazów produktów. Zdefiniowane klasy to:

- $Tags$ – pusta klasa, niezbędna do dynamicznej obsługi definiowanych bezpośrednio w bazie tagów przez bibliotekę $Taggit$ [B6],
- $Category$ – reprezentuje kategorię produktu,
- $Product$ – reprezentuje produkt w sklepie,
- $ProductImages$ – przechowuje dodatkowe obrazy związane z danym produktem,
- $CartOrder$ – reprezentuje zamówienie w koszyku użytkownika,
- $CartOrderProducts$ – przechowuje szczegóły produktów w koszyku,
- $ProductReview$ – reprezentuje recenzje produktu,
- $Address$ – reprezentuje adres rozliczeniowy użytkownika.

Plik admin.py

Plik ten jest odpowiedzialny za konfigurację części panelu administracyjnego odpowiedzialnej za produkty, kategorie, zamówienia i adresy użytkowników.

Plik views.py

Plik ten odpowiada za część warstwy logiki aplikacji odpowiedzialnej za działanie sklepu. Znajdują się w nim następujące funkcje:

- $index$ – wyświetla wyróżnione produkty na stronie głównej,
- $product\_list\_view$ – wyświetla listę opublikowanych produktów,
- $category\_list\_view$ – wyświetla dostępne kategorie,
- $category\_product\_list\_view$ – wyświetla produkty w wybranej kategorii,
- $product\_detail\_view$ – wyświetla szczegółowy widok produktu, w tym recenzje i powiązane produkty, czyli produkty z tej samej kategorii,
- tag_list – filtruje produkty po tagach, korzystając z biblioteki $Taggit$,
- $ajax\_add\_review$ – dodaje recenzję do produktu za pomocą $Ajaxa$,
- $search\_view$ – odpowiada za wyszukiwanie produktów na podstawie zapytania użytkownika,
- add_to_cart – dodaje produkt do koszyka przechowywanego w sesji, zwracając dane o koszyku,
- $cart\_view$ – wyświetla zawartość koszyka z całkowitą kwotą za wszystkie dodane do niego produkty,
- $delete\_item\_from\_cart$ – usuwa produkt z koszyka,
- $update\_cart$ – aktualizuje ilość produktów w koszyku,
- $checkout\_view$ – wyświetla podsumowanie zamówienia i umożliwia płatność przez $PayPal$, który został zintegrowany przez odpowiednią bibliotekę [B7],
- $payment\_completed\_view$ – obsługuje zakończenie płatności i aktualizuje status zamówienia,
- $payment\_failed\_view$ – obsługuje nieudane płatności,
- $customer\_dashboard$ – obsługuje stronę profilu użytkownika, gdzie ten może edytować swoje dane,

- $order\_detail$ – wyświetla szczegóły dokonanych przez użytkownika zamówień,
- $make\_address\_default$ – ustawia wybrany przez użytkownika adres jako domyślny.

Plik context_processor.py

Plik ten zawiera funkcję, która jest używana do pobierania danych potrzebnych do wyświetlania szablonów aplikacji.

Plik urls.py

Plik ten konfiguruje URL-e modułu core, przypisując konkretne ścieżki URL do odpowiednich widoków zdefiniowanych w views.py.

Plik forms.py

Plik ten zawiera definicję formularza używanego do dodawania recenzji produktów.

Moduł userauths

Plik models.py

Plik ten odpowiada za zdefiniowanie struktury danych dotyczących użytkowników sklepu w bazie danych. Zdefiniowane klasy to:

- $User$ – reprezentuje użytkownika systemu, jednocześnie rozszerzając domyślny model użytkownika $Django$ o dodatkowe pola,
- $Profile$ – reprezentuje profil użytkownika.

Ponad to zdefiniowane są tu dwie funkcje:

- $create\_user\_profile$ – jest to sygnał wywoływany po utworzeniu nowego użytkownika, tworzący mu pusty profil w systemie,
- $save\_user\_profile$ – jest to sygnał wywoływany za każdym razem, gdy użytkownik wywołuje zapis swojego profilu.

Plik admin.py

Plik ten jest odpowiedzialny za konfigurację części panelu administracyjnego odpowiedzialnej za zarejestrowanych w systemie użytkowników oraz ich profili.

Plik views.py

Plik ten odpowiada za część warstwy logiki aplikacji odpowiedzialnej za rejestracje, logowanie, wylogowywanie i uaktualnianie profilu użytkownika . Znajdują się w nim następujące funkcje:

- $register\_view$ – obsługuje rejestrację nowego użytkownika,
- $login\_view$ – obsługuje logowanie zarejestrowanego użytkownika,
- $logout\_view$ – obsługuje wylogowanie użytkownika,
- $profile\_update$ – obsługuje aktualizację profilu użytkownika.

Plik urls.py

Plik ten konfiguruje URL-e modułu userauths, przypisując konkretne ścieżki URL do odpowiednich widoków zdefiniowanych w views.py.

Plik forms.py

Plik ten zawiera definicję dwóch formularzy, które są używane do rejestracji użytkownika i aktualizacji profilu.

Function.js

Skrypt ten zapewnia dynamiczną interakcję użytkownika ze stroną w sposób asynchroniczny. Implementacja zawartych tu klas opiera się o technologię $Ajax$.

Zdefiniowane w tym pliku funkcje to:

- $commentForm$ – wysyła dane recenzji produktu do serwera, by po zapisaniu recenzji w bazie danych wyświetlić nową recenzję na stronie,
- $add-to-cart-btn$ – dodaje produkt do koszyka, a następnie zmienia ikonę guzika i aktualizuję liczbę produktów w koszyku użytkownika,
- $delete-product$ – usuwa produkt z koszyka na serwerze i odświeża zawartość koszyka na stronie,
- $update-product$ – wysyła nową ilość produktu na serwer i po jego aktualizacji odświeża zawartość koszyka,
- $make-default-address$ – wysyła żądanie do serwera, aby zaktualizować status wybranego adresu, a następnie aktualizuje interfejs strony.

Folder templates

W folderze tym przechowywane są szablony odpowiedzialne za warstwę prezentacji aplikacji. Szablony podzielone są zgodnie z modułami systemu oraz na jeden szablon częściowy. Zdefiniowane szablony to:

Szablon partials

- $base.html$ – szablon który doładowywany jest jako część każdego innego szablonu. Zawiera on uniwersalny nagłówek oraz stopkę strony.

Szablony modułu core

- $cart.html$ – szablon odpowiedzialny za wygląd strony koszyka zakupowego,
- $cart-list.html$ – szablon asynchronicznie doładowywany do szablonu $cart.html$ w celu aktualizacji listy produktów w koszyku,
- $category-list.html$ – szablon odpowiedzialny za wygląd strony kategorii; wyświetlają się w nim wszystkie dostępne kategorie,
- $category-product-list.html$ – szablon odpowiedzialny za wygląd strony wybranej kategorii; wyświetlają się w nim wszystkie produkty danej kategorii,
- $checkout.html$ – szablon odpowiedzialny za wygląd strony podsumowania zamówienia,
- $dashboard.html$ – szablon odpowiedzialny za wygląd strony konta użytkownika; może on tu sprawdzać historię zakupów, zmieniać dane profilowe czy wprowadzać nowy adres rozliczeniowy
- $index.html$ – szablon odpowiedzialny za wygląd strony głównej, wyświetlający wszystkie produkty oznaczone jako *featured*

- $order - detail.html$ – szablon odpowiedzialny za wygląd strony szczegółów zamówienia z historii zamówień użytkownika,
- $payment - completed.html$ – szablon odpowiedzialny za wygląd strony potwierdzającej sprzedaż produktu; użytkownik ma tu możliwość wydrukowania lub pobrania na dysk potwierdzenia zamówienia,
- $payment - failed.html$ – szablon odpowiedzialny za wygląd strony informującej, że sprzedaż produktu się nie powiodła,
- $product - detail.html$ – szablon odpowiedzialny za wygląd strony szczegółów danego produktu; wyświetlają się tu m.in. szczegółowe opisy i dane produktu, jego recenzje czy oferty powiązanych produktów,
- $product - list.html$ – szablon odpowiedzialny za wygląd strony wyświetlającej wszystkie dostępne produkty,
- $search.html$ – szablon odpowiedzialny za wygląd strony wyszukiwania danego produktu po jego nazwie,
- $tag.html$ – szablon odpowiedzialny za wygląd strony wyszukiwania produktów po ich tagach.

Szablony modułu userauths

- $profile.edit.html$ – szablon odpowiedzialny za wygląd strony edycji danych profilu użytkownika,
- $sign - in.html$ – szablon odpowiedzialny za wygląd strony logowania,
- $sign - up.html$ – szablon odpowiedzialny za wygląd strony rejestracji użytkownika.

Rozdział 4. Bibliografia

B1 – dokumentacja $Pythona$ - https://www.python.org/doc/

B2 – dokumentacja $Django$ - https://www.djangoproject.com

B3 – dokumentacja $SQLite$ - https://docs.python.org/3/library/sqlite3.html

B4 – dokumentacja $Jazzmina$ - https://django-jazzmin.readthedocs.io

B5 – dokumentacja $Ajaxa$ i $jQuery$ - https://api.jquery.com/category/ajax/

B6 – dokumentacja $Taggit$ - https://django-taggit.readthedocs.io/en/latest/

B7 – dokumentacja $Paypala$ - https://developer.paypal.com/braintree/docs/guides/paypal/server-side/python