Житомирський державний університет імені Івана Франка
Кафедра прикладної математики та інформатики

ДИПЛОМНА РОБОТА
з інформатики на тему:
“***Розробка студентського сайту фізико-математичного факультету засобами Django Framework***”

магістрантки VII курсу
напряму підготовки
8.04030201 Інформатика*
денного відділення
Куліковської Оксани Володимирівни
Керівник: Спірін Олег Михайлович,
доктор педагогічних наук,
професор кафедри прикладної
математики та інформатики

Житомир 2017

Зміст

Вступ

Щодня в мережі Інтернет з'являється велика кількість різних сайтів. За допомогою таких сайтів люди можуть дізнаватися безліч цікавих та корисних звісток, не виходячи з дому. Інтернет – це один з інструментів швидкого розповсюдження даних. Тому клієнти мають цілодобово володіти найсвіжішим матеріалом. Важливо, щоб на сайті містилися відомості певної тематики для необмеженого, різнорідного кола людей так, щоб їм було зручно, але в той же час сайт має бути вигідним для власників. Імідж складає стійкий образ фірми, викликані асоціації, емоційне ставлення. Зворотній зв'язок сайту надає можливість знати настрої клієнтів, враження про продукцію, про якість послуг, що надаються через, наприклад, гостьову книгу.

Студентство, як і всі ланки людства, являє собою окрему спілку і потребує взаємодії у межах групи, факультету, інституту тощо. Для задоволення їх потреб варто створити студентський сайт. Студентський сайт фізико-математичного факультету має функціонувати як повноцінна система викладу звісток, оголошень чи інших важливих даних, що має вміщувати анонсування подій або новин факультету та університету, умови вступу та навчання, відомості про викладацький склад, повідомлення для випускників, опис проектів, над якими працює факультет, огляд додаткових послуг, що надає фізико-математичний факультет, навчальний заклад (клубів, гуртків, студій) тощо.

Подібний сайт, а саме сайт фізико-математичного факультету Тернопільського національного педагогічного університету є основою для нашої розробки. Його можна знайти за посиланням: https://www.fizmat.tnpu.edu.ua/. На сайті можна знайти відомості про факультет, кафедри, інноваційні проекти, новини тощо.

Об’єктом дослідження є студентський сайт фізико-математичного факультету.

Предмет дослідження – розробка студентського сайту фізико-математичного факультету.

Метою роботи є дослідження можливостей створення студентського сайту фізико-математичного факультету засобами Django Framework.

Необхідно визначити головні інструменти створення такого сайту. А саме, потрібно реалізувати такі завдання:

- дослідити особливості різних видів фреймворків;
- описати використання Django в проекті;
- сформувати структуру сайту;
- розробити сайт фізико-математичного факультету.

Розділ 1. Дослідження роботи з фреймворками

1.1. Поняття фреймворку

Фреймворк (Framework, каркас, платформа, структура, інфраструктура) – програмна платформа, що визначає структуру програмної системи або інфраструктура програмних рішень, що полегшує розробку складних систем [1]. У комп'ютерному програмуванні, програмна платформа являє собою абстракцію, в якій програмне забезпечення, що забезпечує загальну функціональність може бути вибірково змінена додатково написаним користувачем кодом. Фреймворк є універсальним середовищем програмування, що забезпечує певну функціональність, як частина більш великої програмної платформи для полегшення розробки програмних додатків, продуктів і рішень. Фреймвоки можуть включати в себе програми підтримки, компілятори, бібліотеки коду, набори інструментальних засобів, і інтерфейси прикладного програмування (API), які об'єднують всі різні компоненти, що забезпечують розробку проекту або системи.

Фреймвоки мають основні відмінні риси, що відокремлюють їх від звичайних бібліотек:

- інверсія контролю: у фреймворці, на відміну від бібліотек або в стандартних користувальницьких додатків, потік загальної програми керування викликається не користувачем, а фреймворком;
- розширюваність: користувач може розширити структуру, як правило, шляхом селективного перевизначення; або програмісти можуть додати спеціалізований код користувача, щоб забезпечити певну функціональність;
- немодифікований фреймворком код: загалом, код не передбачається модифікувати, приймаючи користувачем реалізовані розширення. Іншими словами, користувачі можуть розширити фреймворк, але не повинні змінювати його код; фреймворк дозволяє спростити процес розробки складного високотехнологічного проекту, для досягнення таких завдань фреймворк надає наступні можливості:

- мовна сумісність: додатки, написані на різних мовах програмування, можуть використовувати фрагменти коду один одного. Більш того, частини однієї програми можна написати на різних мовах і при цьому вони будуть «розуміти» один одного і активно взаємодіяти;
- сумісність з операційною системою: користувачі можуть працювати на будь-якій операційній системі, яку вони забажають встановити;
- універсальний призначений для користувача інтерфейс;
- стійкість коду: платформа активно забезпечує безпеку додатків, реалізуючи ряд дуже важливих механізмів;
- сумісність і повторно використовуваний код;
- кросплатформність: фреймворк дозволяє створювати програми, що виконуються на різних програмних і технічних платформах [1].

Більшість фреймворків розраховані на вирішення конкретних завдань. Зрозуміло, що кожен фреймворк має свої особливості роботи з ним, які потрібно розуміти перед початком розробки проекту.

1.2. Різновиди фреймворків

Розвиток технологій створення сайтів посприяв появі великої кількості фреймворків. Їх пожна поділити на категорії, залежно від мови, яка використовується.

Популярні php-фреймворки:

Yii — це високоефективний, заснований на компонентній структурі PHP-фреймворк, що підходить для розробки великих веб-додатків. Він дозволяє максимально застосувати концепцію повторного використання коду та може істотно прискорити процес веб-розробки. Таким чином, він підходить для створення всіх видів веб-додатків: форумів, порталів, систем управлінь контенту, сайтів електронної комерції тощо. Для розробників, які бажають використовувати Yii, вкрай корисним буде розуміння концепції об єктно орієнтованого програмування (ООП), так як Yii — це строго об єктно орієнтований фреймворк [3]. Виділимо особливості:

- висока продуктивність щодо інших фреймворків написаних на PHP;

- парадигма MVC (Model View Controller);
- інтерфейси DAO і ActiveRecord для роботи з базами даних (PDO);
- кешування сторінок і окремих фрагментів;
- підтримка інтернаціоналізації;
- введення та валідація форм;
- перехоплення і обробка помилок;
- використання AJAX і інтеграція з jQuery;
- аутентифікація і авторизація;
- підтримка тем оформлення та їх легкої зміни;
- можливість підключення сторонніх бібліотек;
- відстежувати зміни в структурі бази даних (міграції бази даних);
- автоматичне тестування;
- підтримка REST (Representational state transfer – передача стану уявлення) [5].

Symfony являє собою набір повторно використовуваних компонентів PHP, що дозволяє розробникам створювати масштабовані, високопродуктивні додатки. Серед 30 компонентів розробник має повну свободу експериментувати і працювати в середовищі RAD (rapid application development - швидка розробка застосунків). Symfony API-інтерфейси дозволяють також легко інтегруватися з додатками сторонніх розробників, і він може бути використаний з популярними front-end фреймворками, такими як AngularJS [5].

Zend – інструмент для створення професійних проектів на PHP. Він розроблений самими творцями мови PHP і використовується не тільки в "домашніх" цілях, але і при розробці великих проектів. Даний фреймворк заснований на PHP, являється об’єктно-орієнтованим, використовує парадигму MVC. Крім MVC-компонентів містить безліч бібліотек, корисних для побудови програми, наприклад, реалізовані компоненти для інтеграції зі сторонніми сервісами. Характерні наступні особливості:

- всі компоненти написані на повністю об'єктно-орієнтованому коді PHP 5 та E_STRICT-сумісні;

- архітектура «слабкого зв'язування» з мінімальними залежностями між частинами проекту (англ. use-at-will architecture with loosely coupled components and minimal interdependencies);
- розширювана реалізація MVC, за замовчуванням підтримує макети і PHP-шаблони;
- підтримка різних СУБД, включаючи MariaDB, MySQL, Oracle Database, IBM DB2, Microsoft SQL Server, PostgreSQL, SQLite і Informix;
- формування, відправлення та отримання поштових повідомлень за протоколами mbox, Maildir, POP3 і IMAP4;
- гнучка система кешування з підтримкою різних типів - в пам'яті або в файловій системі [5].

Kohana – це веб-фреймворк з відкритим кодом, заснований на PHP5 і використовує концепцію HMVC (Hierarchical Model View Controller - Ієрархічні Модель-Вид-Контролер). Швидка розробка додатків забезпечена наявністю багатьох основних компонентів: інструменти перекладу, доступ до бази даних, шифрування, валідація тощо. Можна використовувати специфічні бібліотеки та інструменти, легко розширюючи існуючі компоненти та додавши нові бібліотеки. Фреймворк ефективно та ретельно оптимізований для використання. Прості та ефективні інструменти допомагають виявляти і швидко вирішувати проблеми продуктивності. Дуже добре документований код і проста структура маршрутизації дозволяє легко зрозуміти, що відбувається [4].

CodeIgniter – фреймворк з відкритим вихідним кодом написаний на PHP. Використовується для розробки повноцінних веб-ситем та програм. Переваги:

- дозволяє не морочитися на дрібницях, і зосередитися на логіці проекту;
- економить час при розробці проекту;
- всі проекти мають одну і ту ж структуру файлів, що дозволяє краще орієнтуватися в коді;
- грамотно реалізовані функції;

- можливість розширення. Якщо потрібна реалізація якийсь функції, висока ймовірність знайти потрібний плагін до codeigniter'у на просторах інтернету. Так само якщо є своя реалізація потрібної функції, яка постійно використовується у своїх проектах, нічого не заважає оформити її як плагін [6].

Найбільш популярні ruby-фреймворки:

Ruby on Rails (RoR) – це багаторівневий MVC-фреймворк для побудови веб-додатків, що використовують реляційні і NoSQL бази даних (наприклад, MySQL, MariaDB, PostgeSQL, MongoDB). Фреймворк написаний на мові програмування Ruby. Rails підходить як для розробки звичайних сайтів, які повинні бути реально швидкими, відмовостійкими і працюють під високим навантаженням, так і для веб-додатків зі складною бізнес-логікою і динамічними web-інтерфейсами. Ruby on Rails є відкритим програмним забезпеченням і розповсюджується під ліцензією MIT. Основними принципами розробки на Rails є:

- принцип DRY (Do not repeat yourself) – фреймворк надає механізми повторного використання програмного коду, це дозволяє не тільки мінімізувати дублювання коду, але і підвищити швидкість розробки;
- принцип Convention over configuration – за замовчуванням у фреймворку використовуються численні угоди по конфігурації, типові для більшості додатків; це дуже спрощує створення додатків, так як явна специфікація конфігурації потрібно тільки в нестандартних випадках;
- автоматизоване тестування – в складі RoR поставляються засоби для проведення повністю автоматичного модульного, інтеграційного і функціонального тестування, а ідеологія Ruby on Rails передбачає використання методів розробки через тестування (TDD - Test Driven Development). Все це робить розроблені додатки реально надійними [2].

Sinatra – маленький, але досить цікавий DSL (Domain-specific language) фреймворк, написаний на Ruby. На відміну від Ruby on Rails він не наслідує типовий паттерн MVC. Sinatra створювався для того, щоб програміст зміг швидко створити веб додаток, написаний на Ruby з мінімальними зусиллями.

Padrino – фреймворк з відкритим вихідним кодом, написаний на Ruby та заснований на фреймворці Sinatra. Його структура намагається зробити швидку розробку простих веб-додатків та з мінімальними зусиллями. Основні функції:

- повна підтримка популярних бібліотек;
- можливість створення проектів, контролерів, моделей, міграцій тощо через padrino-gen;
- на відміну від інших ruby-фреймворків, спроектований з урахуванням можливості монтування підпроектів;
- можливість розсилати листи (аналог ActionMailer);
- вбудована адмінка;
- уніфікований інтерфейс для логізації (logining), з можливістю взаємодії з ORM або будь-якою бібліотекою, що використовується [5].

Популярні python-фреймворки:

Django – фреймворк для веб-додатків на мові Python. Один з основних принципів фреймворка – DRY (don't repeat yourself). Веб-системи на Django будуються з одного або декількох додатків, які рекомендується робити віддаленими і підключеними. Це одне з помітних архітектурних відмінностей цього фреймворка від деяких інших. Також обробники URL в Django конфігуруються явно (за допомогою регулярних виразів), а не автоматично задаються зі структури контролерів. Деякі можливості Django:

- ORM, API доступу до БД з підтримкою транзакцій;
- вбудований інтерфейс адміністратора, з уже наявними перекладами багатьма мовами;
- URL-диспетчер на основі регулярних виразів;
- розширювана система шаблонів з тегами і спадкуванням;
- система кешування;
- підключається архітектура додатків, які можна встановлювати на будь-які Django-сайти;
- авторизація та аутентифікація, підключення зовнішніх модулів аутентифікації: LDAP, OpenID та інші;

- система фільтрів («проміжного шару») для побудови додаткових обробників запитів, як наприклад включені в дистрибутив фільтри для кешування, стиснення, URL нормалізації і підтримки анонімних сесій;
- бібліотека для роботи з формами (успадкування, побудова форм по існуючої моделі БД);
- вбудована автоматична документація по тегам шаблонів і моделей даних, доступна через адміністративне додаток [5].

Flask – мікрофреймворк для створення веб-додатків на мові програмування Python, що використовує набір інструментів Werkzeug (WSGI utility library for Python), а також шаблонизатор Jinja2. Flask називається мікрофреймворк, так як він не вимагає особливих інструментів або бібліотек. Він не має налаштувань бази даних, форм перевірки, або будь-яких інших компонентів, де раніше існуючі сторонні бібліотеки забезпечують загальні функції. Проте, Flask підтримує розширення, які можуть додати функції, ніби вони були реалізовані в самій Flask. Існують розширення для об'єктно-реляційної моделі, валідації форм, обробка завантаження, різні технології аутентифікації і декількох взаємопов'язаних інструментів загальної структури. Розширення оновлюються набагато частіше, ніж в основній програмі Flask. Особливості:

- містить сервер розробки і відладчик;
- інтегрована підтримка для модульного тестування;
- RESTful запит;
- Підтримка захищених куків (на стороні клієнта сеансів);
- 100% WSGI 1.0 сумісний;
- на основі Unicode;
- сумісність з Google App Engine;
- доступні розширення для підвищення бажаних можливостей [7].

Twisted – це подієво-орієнтований мережевий фреймворк, написаний на Python і розповсюджується під ліцензією MIT (вільне програмне забезпечення). Twisted є платформою для розробки інтернет-додатків. У той час як Python сама

по собі є дуже потужною мовою, є багато послуг, які відсутні в інших мовах. Twisted є хорошим (серед своєрідних) чисто Python-фреймворком, в залежності від того, як ставитися до неї, і вона продовжує розвиватися.

В якості платформи, Twisted повинна бути спрямована на об’єднання. В ідеалі, всі функціональні можливості будуть доступні через всі протоколи. В іншому випадку, всі функції повинні бути налаштованим принаймні одним протоколом, з цільним і послідовним користувацьким інтерфейсом. Наступним етапом розвитку буде зосереджена увага на конфігурації системи, яка об'єднає безліч розрізнених частин існуючої інфраструктури [8].

Tornado – веб-фреймворк, асинхронна бібліотека мережі. При використанні незаблокованої мережі введення/виведення, Tornado може масштабуватися до десятків тисяч відкритих з'єднань, що робить його ідеальним для тривалих запитів, веб-сокетів та інших додатків, що вимагають довготривале з'єднання з кожним користувачем [9].

Pyramid – Python-фреймворк з відкритим вихідним кодом. Його основна мета полягає в забезпеченні простоти та легкості у використанні для розробників Python у процесі створення веб-додатків. Pyramid наслідує такі принципи проектування та розробки:

- простота – навіть при малому розумінні Pyramid можна отримати результат; тобто не треба використовувати специфічну технологію для відтворення додатку (проекту), достатньо знати лише основний набір концепцій;
- мінімалізм – Pyramid намагається вирішувати лише базові проблеми створення веб-додатку;
- документіція – жоден аспект Pyramid не задокументований (слідує з концепції мінімалізму);
- швидкість – Pyramid створений для забезпечення помітно швидкого виконання загальних завдань, наприклад, шаблон чи проста генерація відповіді;
- надійність – фреймворк розроблений консервативно на вичерпно протестований;

- відкритість – як і Python, програмне забезпечення Pyramid поширюється під вільною ліцензією з відкритим вихідним кодом [10].

Популярні java-фреймворки:

Spring MVC – фреймворк, метою якого є підтримка в Spring архітектури MVC. Spring забезпечує готові компоненти, які можуть бути використані (і використовуються) для розробки веб-додатків. В Spring MVC можна використовувати будь-який об'єкт в якості команди або об'єкта. Тут не потрібно використовувати специфічний для фреймворку інтерфейс або базовий клас. Spring має дуже гнучке зв'язування даних [11].

JavaServer Faces (JSF) – це фреймворк для веб-додатків, написаний на Java. Він служить для полегшення розробки користувацьких інтерфейсів для Java застосунків. На відміну від більшості MVC фреймворків, які керуються запитами, підхід JSF ґрунтується на використанні компонентів [12].

Google Web Toolkit (GWT) – вільний Java-фреймворк, який дозволяє веб-розробникам створювати Ajax-додатки. Використовуючи GWT, програмісти можуть розробляти та налагоджувати Ajax-додатки на мові Java з використанням засобів Java-розробки за своїм вибором. Коли додаток завершений, GWT компілятор переводить Java додаток в самостійні JavaScript файли, які не обов'язково достатньо оптимізовані. При необхідності, в JavaScript також може бути доданий в Java код, використовуючи Java коментарі [12].

Фреймворки від Microsoft: ASP.NET (ASP.NET MVC) і Net Framework [2].

ASP.NET – платформа, яку використовують мільйони розробників для створення додатків та сервісів на будь-який пристрій чи операційну систему. ASP.NET спирається на багатомовні можливості .NET, що дозволяє писати код сторінок на C#, VB, C/C++ та ін.

.NET дозволяє розробляти високоякісні, професійні додатки. У поєднанні з великими класами бібліотек і потужними засобами, представленими сімейством Visual Studio, .NET є найбільш продуктивною платформою для розробників [13].

1.3. Особливості Django

Django – один з найефективніших сучасних фреймворків для розробки веб-проектів. Складно придумати щось, що дозволяє так само швидко реалізувати портал або контентний проект. Причина подібної ефективності – чіткий механізм роботи з проектом, зручна ORM (object-relational mapping, об'єктно-реляційна проекція) – система моделей для спрощення використання бази даних, вбудований функціональний движок шаблонів [14]. Django включає в себе все необхідне для створення сайту. Має засоби взаємодії з базами даних, інструменти налагодження, адміністративний сайт, який можна використовувати для роботи зі збереженими в базі даними, а також веб-сервер.

Перед використанням Django варто познайомитися з деякими принципами та термінами.

Проект – сукупність веб-додатків, що становлять один Веб-сайт. Іншими словами, проект - це сам веб-сайт, розроблений із застосування Django. Фізично проект представляє собою звичайну папку операційної системи.

Структура папок і файлів проекту, яка формується при його створенні самої Django, така:

<ІМ'Я ПРОЕКТУ>

manage.py

<ІМ'Я ПРОЕКТУ>

__init__.py

settings.py

urls.py

wsgi.py

В зовнішній пвпці, де зберігаються файли проекту знаходиться файл manage.py. Цей файл зберігає код утиліти, яка дозволить нам виконувати різні дії над проектом, зокрема, створювати в ньому додатки.

В папці проекту присутня ще одна папка з таким же ім'ям. Це пакет проекту – він містить модулі, які стосуються самого проекту. Про те, що це саме пакет, говорить файл __init__.py, що зберігається в цій папці.

У пакеті проекту ми бачимо наступні модулі:

- settings – зберігає настройки проекту у вигляді набору змінних;
- urls – зберігає відомості про прив'язку додатків до інтернет-адрес;
- wsgi – службовий модуль, який виступає «посередником» між веб-сервером і проектом.

В налаштуваннях проекту зазначаються відомості про використовувану в додатках проекту базу даних (або базах даних – ми можемо вказати їх кілька), список активних додатків проекту, мовні параметри та деякі інші дані. Відзначимо, що всі ці настройки поділяються усіма додатками, що входять в проект. Папка проекту може перебувати в будь-якому місці файлової системи комп'ютера. Так що ми можемо створити проект там, де нам зручно.

Додаток Python входить до складу проекту і реалізує функціональність одного з розділів сайту і всіх його підрозділів. Кількість додатків в проекті не обмежена. Фізично додаток являє собою пакет, папка якого знаходиться в папці проекту. Ім'я цього пакета стане ім'ям програми, а сам пакет називається пакетом додатку.

Пакет додатку формується самою Django при створенні програми. Спочатку він містить наступні модулі:

- models – зберігає код моделей, що входять до складу програми;
- views – зберігає код контролерів, що входять до складу програми;
- admin – зберігає код, що задає параметри адміністративного додатка, що входить до складу Django;
- tests – тестовий модуль.

Зрозуміло, ми можемо, якщо виникне така необхідність, створити в пакеті додатки інші модулі, що зберігають код моделей, контролерів або додаткових функцій і класів, що ми задіємо в коді сайту. Django в цьому плані ніяк нас не обмежує. Ось тільки всі шаблони, які застосовуються в сайті, треба створити вручну.

Навіть якщо додаток входить до складу проекту, це ще не говорить про те, що він буде задіяний в сайті. Щоб додаток успішно працював, слід, по-

перше, виконати його прив'язку до інтернет-адреси, а по-друге, вказати його в списку активних додатків, що знаходиться в модулі settings пакета проекту. Тільки після цього додаток стане активним.

Потрібно сказати, що частина допоміжних модулів Django реалізована також у вигляді додатків (вбудовані додатки). Таким вбудованим додатком є, зокрема, підсистема, що реалізує розмежування доступу. Адміністративний сайт, за допомогою якого ми можемо працювати зі збереженими в базі даними, також є додатком подібного роду. Вбудовані додатки або також прив'язуються до інтернет-адреси та, тим самим, формують новий розділ сайту, або працюють постійно, забезпечуючи допоміжну функціональність. У будь-якому випадку їх також потрібно вказати в списку активних додатків, інакше вони не будуть задіяні.

Прив'язка інтернет-адрес. Ми знаємо, що кожен додаток сайту, запускається у відповідь на звернення до певної інтернет-адреси, до якої він був прив'язаний. Єдиний виняток тут – вбудовані додатки, з якими ми тільки що познайомилися і які працюють постійно і не вимагають такої прив'язки.

У бібліотеці Django прив'язка інтернет-адреси до додатка виконується в модулі urls пакета проекту. Або, кажучи іншими словами, прив'язка адрес до додатків виконується на рівні проекту. Але в реальності один додаток може виконувати відразу кілька дій. Скажімо, додаток списку товарів може виводити як і сам цей список, так і відомості про обраний товар, а додаток гостьової книги – як виводити гостьову книгу, так і додавати в неї новий запис. Як це реалізувати? Дуже просто. Окремим контролерам додатків ставляться у відповідність віртуальні папки згаданих раніше папок, що формують підрозділи даних розділів сайту.

При прив'язці інтернет-адреси до контролера ми можемо вказати, що останній повинен приймати будь-які дані. Ці дані будуть передані в складі інтернет-адреси із застосуванням методу GET. Наприклад, оскільки ми збираємося реалізувати висновок відомостей про обраний товар, нам доведеться передавати відповідним контролеру ідентифікатор цього товару.

Структура Django-сайту. Як ми вже знаємо, додаток Django реалізує функціональність одного розділу сайту і всіх його підрозділів. Це досягається тим, що до інтернет-адреси прив'язується не сам додаток, а його контролер.

У зв'язку з цим можна сформулювати правила структурування сайтів, написаних на Django:

- один розділ сайту реалізується одним додатком. У число таких розділів входить і головна сторінка сайту;
- один підрозділ розділу сайту реалізується одним контролером, що входять до складу програми;
- по можливості один додаток не повинен реалізовувати функціональність, що відноситься до іншого розділу сайту. Або, кажучи іншими словами, один додаток не повинен вторгатися в справи іншого;
- однак адміністративні завдання (наприклад, наповнення сайту), по можливості, повинні виконуватися окремим додатком, що носить назву адміністративного.

Дотримуватися цих правил нескладно, благо сам Python з його незалежністю окремих модулів і пакетів один від одного підштовхує нас до цього. Разом з тим, якщо нам знадобиться в одному додатку використовувати, скажімо, модуль з іншої програми, ми завжди зможемо це зробити, виконавши операцію імпорту.

Формати баз даних. Django для зберігання даних дозволяє використовувати бази різних форматів. Офіційно, самими розробниками цієї бібліотеки заявлена підтримка PostgreSQL, MySQL і Oracle, доступні також і сторонні бібліотеки, розроблені третіми програмістами і забезпечують підтримку інших форматів баз даних. Крім цього, Django підтримує формат баз даних SQLite. СУБД, що працює з такими базами даних, вбудована в сам Python, так що ніяких додаткових програм і бібліотек для реалізації її роботи нам встановлювати не доведеться. А для потреб розробки сайтів її можливостей цілком достатньо – якщо ж буде потрібно, ми легко зможемо переналаштувати сайт на роботу з базою іншого формату, того ж MySQL.

Бази даних SQLite мають і іншу перевагу – нам не доведеться створювати їх самим. Потрібно буде лише занести відомості про базу даних в налаштування сайту, створити моделі і виконати просту команду синхронізації, після чого Django сама створить базу, а в ній – всі необхідні таблиці, індекси і зв'язки.

Налагоджувальний Web-сервер Django. В процесі розробки Web-сайту нам доведеться неодноразово запускати його, щоб перевірити, як він працює і чи працює взагалі. Зрозуміло, що для цього потрібен Web-сервер, наприклад, популярний Apache. Розробники сайтів, які застосовують інші мови і платформи (PHP, Zend, Yii і ін.), просто змушені встановити його на комп'ютер і відповідно налаштувати.

Але, оскільки ми використовуємо мову Python і бібліотеку Django, нам не потрібен окремий Web-сервер. До складу самої цієї бібліотеки входить оцінний Web-сервер, який встановлюється разом з нею і не вимагає ніякого налаштування. Запустити його можна виконанням дуже простий команди. Налагоджувальний Web-сервер Django за замовчуванням прив'язаний до TCP-порту під номером 8000. Тому, щоб запустити сайт на виконання, ми повинні набрати в Web-браузері інтернет-адресу виду: http://localhost:8000/. Втім, є можливість при запуску налагоджуваного Web-сервера вказати, щоб він працював через інший порт.

Розробники Django наполегливо не рекомендують використовувати оцінний Web-сервер для публікації готового сайту внаслідок проблем з безпекою. Тому, щоб опублікувати Django-сайт, слід застосувати повнофункціональний Web-сервер, скажімо, Apache [15].

Ми розглянули теорію Django-програмування та побудови Django-слйтів. Ми познайомилися з проектами і програмами, дізналися про прив'язку інтернет-адрес до додатків, підтримуваних форматах баз даних і налагоджуваному Web-серверу, який стане в нагоді нам при налагодженні.

Розділ 2. Розробка проекту

2.1. Організація структури сайту

Мета створення сайту – інформувати про події фізико-математичного факультету як особливу ланку системи університету, популяризувати імідж факультету, активізувати здібності студентів, збільшити кількість студентів та заохочувати абітурієнтів до вступу.

Цільовою аудиторією сайту є студенти, абітурієнти та викладачі фізико-математичного факультету. Для задоволення їх потреб необхідно якомога чіткіше визначити вміст сайту, відповідно до чого створюємо структуру сайту. Це важливо, оскільки користувачі повинні досить швидко та без труднощів знаходити потрібні вістки. Тому ми аналізуємо можливий шлях користувачів по сторінках сайту та розробляємо навігацію так, щоб вона була інтуїтивно зрозуміла кожному відвідувачу.

Будь-який сайт вміщує сторінки, розділи, підрозділи, додаткові матеріали, навігацію (посилання), розташування цих елементів і їх зв’язок між собою. З позиції розробника, структуру сайту умовно можна поділити на два рівні – логічний і фізичний. На логічному рівні структурою сайту є сукупність сторінок, що поєднуються між собою єдиним дизайном, стилем і посиланнями. На фізичному рівні структура сайту являє собою впорядкований набір файлів різного типу (HTML-сторінки, зображення, програми, мультимедійні файли) [16]. Продумана і зручна структура допомагає нам значно скоротити витрати на збір інформативної складової сайту та уникнути великої кількості помилок на етапі виробництва. Якщо проект на якомусь етапі розробки або після завершення буде переданий іншим фахівцям, то залишиться зрозумілим за рахунок оптимальної структури.

Головна сторінка сайту вміщує дані про факультет та університет. Обов’язково пишемо назву сайту, додаємо логотип. Оскільки вона зазвичай відкривається першою, розміщуємо цікаву статтю про факультет.

Новини відображаються відсортованими у зворотному порядку (першою є остання додана). У загальному списку показуємо заголовок – коротка назва,

текст – частина посту, дату публікації звістки та зображення (необов'язковий параметр). Клацнувши по заголовку відкривається детальний текст – оригінал статті.

В розділі студентам окреслюємо відомості про заходи, майбутні конференції, умови участі, посилання на них, а також важливі питання з старостату.

Про факультет – повідомлення про те, де нас можна знайти, також розміщуємо посилання на сайт університету, спільноти соціальних мереж, контакти.

Ключовий елемент повноцінного сайту – авторизація. Тут нам буде потрібно механізм аутентифікації. Система аутентифікації користувачів управляє акаунтами користувачів, групами, правами і призначеними для користувача сесіями (за допомогою cookie). Цю систему часто називають системою аутентифікації і авторизації. Для авторизації необхідно перевірити, хто є цей користувач (зазвичай, за допомогою перевірки імені та пароля по базі даних користувачів) та перевірити, що користувач авторизований для виконання певних дії (зазвичай, за допомогою перевірки по таблиці прав). Обов'язкові поля: логін, пошта, пароль двічі.

Структуру сайту на логічному рівні можна зобразити схемою (див. рис. 1).

Зовнішній вигляд сайту фізико-математичного факультету базується на загальних принципах побудови сайтів, а саме: меню, контент та підвал сторінки. В меню даємо можливість переключитись (перейти) в інший розділ чи підрозділ сайту в залежності від того, що відвідувач бажає знайти. Середину сайту займає контент – основний вміст кожної сторінки, де розміщені статті, фото, тощо. У підвалі заключна частина, де показана короткі дані про створення сайту, партнери та ін.

Оформлення виконуємо в одному стилі, меню закріплене зверху, на ньому розміщений логотип – сова (символ мудрості). Стилістика яскравого дизайну. Кольорова гама акцентує увагу на важливих елементах виразними

відтінками червоного, пурпурного, зеленого, колір фону – білий, шрифт надрукований чорним та сірим кольорами.

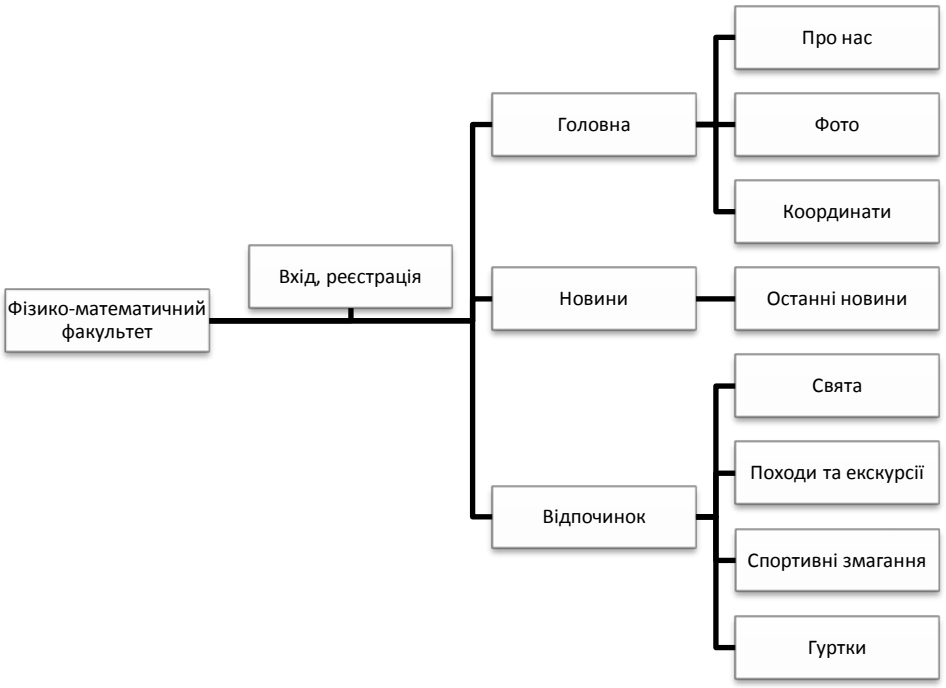


Рис. 1. Схема сайту фізико-математичного факультету

2.2. Проектування бази даних

Проектування бази даних (БД) для сайту – це один із ключових моментів його розробки. Саме даний етап закладає основи, що надалі впливають на швидкість функціонування й складність розробки всього проекту [17]. Також БД надає переваги для створення динаміки сайту. Суть полягає в тому, що сторінки формуються «на льоту» в результаті взаємодії скриптів і баз даних після відповідного запиту клієнта до веб-сервера. Для створення структури нової бази, її наповнення, редагування вмісту і відображення даних використовується комплекс програмних засобів – система управління базами даних (СУБД). Найбільш поширеними СУБД є MySQL, PostgreSQL, Oracle, Microsoft SQL Server. Зручність роботи з СУБД забезпечують спеціальні веб-додатки, які дозволяють за допомогою графічного інтерфейсу виконувати адміністрування сервера баз даних, запускати спеціальні команди, а також працювати з контентом таблиць і баз даних.

Розробимо модель бази даних студентського сайту фізико-математичного факультету. Оскільки сайт спрямований на студентство, їх інтереси та потреби, БД включатиме наступні таблиці:

- Users – Користувачі;
- Groups – Навчальні групи студентів;
- Menu_data – Назва та вміст пунктів меню;
- Categories – Категорія/тема;
- Posts – Новини та пости;
- Comments – Коментарі.

Для будь-якого сайту важливим є можливість зареєструватися на ньому. Дані реєстрації та для входу користувача будуть зберігатися в таблиці Users. На сайті студенти зможуть об'єднуватися в групи, для чого використаємо таблицю Groups. В таблиці Posts будуть зберігатися пости з новинами чи іншими важливими відомостями. В залежності від змісту посту їх будемо відносити до різних категорій, які визначимо в таблиці Categories. Надамо можливість коментувати пости. Коментарі будемо зберігати в таблиці Comments.

Варто визначити поля, що будуть міститися в таблицях. Кожна таблиця містить відомості певної тематики, а кожне поле в таблиці містить відповідні дані по темі таблиці [18]. Для кожної таблиці ми визначили ключове поле, а для кожного поля таблиці – тип даних (див табл. 1).

Таблиця 1

Поля та типи даних

Назва поля	Тип даних
Users	
id_user (ключове поле)	числовий
name	текстовий
login	текстовий
password	текстовий
group	текстовий
Groups	
id_group	числовий
name	текстовий
Menu_data	

id_m	числовий
text	текстовий
Categories	
id_category	числовий
name	текстовий
info	текстовий
Posts	
id_post	числовий
name	числовий
author	текстовий
category	текстовий
Comments	
id_com	числовий
post_id	числовий
user	числовий
text	текстовий

У процесі проектування бази даних студентського сайту ми взяли до уваги категорію користувачів, для яких розроблятиметься сайт – студенти, та їх потреби, інтереси тощо. Створені таблиці та їх зв'язки будувалися на основі забезпечення цілісності даних.

2.3. Створення сайту засобами Django Framework

Чіткий механізм роботи з проектом, зручна ORM-система моделей для спрощення використання бази даних, вбудований функціональний движок шаблонів стали причиною використання Django. Він є одним із найефективніших сучасних фреймворків для розробки веб-проектів.

Генерація проекту виконується командою

django-admin.py startproject Students

В результаті виконання команди в поточному каталозі буде створено каталог Students, в якому буде розміщено manage.py – скрипт для управління Django-проектом і каталог Students.

Створимо додаток для виконання нашого завдання.

У свою чергу, проект – набір додатків, які повністю реалізують роботу сайту. Наприклад, реалізуючи сайт великої туристичної компанії, ми можемо додати в наш проект окремий додаток, окремо – для блогу, і додатковий додаток для каталогу країн або турів.

Для початку роботи з Django ми створимо сторінку сайту, де будемо відображати новини.

Перебуваючи в тому ж каталозі, що і файл manage.py створимо додаток, який буде відповідати роботі опитувань з назвою news:

python manage.py startapp news

У новому каталозі news ми побачимо

__init__.py

models.py

tests.py

views.py

Кожна програма Django складається з пакету Python, який наслідують певні умови. Django містить команду, яка створює структуру для нового додатка, що дозволяє зосередитися на написанні коду, а не на створенні каталогів.

Для налаштування бази даних на початку розробки будемо використовувати SQLite3. Сама база при цьому зберігається в одному файлі на диску.

Для підключення бази даних до Django проекту відредагуємо в файлі news / settings.py блок DATABASES

Щоб використовувати sqlite, досить вказати django.db.backends.sqlite3 як ENGINE і прописати ім'я файлу, в якому буде зберігається база, в параметрі NAME.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Тепер створимо ваші моделі – структуру бази даних з додатковими мета-даними. Модель – це основне джерело даних. Вона містить набір полів і поведінку даних, які ми зберігаємо. Django дотримується принципу DRY, тобто

кожну окрему концепцію або частину даних слід зберігати лише в одному місці.

Також частиною роботи з даними є міграції. На відміну від Ruby On Rails, міграції винесені з файлу моделей і є просто історією, яку Django може використовувати для зміни бази даних відповідно до поточної структури моделей.

У нашому додатку ми створимо дві модель Article, що містить назву, текст, автора тексту та дату публікації. Ці поняття відображаються простим класом Python. В файлі news / models.py запишемо наступне:

```
SHORT_TEXT_LEN = 500

class Article(models.Model):
    title = models.CharField(max_length=200)
    text = models.TextField()
    user = models.ForeignKey(User)
    pub_date = models.DateTimeField('date_published')
```

Отже, кожну модель ми представляємо класом, успадкованим від django.db.models.Model. Дана модель містить декілька атрибутів, кожен з яких відображає поле в таблиці бази даних. Кожне поле визначається відповідним екземпляром класу Field – наприклад, CharField для текстових полів і DateTimeField для полів дати і часу. Це вказує Django, які типи даних зберігають поля.

Назви кожного поля Field (наприклад, title або pub_date) – це назва поля, в "машинному" (machine-friendly) форматі. Ці назви використовуються у коді, а база даних буде використовувати їх як назви колонок.

Можна використовувати перший необов'язковий аргумент конструктора класу Field, щоб визначити зручну для сприйняття назву поля. Вона використовується в деяких компонентах Django. Якщо ця назва не вказана, Django буде використовувати "машинну" назву. Ми вказали, що відображається назва тільки для поля Article.pub_date. Для всіх інших полів буде використовуватися "машинна" назва.

Деякі класи, успадковані від Field, мають обов'язкові аргументи. Наприклад, CharField вимагає, щоб йому передали max_length. Це використовується не тільки в схемі бази даних, але і при валідації.

Зауважимо, що зв'язок між моделями визначається за допомогою ForeignKey. Це вказує Django, що кожна Article пов'язана з одним об'єктом User. Django підтримує всі основні типи зв'язків: багато-до-одного, багато-до-багатьох і один-до-одного. Тепер Django знає про створений додаток.

Для зручного відображення об’єкту відредагуємо модель Article (в файлі news / models.py) і додаємо метод __str __():

```
def __str__(self):
    return self.title

def get_short_text(self):
    if len(self.text) > SHORT_TEXT_LEN:
        return self.text[:SHORT_TEXT_LEN]
    else:
        return self.text
```

Командою makemigrations [<app_label>] створюємо таблиці:

django-admin makemigrations

Створює нові міграції на основі змін у моделях.

Можна вказати додатки, для яких слід створити міграції, також будуть враховуватися пов'язані додатки (наприклад, які містять моделі, на які посилаються ForeignKey).

Синхронізує стан бази даних з поточним станом моделей і міграцій:

django-admin migrate

Поведінка команди migrate [<app_label> [<migrationname>]] залежить від зазначених аргументів:

- Без аргументів: буде виконана синхронізація всіх додатків.
- <App_label>: Будуть виконані міграції для зазначеного додатка до самої останньої міграції. Це може викликати міграцію інших додатків через залежності в міграціях.
- <App_label> <migrationname>: Приводить базу даних до стану, які були б після завершення зазначеної міграції, але наступні міграції не виконані.

Це може привести до скасування міграцій, якщо були виконані міграції, які слідують після зазначеної нами. Щоб скасувати всі міграції для додатка, використовується zero.

Команда createsuperuser працює тільки, якщо встановлена система авторизації (django.contrib.auth). Створює суперкористувача (користувач з усіма правами). Можна використовувати для створення найпершого суперкористувача, або щоб програмно створювати суперкористувача.

При запуску через консоль команда зажадає ввести пароль. Якщо виконати команду програмно, не через консоль, буде створений користувач без пароля, він не зможе авторизуватися, поки не буде встановлено пароль для нього. Встановлюємо пароль:

django-admin createsuperuser

Ім'я користувача і email можна вказати за допомогою опцій -username і -email. Якщо одна з опцій не вказана, createsuperuser попросить ввести значення, при запуску команди через консоль.

Опція -database дозволяє вказати базу даних, в якій буде створено новий користувач.

Наступний крок – додати посилання на news.urls в головній конфігурації URL-ів. В “mysite / urls.py” додамо import django.conf.urls.include, потім include() додамо в список urlpatterns. Маємо наступний код:

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls)),
    url(r'^', include('news.urls'))
)
```

Функція url() приймає чотири аргументи, два обов'язкових: regex і view, і два необов'язкових: kwargs і name.

- url() argument: regex

Термін "regex" зазвичай використовується як короткий варіант "regular expression" (регулярний вираз), що являється шаблоном для розпізнавання рядків, або в нашому випадку URL-ів. Django починає з першого регулярного

виразу і йде далі за списком, порівнюючи URL запиту з кожним регулярним виразом поки не буде знайдено збіг.

Регулярні вирази не перевіряють GET і POST аргументи або доменне ім'я. Наприклад, при запиті на https://www.example.com/myapp/, буде перевірятися myapp/. При запиті на https://www.example.com/myapp/?page=3, так само буде перевірятися myapp/.

Зауваження про продуктивність: регулярні вирази компілюються при початковому імпорті модуля з конфігурацією URL-ів. Вони працюють дуже швидко (до тих пір, поки вони досить прості).

- url () argument: view

коли Django знаходить відповідний регулярний вираз, Django викликає зазначену функцію представлення, передаючи першим аргументом об'єкт HttpRequest і всі розпізнані регулярним виразом значення як інші аргументи. Якщо регулярний вираз використовує просте розпізнавання значень – вони передаються як позиційні аргументи, якщо використовується іменоване розпізнання – значення передаються як іменовані аргументи.

- url () argument: kwargs

В уявлення можна передати зумовлені іменовані аргументи.

- url () argument: name

Додавши ім'я URL-у, ми можемо звернутися до нього по імені з будь-якої точки проекту, особливо шаблону. Такий підхід дозволяє робити глобальні зміни у налаштуваннях URL-ів, змінюючи мінімум файлів.

Функцією для обробки запиту являється уявлення, яке використовує шаблон для генерації сторінки. Уявлення – це просто функція Python (або метод уявлення-класу). Django вибирає уявлення, аналізуючи запитаний URL (точніше частина URL-а після домену).

URL-шаблон – це загальна форма URL-а. Наприклад: / newsarchive / <year> / <month> /.

Щоб з URL-а отримати уявлення, Django використовує так званий 'URLconf'. URLconf визначає відповідність URL-шаблонів (є регулярними виразами) і уявлень.

Тепер створимо ще парочку уявлень в news / views.py. Ці уявлення трохи відрізняються, так як приймають аргументи:

```
from django.http import HttpResponse
from django.shortcuts import render, get_object_or_404
from blog.models import Article

def home(request):
    template = loader.get_template('news/home.html')
    articles = Article.objects.all()
    context = {
        'articles': articles
    }
    return HttpResponse(template.render(context))
```

Цей код завантажує шаблон news / home.html і передає йому контекст. Контекст – це словник, який містить назву змінних шаблону і відповідні їм значення.

Процес завантаження шаблону, додавання контексту і повернення об'єкта HttpResponse, цілком тривіальний. Django надає функцію для всіх цих операцій. Ось як буде виглядати наш home():

```
from django.shortcuts import render, get_object_or_404
from blog.models import Article

def home(request):
    articles = Article.objects.all()
    context = {
        'articles': articles
    }
    return render(request, 'blog/news.html', context)
```

Так як ми використовуємо такий підхід у всіх наших уявленнях, немає необхідності імпортувати loader, RequestContext і HttpResponse.

Функція render() першим аргументом приймає об'єкт запиту, також назва шаблону і необов'язковий словник значень контексту. Повертає об'єкт HttpResponse містить виконаний шаблон із зазначеним контекстом.

Прив'яжемо наше уявлення новин в модулі news.urls додавши виклик url():

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
import re
urlpatterns = patterns('',
    url(r'^$', 'news.views.home', name='home'),
)
```

Створюємо каталог templates в каталозі додатку news. Django буде шукати шаблони в цьому каталозі.

Налаштування TEMPLATES вказує Django як завантажувати і виконувати шаблони. За замовчуванням використовується бекенд DjangoTemplates, з опцією APP_DIRS рівній True. В цьому випадку DjangoTemplates перевіряє підкаталог "templates" в додатках, зазначених в INSTALLED_APPS.

У щойно створеному каталозі templates, створюємо каталог news, і в ньому файл home.html. Тобто формуємо файл polls/templates/news/home.html. З огляду на як працює завантажувач шаблонів app_directories, можна звертатися до шаблону як news/home.html.

Ми б могли створити наш шаблон безпосередньо в polls/templates (а не підкаталозі туці), але це погана ідея. Django буде використовувати перший знайдений шаблон, і якщо існує шаблон з аналогічною назвою в іншому додатку, Django не зможе розрізнити їх. Щоб цього уникнути, ми будемо використовувати простір імен. Точніше, просто додамо їх в ще один підкаталог з назвою, аналогічною назві додатка.

Структура шаблонів Django

Керуючими елементами шаблонів Django є змінні, фільтри і теги.

При рендерингу шаблону змінні замінюються на своє значення, обчислене в контексті виклику. Синтаксис – подвійні фігурні дужки: {{title}}.

Фільтри служать для простих перетворень змінних. Синтаксис – змінна | імя_фільтра: "параметри". Фільтр може зустрічатися як в змінних, так і в якості параметра тега. Наприклад: {{title | lowercase}}.

При рендерингу шаблону теги, грубо кажучи, замінюються на результати роботи асоційованої з цим тегом функції на Python. Синтаксис: {% тег параметри %}, наприклад: {% url blog_article slug = article.slug %}.

Крім того, є три спеціальних тега: include, block і extend. Тег include підставляє запитаний шаблон (відрендерене в поточному контексті).

Тепер перейдемо до особливостей Django: на тегах block і extend будується успадкування шаблонів. Зупинимося на них докладніше.

Основна фішка шаблонів Django – наслідування. Шаблон може розширювати (уточнювати) поведінку батьківського шаблону.

Будь-яка ділянка шаблону може бути обгорнутою в блоковий тег (природно, що тег не може починатися перед, а закінчуватися всередині циклу). Блоку дається ім'я.

{% block content %}

тіло блоку

{% endblock %}

За допомогою тега extend ми вказуємо, який шаблон ми будемо уточнювати. Розширюючи шаблон, ми можемо переоприділити будь-які блоки, які є в батьківському шаблоні. Все, що знаходиться поза цих блоків, буде пропущено. Виходить потужний механізм, практично виключає необхідність повторення частин шаблонів. Цей підхід збільшує повторне використання коду і спрощує додавання елементів до розділених частин сайту, наприклад, до навігації. Також варто враховувати наступні поради для роботи з наслідуванням шаблонів:

- Якщо використовується {% extends %} в шаблоні, він повинен бути першим тегом в цьому шаблоні. В іншому випадку, успадкування шаблонів працювати не буде.
- У загальному випадку, чим більше тегів {% block %} в основному шаблоні, тим краще. Дочірні шаблони не зобов'язані визначати всі блоки основного шаблону. Так що можна вказати розумні значення за замовчуванням

в ряді блоків, а потім визначити в дочірньому шаблоні тільки ті, які треба змінити. Краще мати більше обробників.

- Щоб не повторювати код в ряді шаблонів, треба перенести цей код в {% block %} в основному шаблоні.
- Щоб отримати вміст блоку з основного шаблону, змінна {{block.super}} допоможе з цим. Це корисно, якщо вам буде потрібно лише додати дані в блок, замість його повної заміни.
- Не можна визначати безліч тегів {% block %} з однаковим ім'ям в одному шаблоні. Це обмеження існує через те, що ці теги працюють в «обох» напрямках. Тобто, цей тег не просто надає місце для заповнення даними, він також визначає вміст, який заповнює місце в основному шаблоні. Якби було два однаково названих теги {% block %} в шаблоні, то основний шаблон не знав би, який вміст блоку використовувати.
- Шаблон, ім'я якого ви передається в {% extends %} завантажується тим же способом, що і при використанні get_template(). Тобто, ім'я шаблону додається до вмісту параметра TEMPLATE_DIRS.
- У більшості випадків, аргументом для тегу {% extends %} є рядок. Але може бути і змінна, якщо не відоме ім'я основного шаблону до запуску додатка. Це дозволяє реалізовувати динамічні речі.

Форуємо батьківський шаблон, додавши наступний код в base.html:

```
{% load staticfiles %}

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <!-- The above 3 meta tags *must* come first in the head; any
other head content must come *after* these tags -->
    <meta name="description" content="">
    <meta name="author" content="">

    <title>{% block title %}{% endblock %} &ndash; Blog</title>

    <!-- Bootstrap core CSS -->
```

```
<link href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.6/journ
al/bootstrap.min.css" rel="stylesheet">

      <link href="{% static 'blog/style.css' %}" rel="stylesheet">
  </head>

  <body>

    <nav class="navbar navbar-inverse navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed"
data-toggle="collapse" data-target="#navbar" aria-expanded="false"
aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="{% url 'home' %}">Blog</a>
        </div>
        <div id="navbar" class="collapse navbar-collapse">
          <ul class="nav navbar-nav">
            <li><a href="{% url 'home' %}">Home</a></li>
            <li><a href="{% url 'about' %}">About</a></li>
            <li><a href="#contact">Contact</a></li>
          </ul>
        </div><!--/.nav-collapse -->
      </div>
    </nav>

    <div class="container mainContent">
        {%  block content %}
        {%  endblock %}

    </div><!-- /.container -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/j
query.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bo
otstrap.min.js"></script>
  </body>
</html>
```

Шаблон головної сторінки:

```
{% extends 'blog/base.html' %}

{% block title %}
Home
{% endblock %}
```

```
{% block content %}

    <h1>Home</h1>

    {% for article in articles %}
        <h4><a href="{% url 'article' article.id %}">{{
article.title }}</a></h4>
        <pre>{{ article.get_short_text }}</pre>
    {% endfor %}

{% endblock %}
```

Ось в нас готовий макет головної сторінки.

Реєстрація користувачів. Django надає інструменти для вирішення цього завдання. Система аутентифікації користувачів Django управляє акаунтами користувачів, групами, правами і призначеними для користувача сесіями (за допомогою cookie). Цю систему часто називають системою аутентифікації і авторизації (auth/auth або система AA). Подвійне ім'я означає, що упізнання користувачів часто складається з двох кроків. Нам необхідно:

- Перевірити (автентифікувати), хто є цей користувач, наприклад, за допомогою перевірки імені та пароля по базі даних користувачів.
- Перевірити, що користувач авторизований для виконання певних дії. Можна реалізувати за допомогою перевірки по таблиці прав.

Згідно з цими потребами система AA складається з ряду частин:

- Користувачі: Люди, зареєстровані на сайті.
- Права: Бінарні (так/ні) прапори, що визначають, чи може даний користувач виконувати певне завдання.
- Групи: Загальний спосіб застосування міток і прав на групу користувачів.
- Повідомлення: Простий спосіб організації черги і відображення системних повідомлень користувачам.
- Профайли: Механізм додавання в об'єкт користувача певних полів.

В Django є власний інтерфейс адміністратора, де є велика кількість цих інструментів. Якщо ми управляємо користувачами і групами через інтерфейс адміністратора, в дійсності ми управляємо даними в таблицях системи аутентифікації.

Підтримка аутентифікації скомпонована у вигляді модуля в django.contrib.auth. За замовчуванням, необхідні настройки вже включені в settings.py, створюваний за допомогою команди django-admin startproject, і являють собою два записи в параметрі конфігурації INSTALLED_APPS:

- 'Django.contrib.auth' містить ядро системи аутентифікації та її стандартні моделі.
- 'Django.contrib.contenttypes' є фреймворком типів, який дозволяє призначати права на створювані нами моделі.

і записи в параметрі конфігурації MIDDLEWARE_CLASSES:

- SessionMiddleware управляє сесіями під час запитів.
- AuthenticationMiddleware асоціює користувачів з запитами за допомогою сесій.
- SessionAuthenticationMiddleware разлогінює користувача з усіх його сесій, якщо він поміняє пароль.

При наявності цих налаштувань, застосування команди manage.py migrate створює в базі даних необхідні для системи аутентифікації таблиці, створює права для будь-яких моделей всіх зареєстрованих додатків. Після установки ми готові працювати з користувачами в функціях уявлень.

Основою системи аутентифікації є об'єкти User. Вони представляють користувачів сайту і використовуються для перевірки прав доступу, реєстрації користувачів, асоціації даних з користувачами. Для представлення користувачів в системі аутентифікації використовується тільки один клас, таким чином 'суперкористувачі' або 'персонал' – це такі ж об'єкти користувачів, просто з певними атрибутами.

Основні атрибути користувача:

- username – обов’язковий атрибут. Довжина 30 символів або менше. Імена користувачів можуть містити буквено-цифровий, _, @, +,. і - символи.
- password – обов’язковий атрибут. В базі даних збірігається хэш пароля, Django не зберігає вихідний пароль. Не хешований пароль можуе бути як завгодно довгий та містити будь-який символ.

- email – не обов’язковий. Адреса електронної пошти.
- is_active – бульова змінна. Позначає, чи повинен цей обліковий запис користувача вважається активним. Варто встановити за замовчуванням цей прапор в значення False замість видалення облікових записів. Таким чином, якщо програми мають будь-які зовнішні ключі для користувачів, вони не будуть ламатися. Цей атрибут не контролює, чи може користувач увійти. Якщо ми хочемо відмовитися від входу на основі is_active = False, ми наємо написати власну систему входу. Проте, AuthenticationForm базований на методі login() виконує цю перевірку, як і методи перевірки, такі як has_perm() і автентифікація в Django admin.
- groups – вінношення many-to-many. Довжина імені групи – до 80 будь-яких символів.

Для аутентифікації користувача по імені і паролю використовується authenticate(). Параметри авторизації передаються як іменовані аргументи, за замовчуванням це ім'я користувача та пароль, якщо пароль і ім'я користувача вірні, буде повернуто об'єкт користувача. Якщо пароль неправильний, authenticate() не повертає None.

Визначити, чи авторизувався користувач можна за допомогою методу is_authenticated().

Щоб прив'язати до сесії авторизованого користувача, використовується функція login(). Вона приймає об'єкт HttpRequest і об'єкт користувача. Функція зберігає ідентифікатор користувача в сесії використовуючи Django додаток для роботи з сесіями. Слід зазначити, що будь-які дані встановлені в анонімної сесії будуть збережені в сесії користувача після його авторизації.

Коли ми авторизуємо користувача, то повинні успішно виконати його аутентифікацію за допомогою функції authenticate() перед викликом функції login(). Функція аутентифікації встановлює атрибут у класі User, який вказує бекенд щодо якого був успішно аутентифікований даний користувач, що знадобиться пізніше для процесу авторизації. При спробі авторизації об'єкта

користувача, який був отриманий безпосередньо з бази, буде викинута помилка.

Для скасування авторизації користувача, який був авторизований за допомогою функції django.contrib.auth.login(), слід використовувати функцію django.contrib.auth.logout() в коді уявлення. Функція приймає об'єкт HttpRequest і не повертає ніяких значень. logout() не викидає ніяких помилок, якщо користувач не був раніше авторизований.

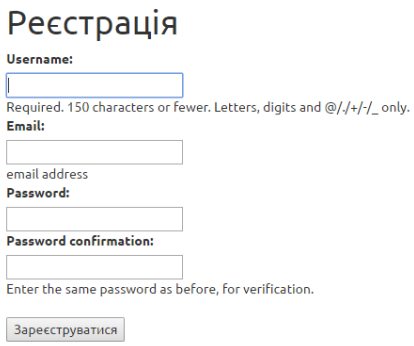
При виконанні цієї функції в рамках поточного запиту будуть очищені всі дані сесії. Всі існуючі дані будуть стерті. Це відбувається для того, щоб запобігти можливості доступу до цих даних для іншого користувача, який буде використовувати той же браузер для своєї авторизації. Якщо буде потрібно помістити якісь дані в сесію, що повинні бути доступні користувачу відразу після скасування його авторизації, потрібно виконувати це після виклику функції django.contrib.auth.logout().

Встановлюємо django_registration-2.2-py3.5.egg-info та створюємо необхідні шаблони.

/templates/registration/registration_form.html

```
{% extends "blog/base.html" %}
{% csrf_token %}
{% block content %}
<h1>Регистрация</h1>
<form method="post" action=""> {% csrf_token %}
<dl class="register">
{% for field in form %}
    <dt>{{ field.label_tag }}</dt>
    <dd class="clearfix">{{ field }}
    {% if field.help_text %}<div class="clearfix">{{
field.help_text }}</div>{% endif %}
    {% if field.errors %}<div class="myerrors clearfix">{{
field.errors }}</div>{% endif %}
    </dd>
{% endfor %}
```

```
</dl>
<input type="submit" value="Зарегистрироваться" class="clearfix"
/>
</form>
{% endblock %}
```



Наступною створимо сторінку новин. Як ми вже зазначили, новини відображаємо відсортованими у зворотному порядку. У загальному списку показуємо заголовок – коротка назва, текст – частина посту, дату публікації звістки. Клацнувши по заголовку відкривається детальний текст – оригінал статті.

Спочатку створимо форму для додавання звісток. Створюємо файл templates/news/new_article.html, де розміщуємо код:

```
{% extends 'news/base.html' %}

{% block title %}
    Новина
{% endblock %}

{% block content %}

    <form method="POST" class="article_form">{% csrf_token %}
        {{ form.as_p }}

        <input type="submit">
    </form>

{% endblock %}
```

Так як ми створюємо POST форму (яка може привести до змін даних), нам слід подбати про Cross Site Request Forgeries – вид атак на відвідувачів веб-сайтів, що використовує недоліки протоколу HTTP . Завдяки Django це

дуже просто. Загалом, всі POST форми, які відправляються на внутрішній URL, повинні використовувати тег {% csrf_token %}.

Створюємо представлення, що приймає дані з відправленої форми та обробляє їх. Допишемо в news/urls.py:

```
url(r'^articles/new/$', 'news.views.article_new', name='d_new'),
```

Розвиваємо функцію представлення news/views.py:

```
def article_new(request):
    if request.method == "POST":
        form = ArticleForm(request.POST)
        if form.is_valid():
            article = form.save(commit=False)
            article.author = request.user
            article.published_date = timezone.now()
            article.save()
            return redirect('article_detail', pk=article.pk)
    else:
        form = ArticleForm()
    return render(request, 'news/new_article.html', {'form':
form})
```

request.POST – це об'єкт з інтерфейсом словника, який дозволяє отримати відправлені дані через назву ключа. Зауважимо що Django також надає request.GET для аналогічного доступу до GET даних, але ми використовуємо request.POST, щоб бути впевненими, що дані передаються тільки при POST запиті.

Головним завданням об'єкта Form є перевірка даних. Коли форма заповнена, викличемо метод is_valid() для виконання перевірки і отримання її результату.

Якщо з введеними даними все добре, відправляємо автора на сторінку з його постом:

news/views.py

```
def show_article(request, article_id):
    article = get_object_or_404(Article, id=article_id)
    return render(request, 'news/article.html', {'article':
article})
```

news/urls.py

```
url(r'^articles/(?P<article_id>[0-9]+)/$',
'news.views.show_article', name='article_detail'),
```

templates/news/article.html

```
{% extends 'news/base.html' %}

{% block title %}
    {{ article.title }}
{% endblock %}

{% block content %}
    <h4>{{ article.title }}</h4>
    <pre>{{ article.text }}</pre>
{% endblock %}
```

Всі статті розмістимо на сторінці новин за вже описаним алгоритмом. Отже, створюємо html-сторінку з новинами та додаємо код в файли-модулі додатку.

```
{% block title %}
News
{% endblock %}

{% block content %}

    <h1>Page with news</h1>

    <h4><a href="{% url 'd_new' %}">Запропонувати новину</a></h4>

    {% for article in articles %}

        <h4><a href="{% url 'article_detail' article.id %}">{{
article.title }}</a></h4>

        <pre>{{ article.get_short_text }}</pre>


    {% endfor %}
{% endblock %}
```

news/urls.py

```
url(r'^articles/(?P<article_id>[0-9]+)/$',
'news.views.show_article', name='article_detail'),
```

news/views.py

```
def home(request):
    articles = Article.objects.all().order_by('-published_date')
    context = {
        'articles': articles
    }
    return render(request, 'news/home.html', context)
```

Отже, ми створили головну сторінку (Рис. 2), що містить фото та короткі дані про факультет.

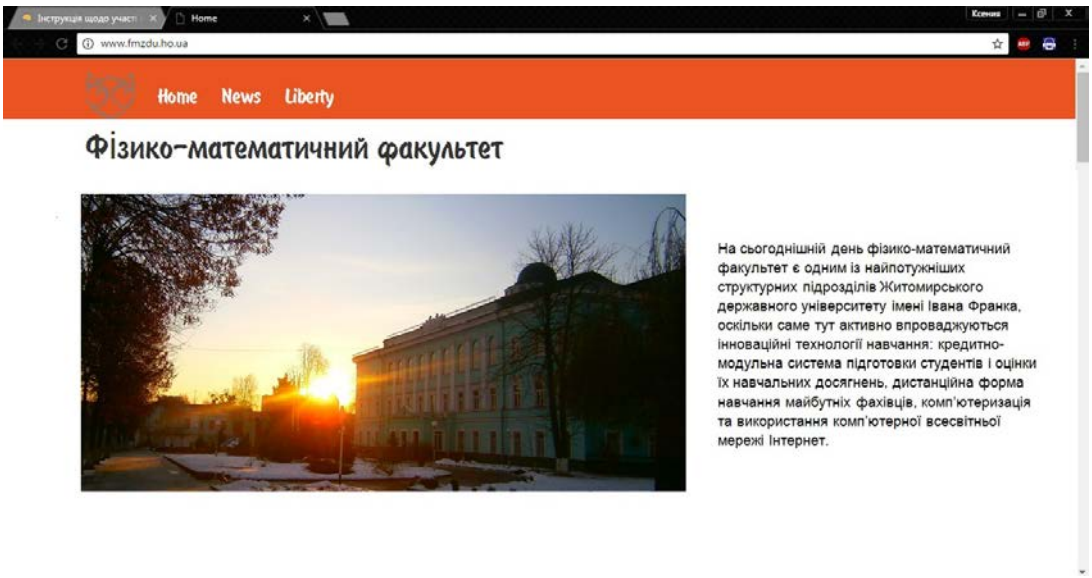


Рис. 2. Головна сторінка сайту фізико-математичного факультету

Сторінка новин (Рис. 3) наповнена вістками, що подаються у відсортованому порядку, починаючи з останніх. Новина складається з заголовку та тексту до неї (відображається максимум 200 символів). Щоб переглянути текст повністю, достатньо клацнути на заголовок. Будь-який користувач може запропонувати новину. Це може бути завдання студентам, запрошення на захід, побажання, дані про поїздки тощо.

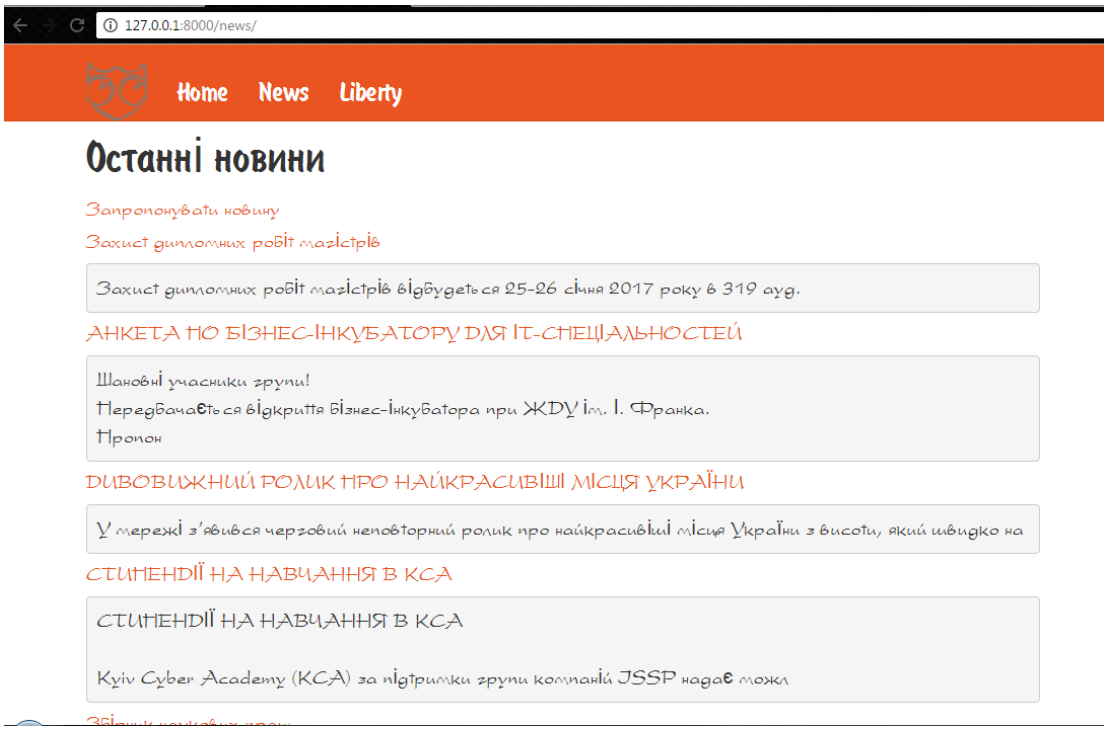


Рис. 3. Сторінка новин

На наступній сторінці (Liberty) містяться дані про змагання, гуртки тощо (Рис. 4). Поки студенти на канікулах, пропонуємо декілька порад для їх проведення.

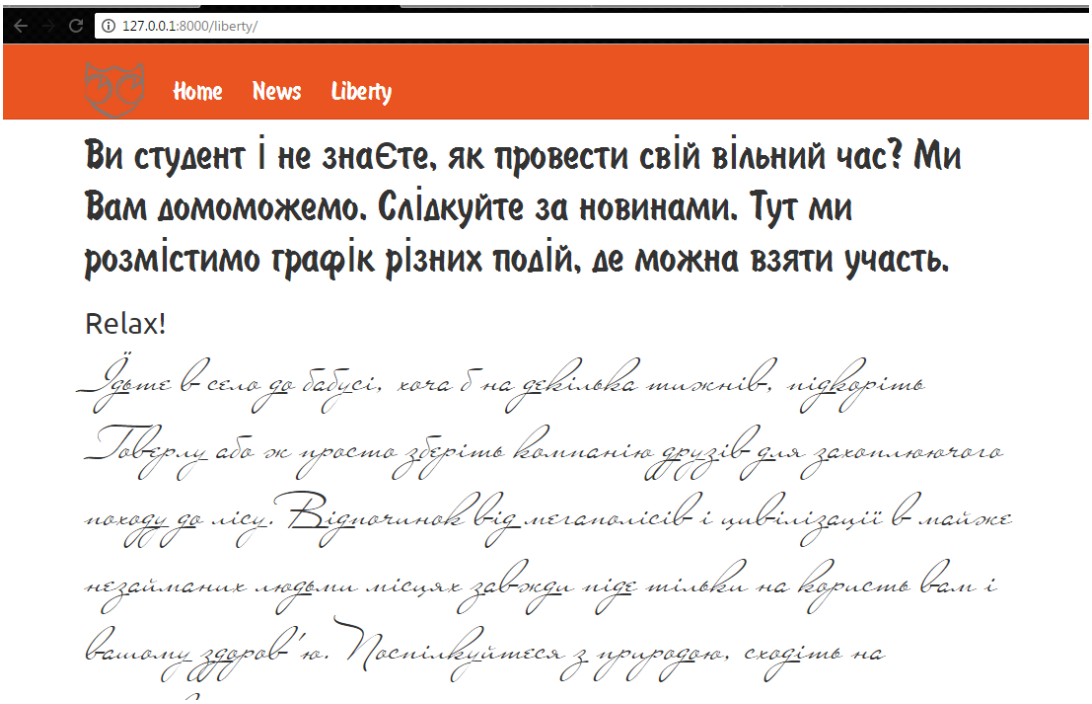


Рис. 4. Проведення вільного часу студентам

На кожній сторінці розміщене зафіксоване верхнє меню, яке містить посилання на інші сторінки сайту. Логотип із зображенням сови в рядку меню є посиланням на головну сторінку. Якщо вікно браузера має маленьку ширину, меню у верхньому правому куті перетворюється із звичайного ряду посилань у випадаюче меню. Це корисно при перегляді сайту на маленькому дисплеї, наприклад, телефоні.

В подальшому планується удосконалення сайту та створення нових можливостей для користувачів.

Висновки

В рамках даного проекту було досліджено особливості роботи з фреймворком та розроблено сайт фізико-математичного факультету.

Досліджено особливості різних видів фреймворків, які поділені на типи за мовою програмування. Визначено, що вибір фреймворку специфічний для кожного проекту і залежить від поставлених завдань. Ми обрали Django, оскільки він зручний та гнучкий в роботі.

Було описано використання Django в проекті. Сам проект складено з окремих частин (додатків, модулів, пакетів тощо), що дає йому гнучкість в роботі і продуктивність у використанні коду. Суть структури додатка проекту на Django зводиться до того, що вона працює частково за образом MVC, але істотно від відрізняються моделі в першу чергу тим, що контролери в MVC – це URL Django.

Сформована структура сайту відповідно до потреб цільової аудиторії сайту фізико-математичного факультету. Вона вміщує сторінки, розділи, підрозділи, навігацію (посилання), розташування цих елементів і їх зв’язок між собою. Продумана і зручна структура допомагає нам значно скоротити витрати на збір інформативної складової сайту та уникнути великої кількості помилок на етапі розробки.

Створений сайт фізико-математичного факультету з використанням Django Framework. У роботі детально описані кроки розробки проекту.

Список використаних джерел:

1. Software framework [Електронний ресурс]. – Режим доступу : URL : https://en.wikipedia.org/wiki/Software_framework. – Назва з екрану.
2. Фреймворки в веб-разработке. [Електронний ресурс]. – Режим доступу : URL : https://web-creator.ru/articles/about_frameworks. – Назва з екрану.
3. Що таке Yii. [Електронний ресурс]. – Режим доступу : URL : https://github.com/RangelReale/wyii/blob/master/docs/guide/uk/quickstart.what-is-yii.txt. – Назва з екрану.
4. An elegant HMVC PHP5 framework that provides a rich set of components for building web applications. [Електронний ресурс]. – Режим доступу : URL : https://kohanaframework.org/. – Назва з екрану.
5. Обзоры web-фреймворков. [Електронний ресурс]. – Режим доступу : URL : https://praktikatech.wordpress.com/category/%D0%BE%D0%B1%D0%B7%D0%BE%D1%80%D1%8B-web-%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA%D0%BE%D0%B2/page/2/. – Назва з екрану.
6. Почему нужно использовать php-framework’и, на примере codeigniter. – Болиев В. [Електронний ресурс]. – Режим доступу : URL : https://habrahabr.ru/post/38696/. – Назва з екрану.
7. Flask. [Електронний ресурс]. – Режим доступу : URL : https://www.fullstackpython.com/flask.html. – Назва з екрану.
8. The Vision For Twisted. [Електронний ресурс]. – Режим доступу : URL : http://twistedmatrix.com/documents/current/core/howto/vision.html. – Назва з екрану.
9. Docs. Tornado Web Server. [Електронний ресурс]. – Режим доступу : URL : http://www.tornadoweb.org/en/stable/index.html. – Назва з екрану.
10. Pyramid Introduction. [Електронний ресурс]. – Режим доступу : URL : http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/introduction.html. – Назва з екрану.

11. Web MVC framework. Режим доступу: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html. – Назва з екрану.

12. JSNI. [Електронний ресурс]. – Режим доступу : URL : http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsJSNI.html. – Назва з екрану.

13. Why .NET? [Електронний ресурс]. – Режим доступу : URL : https://www.microsoft.com/net/intro. – Назва з екрану.

14. Начинаем работу с Django. [Електронний ресурс]. – Режим доступу : URL : http://itman.in/django-howto/#i-4. – Назва з екрану.

15. Дронов В.А. Django: практика создания Web-сайтов на Python. – СПб.: БХВ-Петербург, 2016. – 528 с.: ил. – (Профессиональное программирование).

16. Структура сайту. [Електронний ресурс]. – Режим доступу : URL : http://internetjournal.ucoz.ru/publ/strukturv_sajtu/1-1-0-13. – Назва з екрану.

17. Проектування баз даних. [Електронний ресурс]. – Режим доступу : URL : http://strong-planet.com.ua/database.html. – Назва з екрану.

18. Куліковська О. В. Проектування бази даних студентського сайту / О. В. Куліковська // Актуальні питання сучасної інформатики: Тези доповідей Всеукраїнської науково-практичної конференції з міжнародною участю “Сучасні інформаційні технології в освіті та науці” (10-11 листопада 2016 р.) / за ред. Т. А. Вакалюк. – Житомир: Вид-во ЖДУ ім. І. Франка, 2016. – Вип. 3. – 292 с. – С. 206-208.

19. Сервер для розробки. Активація моделей. [Електронний ресурс]. – Режим доступу : URL : http://dl-cloud.kpi.ua/stud/sites/Django/lesson3.html. – Назва з екрану.

20. Наследование шаблонов. Глава 4. Шаблоны. [Електронний ресурс]. – Режим доступу : URL : http://djbook.ru/ch04s07.html. – Назва з екрану.

21. Уязвимость CSRF. Введение. [Електронний ресурс]. – Режим доступу : URL : https://intsystem.org/security/learn-about-csrf-intro/. – Назва з екрану.

22. Вимоги до веб-сайтів загальноосвітніх навчальних закладів. [Електронний ресурс]. – Режим доступу : URL : http://chutosvita.at.ua/_ld/0/55___.doc. – Назва з екрану.

23. Форсье Дж., Биссекс П., Чан У. Django. Разработка веб-приложений на Python. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 456 с., ил.