МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИИ НАЦІОНАЛЬНИИ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА САЙТУ ДЛЯ АВТОМАТИЗАЦІЇ ЗБИРАННЯ ДОВІДОК»**

Виконав: студент 5 курсу, групи 6.1228-з
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
Єркін Б. О.
(ініціали та прізвище)
Керівник: старший викладач кафедри комп'ютерних наук,
Циммерман Г. А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент: завідувач кафедри програмної інженерії, к.ф.-м.н,
доцент Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

ЗАПОРІЖЖЯ
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИИ НАЦІОНАЛЬНИИ УНІВЕРСИТЕТ

Факультет математичний
Кафедра комп'ютерних наук
Рівень вищої освіти бакалавр
Спеціальність 122 комп'ютерні науки
(шифр і назва)
Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук, д.т.н., професор
_____ Чопоров С.В.
(підпис)

ЗАВДАННЯ
НА КВАЛІФІКАЦІИНУ РОБОТУ СТУДЕНТУ

Єркіну Богдану Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка сайту для автоматизації збирання довідок

керівник роботи Циммерман Генадій Анатолійович
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від «26» грудня 2023 року №103-с

2. Строк подання студентом роботи 24.05.2023

3. Вихідні дані до роботи 1. Постановка задачі
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Система управління базою даних.

5. Перелік графічного матеріалу
презентація

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Циммераман Г.А. старший викладач кафедри комп'ютерних наук		
	Лісняк А.О. завідувач кафедри програмної інженерії, к.ф.-м.н, доцент		

6. Строк подання студентом роботи 25.05.2023

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	05.02.2023	
2.	Аналіз предметної області.	20.02.2023	
3.	Обробка методичних джерел	02.03.2023	
4.	Розробка першого та другого розділу.	18.03.2023	
5.	Розробка третього розділу.	28.04.2023	
6.	Оформлення та нормоконтроль кваліфікаційної роботи бакалавра.	25.05.2023	
7.	Захист кваліфікаційної роботи	25.05.2023	

Студент _____ Б.О. Єркін
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г.А. Циммерман
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «РОЗРОБКА САЙТУ ДЛЯ АВТОМАТИЗАЦІЇ ЗБИРАННЯ ДОВІДОК»: 66 с., 9 рис., 0 табл., 14 джерел, 2 додатків.

АУТЕНТИФІКАЦІЯ, ВЕБ-САЙТ, ВЕБ-ФРЕЙМВОРК ДЖАНГО, СОРТУВАННЯ, ПАЙТОН.

Об'єкт дослідження – розробка веб-сайту для автоматизації збору рефератів за допомогою Python та Django.

Предмет дослідження - аналіз, проектування та реалізація веб-системи, яка полегшує збір довідок в автоматизованому режимі, підвищуючи ефективність та точність управління даними про студентів та співробітників.

Мета роботи: розробка функціонального та зручного веб-сайту, який спрощує процес збору довідок в умовах підприємства. Мета полягає в тому, щоб забезпечити централізовану платформу для управління даними про студентів і співробітників, що дозволяє безперешкодно збирати довідки, безпечну автентифікацію та ефективний пошук даних.

Метод дослідження – аналітичнии.

Дане дослідження було присвячене розробці веб-сайту для автоматизації збору довідкової інформації в умовах підприємства. У проекті було використано мову програмування Python та веб-фреймворк Django для створення надійної та зручної системи. Отримані результати включають успішну реалізацію таких ключових функцій, як автентифікація співробітників, список всіх студентів з можливостями пагінації та сортування, можливість додавання/редагування/видалення студентів, картки профілю студентів, особисті кабінети співробітників, сторінка перегляду журналу адміністратора, сторінка для списку співробітників і студентів, налаштування привілеїв співробітників, автоматичне завершення сеансу і відновлення пароля з дозволу адміністратора. Результати цього дослідження можуть бути корисними для навчальних закладів, компаній та організацій, які потребують впорядкованого та автоматизованого підходу до збору довідкової інформації. Веб-сайт зменшує ручну працю, мінімізує помилки та підвищує продуктивність в управлінні даними про студентів та співробітників.

SUMMARY

Bachelor's Thesis 'Development of a Website for Automating the Collection of References': 66 pages, 9 figures, 0 tables, 14 sources, 2 appendices.

AUTHENTICATION, WEBSITE, DJANGO WEB FRAMEWORK, SORTING, PYTHON.

The object of research is the development of a website for automating the collection of references using Python and Django.

The subject of research is the analysis, design, and implementation of a web system that facilitates the collection of references in an automated manner, enhancing the efficiency and accuracy of managing data about students and staff members.

The purpose of the study is to develop a functional and user-friendly website that simplifies the process of collecting references in an enterprise environment. The goal is to provide a centralized platform for managing data about students and staff members, enabling seamless reference collection, secure authentication, and efficient data retrieval. The research method used is analytical.

This study focused on the development of a website for automating the collection of reference information in an enterprise setting. The project utilized the Python programming language and the Django web framework to create a reliable and convenient system. The obtained results include the successful implementation of key features such as staff authentication, a list of all students with pagination and sorting capabilities, the ability to add/edit/delete students, student profile cards, staff member personal accounts, an administrator's log viewer page, a page listing staff members and students, staff member privilege settings, automatic session termination, and password recovery with administrator approval. The findings of this research can be beneficial for educational institutions, companies, and organizations that require an organized and automated approach to collecting reference information. The website reduces manual work, minimizes errors, and improves productivity in managing data about students and staff members.

Зміст

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

AJAX: Asynchronous JavaScript and XML
API: Application Programming Interface
CLI: Command Line Interface
CRUD: Create, Read, Update, Delete (database operations)
CSRF: Cross-Site Request Forgery
CSV: Comma-Separated Values
CSS: Cascading Style Sheets
CSS3: Cascading Style Sheets version 3
DBMS: Database Management System
DRY: Don't repeat yourself
FTP: File Transfer Protocol
HTML: Hypertext Markup Language
HTML5: Hypertext Markup Language version 5
HTTP: Hypertext Transfer Protocol
HTTPS: Hypertext Transfer Protocol Secure
IDE: Integrated Development Environment
IaaS: Infrastructure as a Service
JPEG: Joint Photographic Experts Group
JS: JavaScript
JSON: JavaScript Object Notation
JWT: JSON Web Token
MVC: Model-View-Controller
ORM: Object-Relational Mapping
PDF: Portable Document Format
PaaS: Platform as a Service
PNG: Portable Network Graphics
SMTP: Simple Mail Transfer Protocol
SQL: Structured Query Language
SSL: Secure Sockets Layer
SSH: Secure Shell
SaaS: Software as a Service
TLS: Transport Layer Security
UI: User Interface
URL: Uniform Resource Locator
URL: Uniform Resource Locator
UX: User Experience
XSS: Cross-Site Scripting
SVG: Scalable Vector Graphics

ВСТУП

Робота з інформацією є невід'ємною частиною академічної та дослідницької діяльності. Однак процес збору та організації може забирати багато часу і бути схильним до помилок, особливо коли йдеться про велику кількість даних. З розвитком технологій використання паперових систем для роботи з даними втратило свою актуальність. Веб-платформи стали кращим варіантом завдяки простоті використання, доступності та безпеці. Однак традиційним системам обліку довідок часто бракує гнучкості та функціональності, необхідних для ефективного управління записами, що призводить до помилок і значних затрат часу. Метою даної дипломної роботи є розробка веб-платформи для автоматизації збору довідок. Запропонована система надаватиме зручний інтерфейс для зберігання та обробки довідок, підтримку з боку адміністратора, а також API для майбутньої взаємодії з зовнішніми інтеграціями

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Проблеми документообігу на підприємстві

Управління документами є ключовою функцією будь-якого підприємства, оскільки воно передбачає створення, зберігання, пошук і знищення різних типів документів. Однак ефективне управління документами може бути складним завданням для багатьох організацій, особливо якщо вони створюють і обробляють великі обсяги електронних і паперових документів. Ось деякі типові проблеми, з якими стикаються підприємства в управлінні документами:

1) Втрачені або неправильно розміщені документи: Однією з найпоширеніших проблем в управлінні документами є втрата або неправильне розташування важливих документів. Це може статися з різних причин, таких як погана організація, неналежне зберігання або відсутність належних механізмів відстеження.

2) Проблеми нормативно-правового регулювання: Ще однією важливою проблемою в управлінні документами є дотримання законодавчих і нормативних вимог. Недотримання таких законів, як GDPR, HIPAA та SOX, може призвести до серйозних штрафів та юридичної відповідальності.

3) Ризики безпеки: Підприємства також стикаються з ризиками безпеки, пов'язаними з управлінням документами, включаючи несанкціонований доступ, витоки та втрату даних. Це може бути особливо складно у випадку з конфіденційними документами, такими як контракти, фінансова звітність та дані співробітників.

4) Неефективні робочі процеси: Неефективний документообіг також може бути проблемою для підприємств, що призводить до затримок у прийнятті рішень і зниження продуктивності. Це може статися через брак автоматизації, ручні процеси або погану співпрацю та комунікацію.

5) Труднощі з доступом до інформації: Підприємства також можуть мати труднощі з швидким і легким доступом до інформації, що ускладнює прийняття обґрунтованих рішень. Це може бути пов'язано з недостатньою організацією, поганими можливостями пошуку або відсутністю інтеграції між різними системами управління документами.

6) Збільшення витрат на зберігання: Нарешті, витрати на зберігання та управління документами можуть бути значною проблемою для підприємств, особливо з огляду на те, що обсяг документів продовжує зростати. Підприємства можуть намагатися знайти економічно ефективні рішення для управління та зберігання документів, що може вплинути на їхній фінансовий результат.

Загалом, ці проблеми підкреслюють важливість ефективного управління документами на підприємстві. Вирішивши ці проблеми, підприємства можуть забезпечити доступ до важливої інформації, коли вона їм потрібна, дотримання законодавчих і нормативних вимог, а також ефективну та економічно доцільну роботу.

1.2 Поняття інформаційної системи

Інформаційна система (ІС) - це сукупність апаратних засобів, програмного забезпечення, даних і людей, які працюють разом для управління та обробки інформації в організації. Вона призначена для підтримки діяльності, операцій та процесів прийняття рішень в організації.

Основні компоненти інформаційної системи включають в себе:

1) Апаратне забезпечення: Сюди входять фізичні пристрої, такі як комп'ютери, сервери, пристрої зберігання даних та інші периферійні пристрої, що використовуються для зберігання, обробки та передачі інформації.

2) Програмне забезпечення: Сюди входять програми, операційні системи та інші програмні засоби, що використовуються для обробки, аналізу та управління інформацією.

3) Дані: Це стосується структурованої та неструктурованої інформації, яка збирається, зберігається та аналізується в інформаційній системі.

4) Процедури: Це правила, політики та процедури, які регулюють використання інформаційної системи в організації.

5) Люди: Це стосується осіб, які експлуатують, підтримують і використовують інформаційну систему для виконання своїх завдань і досягнення цілей.

Метою інформаційної системи є надання своєчасної, точної та релевантної інформації для підтримки прийняття рішень, планування та управління в організації. Вона може бути використана для автоматизації рутинних завдань, оптимізації бізнес-процесів, а також для забезпечення співпраці та комунікації між різними відділами та функціями.

Інформаційні системи можна класифікувати на різні типи залежно від їхніх функцій, сфери застосування та можливостей. Деякі поширені типи інформаційних систем включають:

1) Системи обробки транзакцій (TPS): Ці системи призначені для підтримки операційної діяльності організації, наприклад, обробки замовлень, управління запасами та нарахування заробітної плати.

2) Управлінські інформаційні системи (MIS): Ці системи призначені для надання інформації менеджерам та особам, які приймають рішення, для підтримки їхньої діяльності з планування, контролю та прийняття рішень.

3) Системи підтримки прийняття рішень (DSS): Ці системи призначені для підтримки прийняття складних рішень, надаючи аналітичні інструменти, моделі та симуляції, які допомагають особам, що приймають рішення, оцінити альтернативні шляхи дій.

4) Системи планування ресурсів підприємства (ERP): Ці системи призначені для інтеграції та управління всіма бізнес-функціями та процесами організації, включаючи фінанси, людські ресурси та управління ланцюгами поставок.

Отже, інформаційна система є важливим інструментом для управління та обробки інформації в організації. Надаючи своєчасну, точну та релевантну інформацію, вона дозволяє організаціям приймати обґрунтовані рішення, оптимізувати свою діяльність та залишатися конкурентоспроможними в сучасному мінливому бізнес-середовищі.

1.3 Аналіз відомих автоматизованих систем

На ринку існує безліч автоматизованих систем, які можуть допомогти організаціям впорядкувати свою діяльність, підвищити ефективність та зменшити витрати. Ось аналіз деяких відомих автоматизованих систем:

1) Системи управління взаємовідносинами з клієнтами (CRM): CRM-системи призначені для того, щоб допомогти бізнесу керувати взаємодією з клієнтами та потенційними клієнтами. Зазвичай вони включають такі функції, як управління контактами, відстеження продажів і автоматизація маркетингу. До відомих CRM-систем належать Salesforce, Microsoft Dynamics CRM і Hubspot.

2) Системи планування ресурсів підприємства (ERP): ERP-системи призначені для інтеграції та управління всіма бізнес-функціями та процесами організації. Зазвичай вони включають модулі для управління фінансами, людськими ресурсами, закупівлями та ланцюжками поставок. До відомих ERP-систем належать SAP, Oracle ERP Cloud та Microsoft Dynamics 365.

3) Інформаційні системи управління персоналом (HRIS): HRIS-системи призначені для управління та відстеження інформації про працівників і HR-процесів. Зазвичай вони включають такі функції, як розрахунок заробітної плати, управління пільгами та відстеження продуктивності. Відомі системи HRIS включають Workday, ADP Workforce Now і BambooHR.

4) Системи управління ланцюгами поставок (SCM): Системи SCM призначені для управління потоком товарів і послуг від постачальників до клієнтів. Вони зазвичай включають такі функції, як управління запасами, закупівлі та логістика. Відомі системи SCM включають Oracle SCM, SAP Supply Chain Management та JDA Software.

5) Системи бізнес-аналітики (BI): Системи бізнес-аналітики покликані допомогти організаціям приймати обґрунтовані рішення, надаючи дієві висновки з аналізу даних. Вони зазвичай включають такі функції, як візуалізація даних, звітність та інформаційні панелі. Серед відомих BI-систем - Tableau, Microsoft Power BI та QlikView.

Ці автоматизовані системи мають низку переваг, таких як підвищення ефективності, точності та продуктивності. Вони також забезпечують видимість бізнес-операцій у реальному часі та дозволяють краще приймати рішення. Однак вони мають і певні недоліки, такі як високі витрати на впровадження, потреба в технічній експертизі та ризик збоїв у роботі системи.

Отже, автоматизовані системи стають дедалі популярнішими в сучасному швидкоплинному бізнес-середовищі. Організації повинні ретельно оцінити свої потреби і вибрати правильну систему, яка відповідає їхнім бізнес-цілям і завданням. Вибравши відповідну автоматизовану систему, організації можуть покращити свою діяльність і отримати конкурентну перевагу у відповідних галузях.

1.4 Розвиток Інтернет технологій та їх використання для розробки

Розвиток інтернет-технологій трансформував світ бізнесу та створив численні можливості для розвитку в різних галузях. Поява Інтернету уможливила швидкий обмін інформацією та створення нових ринків, проклавши шлях для інноваційних додатків та послуг.

Спочатку Інтернет використовувався переважно для спілкування електронною поштою та обміну файлами. Однак з розвитком Всесвітньої павутини (WWW) у 1990-х роках Інтернет став більш доступним для широкої громадськості, що призвело до створення численних веб-сайтів та онлайн-сервісів. Це відкрило нову еру електронної комерції та онлайн-транзакцій, які революціонізували спосіб взаємодії бізнесу з клієнтами та ведення операцій.

З розвитком Інтернету з'явилися нові технології, такі як соціальні мережі, хмарні обчислення та мобільні пристрої, що відкрили нові можливості для розвитку. Платформи соціальних мереж, такі як Facebook і Twitter, стали потужними маркетинговими інструментами, що дозволяють компаніям взаємодіяти зі своїми клієнтами та просувати свої продукти і послуги. Хмарні обчислення

дозволили компаніям зберігати дані та додатки і мати до них віддалений доступ, що зменшило потребу в дорогому обладнанні та інфраструктурі. Мобільні пристрої, такі як смартфони та планшети, забезпечили більшу мобільність та гнучкість, даючи змогу працівникам працювати будь-де та будь-коли.

Використання інтернет-технологій багато в чому революціонізувало розробку програмного забезпечення. Інтернет-технології надали розробникам програмного забезпечення нові інструменти та платформи для створення та розгортання додатків, покращення співпраці та комунікації між членами команди, а також покращення користувацького досвіду програмних продуктів.

Однією з головних переваг інтернет-технологій для розробки програмного забезпечення є можливість віддаленої співпраці. Інтернет-технології дозволили командам розробників програмного забезпечення працювати разом незалежно від їхнього фізичного розташування. Це призвело до підвищення продуктивності та покращення співпраці між членами команди. Онлайн-інструменти для співпраці, такі як GitHub, Trello та Asana, дозволяють розробникам спільно працювати над проектами, відстежувати прогрес та керувати завданнями.

Інтернет-технології також надали розробникам нові платформи та середовища для створення і розгортання програмного забезпечення. Хмарні обчислення спростили доступ до обчислювальних ресурсів, таких як сервери та сховища, без необхідності інвестувати в дороге обладнання. Це дозволило розробникам швидко створювати нові середовища для тестування і розгортання, а також швидко масштабувати додатки в міру зростання попиту.

Інтернет-технології також покращили користувацький досвід програмних продуктів. Поширення мобільних пристроїв призвело до розвитку адаптивного веб-дизайну, який дозволяє веб-сайтам і додаткам адаптуватися до різних розмірів і роздільної здатності екранів. Це полегшило користувачам доступ до програмних продуктів на різних пристроях, включаючи смартфони та планшети.

Використання інтернет-технологій також уможливило розробку нових програмних продуктів і послуг. Наприклад, розвиток програмного забезпечення як послуги (SaaS) спростив для бізнесу доступ до програмних продуктів і їх використання без необхідності встановлювати і підтримувати їх на власних серверах. Це призвело до розробки широкого спектру SaaS-продуктів, включаючи системи управління взаємовідносинами з клієнтами (CRM), бухгалтерське програмне забезпечення та інструменти управління проектами.

Отже, використання інтернет-технологій мало значний вплив на розробку програмного забезпечення, надаючи розробникам нові інструменти і платформи для створення і розгортання додатків, покращуючи співпрацю і комунікацію, а також підвищуючи зручність користування програмними продуктами. Оскільки інтернет-технології продовжують розвиватися, цілком ймовірно, що розробка програмного забезпечення продовжуватиме трансформуватися новими інноваційними способами.

1.5 Постановка задачі створення автоматизованої довідково-інформаційної системи

Метою дипломної роботи є розробка веб-платформи для автоматизації збирання довідок за допомогою Python та Django, яка відповідає наступним вимогам:

1) Локальний файлозберігач для зберігання та обробки документів.

2) Вебінтерфейс за допомогою якого користувач взаємодіятиме з вебсайтом або вебзастосунком через браузер.

3) Процедура автентифікації для надання доступу роботи в інформаційній системі.

4) Панель адміністратора, яка дає змогу додавати, видаляти або редагувати будь-яку модель бази даних через веб-інтерфейс.

Запропонована система усуне недоліки традиційних паперових систем управління довідками, зокрема, велика кількість часу на обробку інформації, помилки та відсутність інтеграції з іншими джерелами. Таким чином, при розробці даної веб-платформи буде досягнуто наступні цілі: скорочення часу на обробку довідок, за рахунок спрощення процедури додавання та редагування.

1.6 Аналіз технічного завдання та етапи створення автоматизованої довідково-інформаційної системи

1.6.1 Призначення розробки автоматизованої довідково-інформаційної системи

Призначенням розробки сайту для автоматизації збору довідок за допомогою Python та Django є створення надійної, ефективної та зручної системи для управління та організації довідок на підприємстві. Система покликана впорядкувати процес збору та зберігання довідок, зменшити кількість помилок та неузгодженостей, а також забезпечити легкий доступ до довідок для уповноважених співробітників. Крім того, система підвищить безпеку та конфіденційність конфіденційних даних завдяки впровадженню функцій безпечної автентифікації та авторизації. Розробка цього веб-сайту в кінцевому підсумку призведе до підвищення продуктивності, покращення комунікації та вдосконалення процесів прийняття рішень на підприємстві.

1.6.2 Вимоги до розробки

Технічне завдання на розробку веб-сайту з використанням Python та Django складається з різноманітних функцій та вимог, які мають вирішальне значення

для належного функціонування веб-сайту. Веб-сайт призначений для автоматизації процесу зберігання та управління даними студентів. Нижче буде розглянуто етапи створення веб-сайту на Python та Django на основі наданих вимог:

1) Автентифікація персоналу є важливою функцією для веб-сайту, і вона буде реалізована за допомогою вбудованої системи автентифікації Django. Ця функція забезпечить безпечну автентифікацію, включаючи хешування паролів та управління сесіями. Для входу в систему співробітники повинні будуть надати свою електронну адресу та пароль. Крім того, в системі буде реалізована двофакторна автентифікація за допомогою повідомлення, надісланого на вказаний співробітником номер у телеграмі, що підвищить безпеку веб-сайту.

2) На сайті буде розміщено список усіх студентів, який відображатиметься у вигляді таблиці, заповненої даними з бази даних. Таблиця буде розбита на сторінки для покращення продуктивності при відображенні великої кількості даних. Крім того, таблицю можна буде сортувати, що дозволить співробітникам сортувати дані на основі різних критеріїв, таких як ім'я або дата народження.

3) Співробітники з відповідними правами зможуть додавати, редагувати та видаляти дані про студентів за допомогою форми. Форма міститиме поля для імені та прізвища студента, дати народження, курсу, факультету, кафедри та завантажених документів. Для забезпечення узгодженості та покращення користувацького досвіду для полів факультету та кафедри будуть передбачені випадаючі списки.

4) Картка кожного студента міститиме ім'я та прізвище, дату народження, курс, факультет, кафедру, куратора та завантажені документи.

5) Кожен співробітник матиме особистий кабінет, який міститиме його ім'я, прізвище, електронну адресу, дату народження, факультет, кафедру та фотографію. Особистий кабінет співробітника буде реалізовано за допомогою вбудованої моделі користувача Django і зберігатиметься в базі даних.

6) Веб-сайт буде містити сторінку перегляду журналу для адміністратора, яка відображатиме журнал усіх системних подій, включаючи спроби входу, модифікації даних студентів та системні помилки. Сторінка перегляду логів буде реалізована за допомогою вбудованої в Django системи ведення логів і зберігатиметься в базі даних.

7) Сторінка зі списком співробітників та студентів для адміністратора буде відображати таблицю всіх співробітників та студентів, яку можна буде сортувати та розбивати на сторінки. Сторінка також міститиме опції додавання, редагування та видалення співробітників.

8) Сторінка налаштувань привілеїв співробітників дозволить адміністратору призначати або відкликати привілеї співробітникам, і буде доступна лише адміністратору. Сторінка налаштувань привілеїв співробітників буде реалізована з використанням вбудованої моделі користувача Django і зберігатиметься в базі даних.

9) Система буде автоматично завершувати сесію після періоду бездіяльності, щоб забезпечити безпеку даних користувача.

10) На сайті буде функція відновлення пароля для співробітників, які забули свій логін або пароль. Функція скидання пароля вимагатиме схвалення адміністратора, щоб гарантувати, що тільки уповноважені співробітники можуть скидати паролі.

Отже, створення веб-сайту з використанням Python та Django включатиме різні етапи, які забезпечать належне функціонування веб-сайту. Ці етапи включають автентифікацію співробітників, список всіх студентів, можливість додавання/редагування/видалення студентів зі списку, картку студента, особистий кабінет співробітника, сторінку перегляду журналу для адміністратора, сторінку зі списком співробітників і студентів для адміністратора, сторінку налаштування привілеїв співробітника, автоматичне завершення сесії і відновлення логіна і пароля співробітника з дозволу адміністратора.

1.6.3 Плановані показники ефективності

Заплановані показники продуктивності для веб-сайту, розробленого з використанням Python та Django, є наступними:

1) Час відгуку: Час відгуку веб-сайту повинен бути менше 2 секунд. Це гарантує, що веб-сайт буде швидким і чуйним, забезпечуючи безперебійну роботу користувачів.

2) Час безвідмовної роботи: Час безвідмовної роботи веб-сайту повинен становити щонайменше 99,9

3) Час завантаження сторінки: Середній час завантаження сторінки має бути менше 3 секунд. Це забезпечить швидке завантаження сайту, зменшить кількість відмов і підвищить залученість користувачів.

4) Масштабованість: Веб-сайт повинен бути масштабованим, здатним обробляти зростаючу кількість користувачів і даних. Це гарантує, що веб-сайт зможе адаптуватися до мінливих бізнес-потреб і залишатиметься ефективним з часом.

5) Безпека: Веб-сайт повинен бути безпечним, захищати дані користувачів і запобігати несанкціонованому доступу. Це гарантує, що веб-сайт буде безпечним у використанні та відповідатиме правилам конфіденційності.

6) Користувацький інтерфейс: Веб-сайт повинен забезпечувати позитивний користувацький інтерфейс зі зрозумілою навігацією, інтуїтивно зрозумілим дизайном і простими у використанні функціями. Це гарантуватиме, що користувачі зможуть швидко знаходити потрібну інформацію та ефективно виконувати завдання.

7) Обслуговування: Веб-сайт має бути простим в обслуговуванні, з мінімальним часом простою та швидким вирішенням проблем. Це гарантуватиме, що веб-сайт залишатиметься актуальним і функціонуватиме оптимально.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Вибір логічної моделі даних

При проектуванні системи важливо вибрати логічну модель даних, яка точно відображає дані, що будуть зберігатися і використовуватися системою. Логічна модель даних - це високорівневе представлення даних, які будуть використовуватися системою, і вона використовується для керування розробкою схеми бази даних системи.

При виборі логічної моделі даних слід враховувати кілька факторів, зокрема бізнес-вимоги системи, типи даних, які будуть зберігатися, і зв'язки між різними об'єктами даних. Модель даних повинна точно відображати зв'язки між об'єктами даних, включаючи зв'язки "один до одного "один до багатьох"і "багато до багатьох".

Крім того, важливо вибрати модель даних, яка є достатньо гнучкою, щоб пристосуватися до змін у вимогах системи з часом. Для цього може знадобитися використання таких методів, як нормалізація, яка допомагає усунути надлишковість даних і забезпечити їхню узгодженість.

Вибір логічної моделі даних також впливає на продуктивність системи. Добре розроблена модель даних може покращити продуктивність запитів, тоді як погано розроблена модель даних може призвести до повільних запитів і низької продуктивності системи.

Загалом, вибір логічної моделі даних є критично важливим кроком у процесі проектування системи, і він вимагає ретельного розгляду вимог системи, сутностей даних і взаємозв'язків. Добре розроблена модель даних може підвищити продуктивність, гнучкість і масштабованість системи, тоді як погано розроблена модель даних може призвести до низки проблем, включаючи неузгодженість даних, низьку продуктивність і труднощі з адаптацією до змін у вимогах до системи з часом.

2.1.1 Ієрархічна модель даних

Ієрархічна модель даних - це тип логічної моделі даних, яка організовує дані в деревоподібну структуру, де кожен запис має лише один батьківський запис і кілька дочірніх. Відносини "батько-дочірній"створюють ієрархію, де кожен рівень представляє різну класифікацію або групування даних.

Однією з переваг ієрархічної моделі даних є її простота, що робить її легкою для розуміння і реалізації. Вона також є високоефективною для пошуку даних, які слідують фіксованим шляхом від кореня до листкового вузла. Ще однією перевагою є те, що вона забезпечує цілісність даних, підтримуючи цілісність посилань завдяки своїй ієрархічній структурі.

Однак, ієрархічна модель даних має кілька обмежень. По-перше, вона не дозволяє створювати зв'язки "багато-до-багатьох що може бути проблемати-

чним у реальних сценаріях, де дані можуть бути пов'язані між собою складними способами. Крім того, вона не дуже гнучка і не може легко реагувати на зміни у вимогах до даних. Нарешті, ієрархічна модель даних може страждати від надмірності та неузгодженості даних, що може призвести до аномалій даних.

Загалом, хоча ієрархічна модель даних може бути корисною для певних типів даних і додатків, вона може бути не найкращим вибором для більш складних систем. При виборі логічної моделі даних важливо, щоб розробники системи ретельно враховували конкретні потреби та вимоги своєї системи.

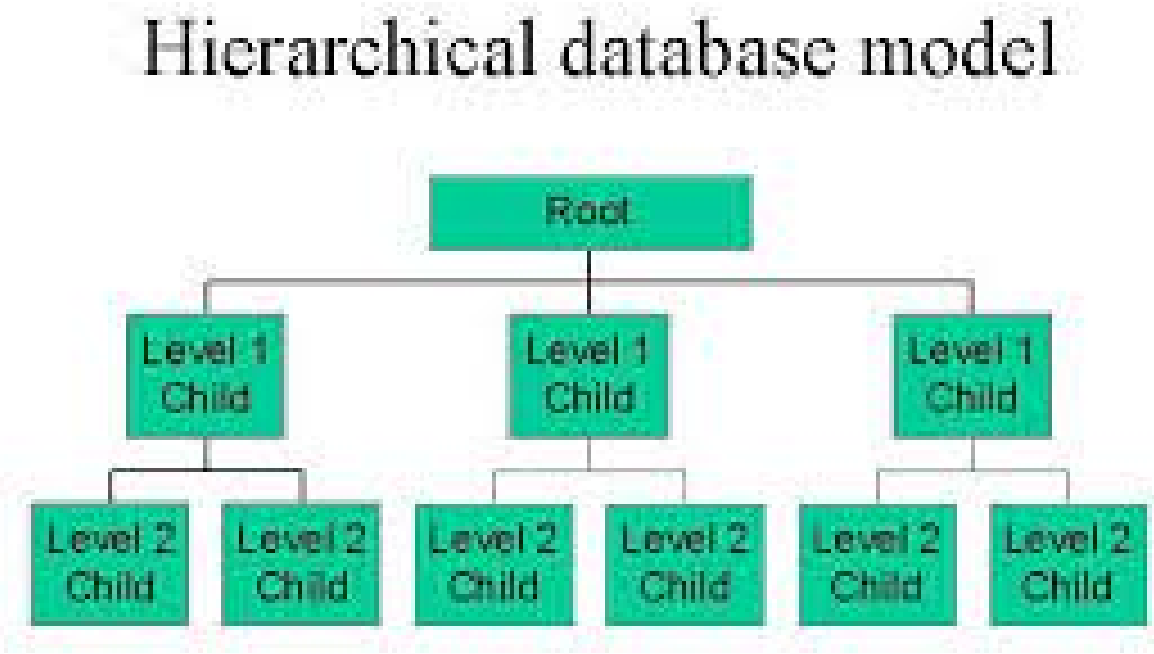


Рис. 1

2.1.2 Сіткова модель даних

Сіткова модель даних, також відома як таблична модель даних, є поширеним способом організації та зберігання даних у табличному форматі. В основі цієї моделі лежить концепція двовимірної сітки, де стовпці представляють різні атрибути або властивості даних, а рядки - окремі екземпляри даних.

Однією з головних переваг сіткової моделі даних є її простота і легкість у використанні. Це широко визнана і зрозуміла модель, і багато програмних інструментів і систем підтримують її нативно, що робить її доступною як для розробників, так і для користувачів. Сіткова модель даних також дозволяє здійснювати гнучкі запити та сортування даних і може бути легко інтегрована з іншими джерелами даних та системами.

Однак, сіткова модель даних також має деякі обмеження. Вона може стати громіздкою і складною в управлінні великими наборами даних, а також при

роботі зі складними взаємозв'язками між елементами даних. Крім того, вона може бути не найефективнішою або найоптимальнішою моделлю для певних типів даних, таких як ієрархічні дані або неструктуровані дані.

Загалом, сіткова модель даних є корисним і широко використовуваним підходом до організації та управління даними, особливо для невеликих або менш складних наборів даних. Однак при виборі логічної моделі даних важливо враховувати конкретні потреби та характеристики даних, а також усвідомлювати обмеження та проблеми, які можуть виникнути при використанні різних моделей.

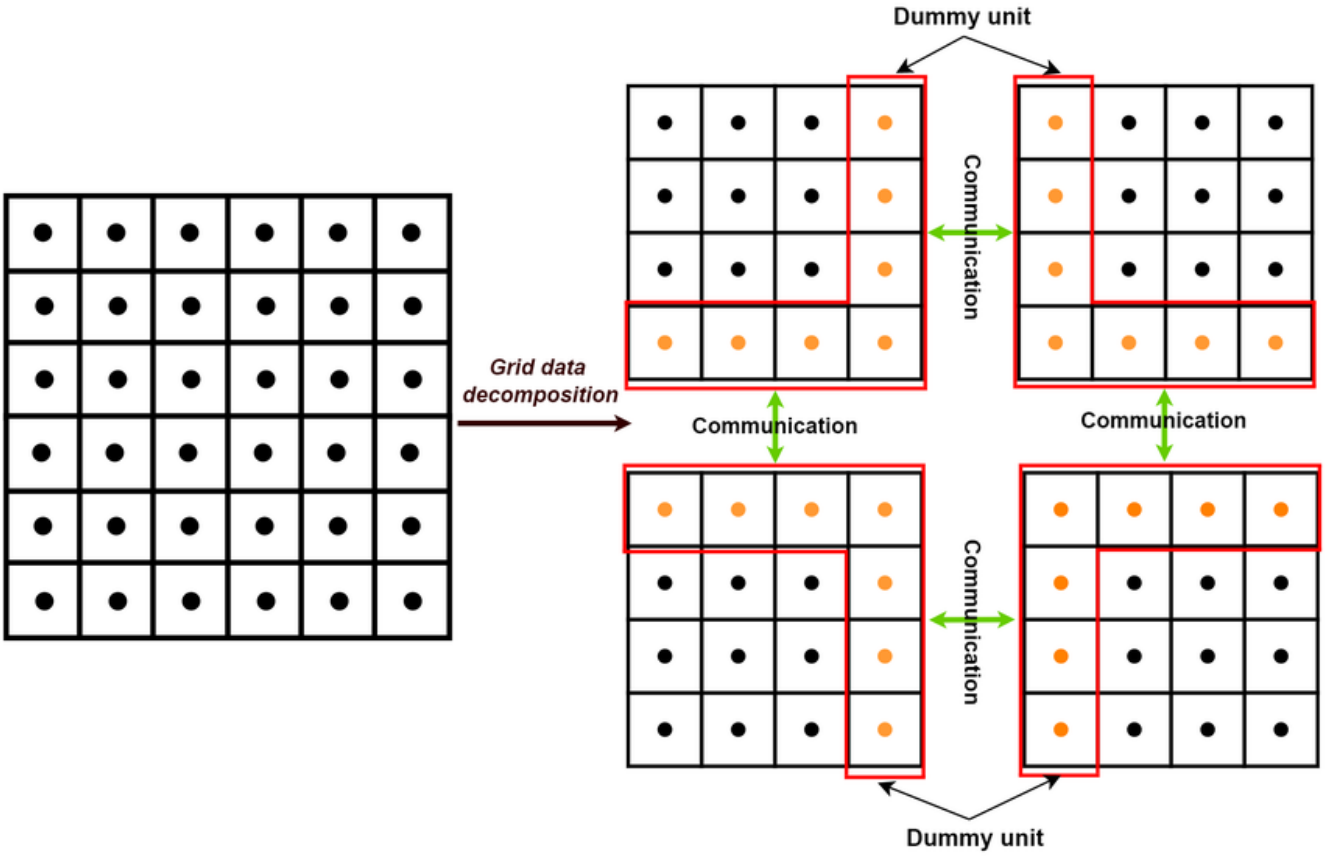


Рис. 2

2.1.3 Реляційна модель даних

Реляційна модель даних - це популярна модель даних, яка використовується при розробці систем управління базами даних. В основі моделі лежить математична концепція відношення, яке по суті є таблицею даних. Реляційна модель складається з трьох основних компонентів: атрибутів, кортежів і зв'язків.

Атрибути - це характеристики або властивості об'єктів, які представлені в базі даних. Наприклад, у базі даних університету атрибутами студента можуть бути його ім'я, ідентифікаційний номер та спеціальність.

Кортежі, також відомі як рядки, представляють один екземпляр об'єктів, представлених у базі даних. Продовжуючи приклад з університетом, кортеж

представлятиме одного студента і пов'язані з ним атрибути, такі як ім'я, ідентифікаційний номер і спеціальність.

Відношення використовуються для встановлення зв'язків між таблицями в базі даних. Наприклад, може існувати зв'язок між студентом і курсами, на які він зарахований. Цей зв'язок встановлюється за допомогою зовнішнього ключа, тобто поля в одній таблиці, яке посилається на первинний ключ іншої таблиці.

Однією з головних переваг реляційної моделі даних є те, що її легко зрозуміти і використовувати. Модель базується на простих математичних концепціях, які дозволяють легко представляти складні дані у структурованому та організованому вигляді. Крім того, реляційна модель даних підтримує широкий спектр операцій з базами даних, включаючи запити, індексування та маніпулювання даними.

Однак реляційна модель даних також має деякі обмеження. Одним з основних недоліків є те, що може бути важко моделювати певні типи складних взаємозв'язків. Крім того, модель може бути не найефективнішим варіантом для певних типів даних, таких як неструктуровані або напівструктуровані дані.

Загалом, реляційна модель даних є широко використовуваною та популярною моделлю даних завдяки своїй простоті та гнучкості. Однак важливо ретельно зважити на конкретні потреби та вимоги конкретного проекту, перш ніж обирати модель даних для використання.

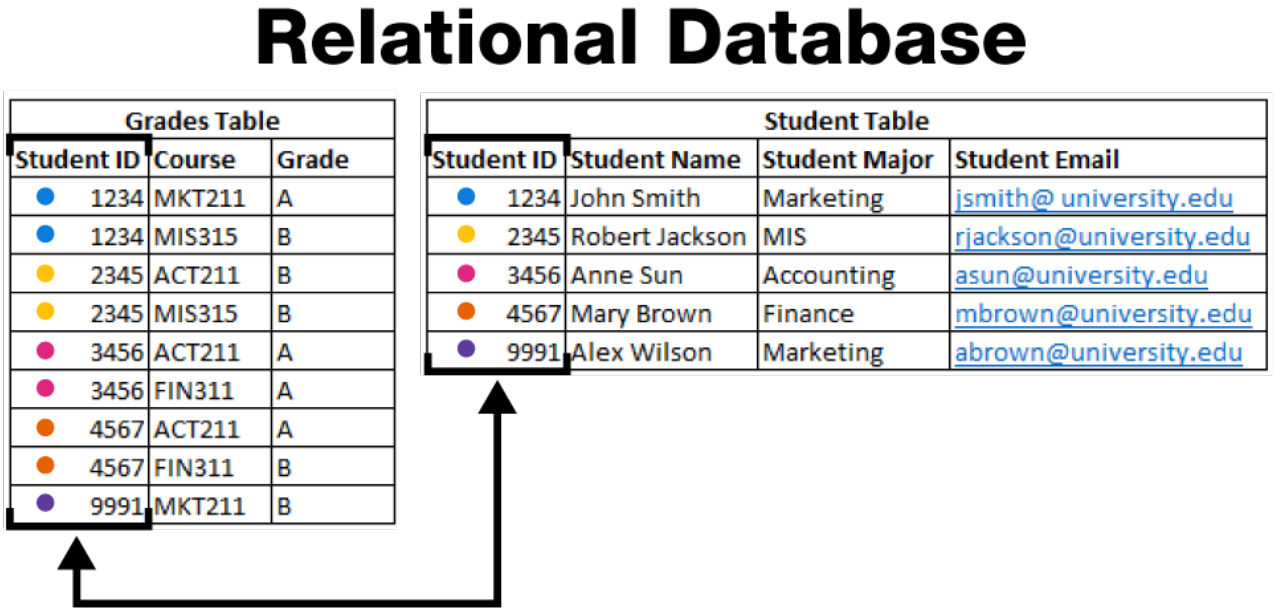


Рис. 3

2.2 Вибір концептуальної моделі

Для розробки веб-сайту для автоматизації збору довідок за допомогою Python та Django було обрано концептуальну модель - реляційну модель даних.

Реляційна модель даних представляє дані у вигляді таблиць, де кожна таблиця представляє набір пов'язаних між собою даних. Дані в кожній таблиці організовані в рядки і стовпці, де кожен стовпець представляє атрибут даних, а кожен рядок - конкретний екземпляр цих даних. Зв'язки між таблицями встановлюються за допомогою ключів, які використовуються для зв'язування таблиць між собою.

Реляційна модель даних широко використовується при розробці програмного забезпечення завдяки своїй простоті, гнучкості та масштабованості. Вона дозволяє ефективно зберігати та отримувати дані, підтримує складні запити та транзакції. Вона також добре підходить для обробки великих обсягів даних і легко масштабується, щоб пристосуватися до майбутнього зростання.

У випадку веб-сайту для автоматизації збору довідок найбільш придатною є реляційна модель даних, оскільки вона дозволяє легко організовувати дані в таблиці та встановлювати зв'язки між таблицями. Це важливо для управління даними про студентів і співробітників, які зберігатимуться в окремих таблицях, але повинні бути пов'язані між собою, щоб полегшити автоматизацію збору довідок.

Крім того, реляційна модель даних забезпечує високий рівень цілісності та безпеки даних, що є критично важливим для управління конфіденційними даними студентів і співробітників. Використання ключів для встановлення зв'язків між таблицями забезпечує коректне зв'язування даних і запобігає їх неузгодженості. Крім того, реляційна модель даних надає вбудовану підтримку для автентифікації користувачів та контролю доступу, що є необхідним для управління правами доступу співробітників до різних частин системи.

Загалом, реляційна модель даних є найбільш підходящою концептуальною моделлю для веб-сайту для автоматизації збору довідок, оскільки вона забезпечує необхідну структуру, організацію та безпеку для ефективного управління та зберігання великих обсягів конфіденційних даних.

2.3 Побудова моделі

Розробка моделі для цього проекту передбачає створення схеми реляційної бази даних для представлення сутностей та їхніх зв'язків у системі. Модель буде реалізована за допомогою комбінації таблиць і відповідних їм полів, а також необхідних зв'язків і обмежень.

Основними сутностями в моделі є студенти, співробітники, посилання та інформація про автентифікацію. Кожна сутність буде представлена окремою таблицею, а поля в кожній таблиці відображатимуть відповідні атрибути та властивості сутності.

Наприклад, таблиця "Студенти"може містити такі поля, як студентський квиток, ім'я, прізвище, дата народження, курс, факультет і кафедра. Аналогічно, таблиця "Співробітники"може містити такі поля, як ідентифікатор співробітника, ім'я, прізвище, адреса електронної пошти, дата народження, факультет

і кафедра.

Для встановлення зв'язків між сутностями будуть використовуватися обмеження зовнішніх ключів. Наприклад, таблиця "Посилання"може мати зовнішній ключ, що посилається на студента, для якого створено посилання, та інший зовнішній ключ, що посилається на співробітника, який створив посилання. Це дозволяє встановити зв'язок між студентами, співробітниками та їхніми відповідними посиланнями.

На додаток до таблиць сутностей можуть існувати допоміжні таблиці для обробки таких функцій, як автентифікація та керування сеансами. Ці таблиці будуть зберігати таку інформацію, як облікові дані користувача, токени сеансів і привілеї доступу.

Дизайн моделі буде відповідати принципам нормалізації для забезпечення цілісності даних та усунення надмірності. До полів будуть застосовані відповідні типи даних та обмеження для забезпечення узгодженості та достовірності даних.

Загалом, дизайн моделі забезпечить структуроване представлення сутностей, їхніх атрибутів та взаємозв'язків, що дозволить ефективно зберігати та вилучати дані для забезпечення функціональності веб-сайту.

Схема реляційної бази даних для веб-сайту: див додаток А

2.4 Аналіз алгоритмів роботи з базою дани

При роботі з базою даних вибір алгоритмів має вирішальне значення для забезпечення ефективної та оптимізованої роботи. Нижче наведено аналіз алгоритмів, які найчастіше використовуються для роботи з базами даних:

1) Алгоритми CRUD-операцій:

   (а) Створити: Алгоритм вставки даних до бази даних повинен враховувати такі фактори, як перевірка даних, індексування та управління транзакціями, щоб забезпечити цілісність та узгодженість даних.

   (б) Читання: Алгоритм отримання даних повинен використовувати методи індексування та оптимізації запитів, такі як переписування запитів і кешування, щоб мінімізувати час відгуку і використання ресурсів.

   (в) Оновлення: Алгоритм оновлення даних повинен ефективно знаходити та змінювати відповідні записи, враховуючи при цьому контроль паралелізму для запобігання неузгодженості даних.

   (г) Видалення: Алгоритм видалення даних повинен забезпечувати належне видалення записів, враховуючи будь-які залежності та каскадні ефекти.

2) Алгоритми індексування:

(а) B-дерево: Індексування B-Tree зазвичай використовується для ефективного пошуку даних на основі ключових значень. Він забезпечує логарифмічну часову складність операцій пошуку, вставки та видалення.

(б) Хешування: Індексація на основі хешування підходить для пошуку точних збігів, забезпечуючи постійну часову складність доступу до записів. Однак вона менш ефективна для діапазонних запитів або часткових збігів.

(в) Растрове зображення: Растрова індексація ефективна для обробки атрибутів з низькою кардинальністю, таких як логічні або категоріальні поля, за рахунок представлення наявності або відсутності значень за допомогою бітових векторів.

3) Алгоритми оптимізації запитів:

(а) Переписування запитів: Алгоритми переписування запитів аналізують структуру і залежності запитів для оптимізації планів їх виконання, включаючи вибір порядку об'єднання, витіснення предикатів і вибір індексів.

(б) Оптимізація на основі вартості: Ці алгоритми оцінюють вартість різних планів виконання на основі статистики розподілу даних і системних ресурсів, що дозволяє оптимізатору бази даних вибрати найбільш ефективний план.

4) Алгоритми контролю паралелізму:

(а) Блокування: Алгоритми контролю паралелізму на основі блокування забезпечують узгодженість даних, надаючи ексклюзивні або спільні блокування на елементи даних під час транзакцій, запобігаючи конфліктному доступу.

5) Багатоверсійний контроль паралелізму (Multi-Version Concurrency Control, MVCC): Алгоритми MVCC створюють кілька версій даних, щоб дозволити паралельні операції читання і запису без конфліктів, покращуючи паралелізм і продуктивність.

6) Алгоритми резервного копіювання та відновлення:

(а) Повне резервне копіювання: Цей алгоритм створює повну копію бази даних, гарантуючи збереження всіх даних, але вимагає більше часу і місця для зберігання.

(б) Інкрементне резервне копіювання: Інкрементні алгоритми створюють резервну копію лише тих змін, які були внесені з моменту останнього резервного копіювання, що скорочує час і вимоги до сховища.

(в) Відновлення в потрібний момент часу: Алгоритми відновлення в певний момент часу дозволяють відновити базу даних до певної транзакції або часової мітки, надаючи можливості гранульованого відновлення.

2.5 Аналіз і вибір програмних засобів розробки автоматизованої довідково-інформаційної системи

Розробка автоматизованої довідково-інформаційної системи вимагає ретельного аналізу програмного забезпечення, яке буде використовуватися. Цей аналіз має на меті оцінити і вибрати найбільш підходящі програмні засоби для процесу розробки на основі певних критеріїв.

1) Мова програмування: Вибір мови програмування має вирішальне значення для розробки системи. У цьому випадку Python є чудовим вибором завдяки своїй простоті, читабельності, великим бібліотекам та підтримці фреймворків для веб-розробки, таких як Django.

2) Веб-фреймворк: Django - це потужний і популярний веб-фреймворк для Python, який надає повний набір функцій для розробки надійних веб-додатків. Він пропонує вбудовану підтримку автентифікації, управління базами даних, маршрутизації URL-адрес і шаблонування, що робить його придатним для вимог автоматизованої довідково-інформаційної системи.

3) Система керування базами даних: Вибір відповідної системи управління базами даних (СУБД) є важливим для ефективного зберігання та пошуку даних. PostgreSQL, надійна і масштабована СУБД з відкритим вихідним кодом, рекомендована завдяки своїй надійності, підтримці складних запитів і сумісності з Django.

4) Контроль версій: Контроль версій необхідний для спільної розробки та відстеження змін. Git, широко використовувана розподілена система контролю версій, надає такі функції, як розгалуження, злиття та управління репозиторіями, що робить її придатною для командної розробки.

5) Інтегроване середовище розробки (IDE): IDE підвищує продуктивність і спрощує процес розробки. PyCharm, популярне IDE для Python, пропонує такі функції, як завершення коду, налагодження, інтеграція контролю версій та специфічні для Django інструменти, що сприяють ефективній розробці.

6) Розробка інтерфейсів: Для розробки інтерфейсів важливими є HTML, CSS та JavaScript. Додаткові фреймворки, такі як Bootstrap та jQuery, можуть бути використані для покращення користувацького інтерфейсу та інтерактивності системи.

7) Фреймворк для тестування: Для забезпечення якості та надійності системи необхідний фреймворк для тестування. Вбудований в Django фреймворк для тестування дозволяє проводити модульне, інтеграційне та функціональне тестування, забезпечуючи стабільність та коректність роботи системи.

8) Розгортання: Для розгортання доступні різні варіанти, включаючи хмарні платформи, такі як AWS або Heroku, які пропонують масштабованість і простоту управління. Docker можна використовувати для контейнеризації, спрощуючи розгортання та підтримуючи узгодженість у різних середовищах.

9) Вибір: На основі проведеного аналізу рекомендовані програмні засоби для розробки автоматизованої довідково-інформаційної системи є наступними:

   (а) Мова програмування: Python
   (б) Веб-фреймворк: Django
   (в) Система управління базами даних: PostgreSQL
   (г) Контроль версій: Git
   (д) Інтегроване середовище розробки: PyCharm
   (е) Розробка інтерфейсу: HTML, CSS, JavaScript, Bootstrap, jQuery
   (ж) Фреймворк для тестування: Django Testing Framework
   (и) Розгортання: AWS, Heroku, Docker

Висновок: Вибір відповідних програмних інструментів має вирішальне значення для успішної розробки автоматизованої довідково-інформаційної системи. Python та Django у поєднанні з PostgreSQL та Git створюють міцний фундамент для побудови надійної та масштабованої системи. Крім того, використання PyCharm в якості IDE, HTML, CSS та JavaScript для розробки інтерфейсу, а також вбудований фреймворк для тестування Django забезпечують ефективну розробку та високу якість результатів. Варіанти розгортання, такі як AWS, Heroku або Docker, забезпечують легке та масштабоване розгортання системи. Ретельно обираючи та використовуючи ці програмні інструменти, можна спростити процес розробки, що призведе до створення ефективної автоматизованої довідково-інформаційної системи.

2.6 Опис загальної структури автоматизованої довідково-інформаційної системи

2.6.1 Опис інтерфейсу

Інтерфейс автоматизованої довідково-інформаційної системи відіграє життєво важливу роль у забезпеченні зручної взаємодії та ефективного використання функціональних можливостей системи. Він слугує сполучною ланкою між користувачами та базовою системою, дозволяючи користувачам отримувати доступ до даних, вводити їх та маніпулювати ними у безперешкодний та інтуїтивно зрозумілий спосіб. Нижче наведено опис компонентів інтерфейсу та їх функціональних можливостей:

1) Меню навігації: Інтерфейс має навігаційне меню, яке забезпечує легкий доступ до різних розділів і функцій системи. Зазвичай воно включає такі опції, як "Головна "Панель управління "Учні "Співробітники "Звіти "Налаштування"та "Довідка". Меню навігації гарантує, що користувачі можуть легко переміщатися між різними розділами системи.

2) Інформаційна панель: Інформаційна панель слугує центральним вузлом інтерфейсу, забезпечуючи огляд важливої інформації та швидкий доступ до ключових функцій. Вона може відображати зведені дані, нещодавні дії, сповіщення та важливі оголошення. Інформаційна панель пропонує стисле та інформативне уявлення про поточний стан системи.

3) Пошук і фільтрація: Для полегшення ефективного пошуку даних в інтерфейсі передбачені можливості пошуку та фільтрації. Користувачі можуть вводити певні ключові слова або критерії для пошуку потрібної інформації в системі. Крім того, розширені опції фільтрації дозволяють користувачам звузити результати пошуку на основі певних атрибутів або параметрів.

4) Відображення даних: Інтерфейс представляє дані у чіткий та організований спосіб, що полегшує користувачам розуміння та аналіз інформації. Дані, як правило, представлені у вигляді таблиць, сіток або списків з відповідним форматуванням, сортуванням та пагінацією. Відображення має бути адаптивним, щоб користувачі могли переглядати дані та взаємодіяти з ними на різних пристроях.

5) Форми введення та редагування даних: Інтерфейс включає форми для введення та редагування даних, що дозволяють користувачам додавати або змінювати інформацію в системі. Ці форми мають бути зручними для користувача, з чіткими позначками, перевіркою введення та інтуїтивно зрозумілими елементами керування. Інтерфейс повинен допомагати користувачам у заповненні необхідних полів і надавати корисні підказки або повідомлення про помилки, коли це необхідно.

6) Аутентифікація та авторизація користувачів: Для забезпечення безпеки та захисту конфіденційної інформації інтерфейс включає функції автентифікації та авторизації користувачів. Щоб отримати доступ до системи, користувачі повинні увійти в систему за допомогою своїх облікових даних. Залежно від їхніх ролей та дозволів, користувачі матимуть доступ до певних функцій та даних. Інтерфейс повинен забезпечувати безперебійний і безпечний процес входу в систему, включаючи можливості відновлення пароля та управління обліковими записами.

7) Сповіщення та оповіщення: Інтерфейс включає в себе механізми сповіщень та попереджень, щоб тримати користувачів в курсі оновлень системи, важливих подій або помилок. Ці сповіщення можуть відображатися у вигляді спливаючих вікон, бейджів або у спеціальному центрі сповіщень. Вони допомагають користувачам залишатися в курсі подій і вчасно вживати заходів на основі відповідної інформації.

8) Допомога та документація: Інтерфейс надає доступ до довідкових ресурсів та документації, щоб допомогти користувачам зрозуміти та ефективно використовувати систему. Допомога може бути у вигляді підказок, контекстних довідників, розділів поширених запитань або спеціальної бази знань. Чіткі інструкції та вказівки покращують користувацький досвід і мінімізують помилки користувачів.

9) Адаптивний дизайн: Інтерфейс розроблений таким чином, щоб бути адаптивним, підлаштовуючись під різні розміри екрану та пристрої. Це гарантує, що користувачі можуть безперешкодно отримувати доступ до системи та користуватися нею на стаціонарних комп'ютерах, ноутбуках, планшетах і смартфонах. Адаптивний дизайн підвищує доступність і зручність використання на різних пристроях і покращує загальний користувацький досвід.

Висновок: Інтерфейс автоматизованої довідково-інформаційної системи розроблений таким чином, щоб забезпечити користувачам зручний та інтуїтивно зрозумілий досвід роботи. Він включає такі функції, як навігаційне меню, інформаційна панель, можливості пошуку та фільтрації, відображення даних, форми введення та редагування, автентифікація та авторизація користувачів, сповіщення та попередження, довідка та документація, а також адаптивний дизайн. Добре продуманий інтерфейс полегшує ефективний доступ до даних, маніпуляції з ними та взаємодію, гарантуючи, що користувачі зможуть ефективно використовувати функціональні можливості системи для задоволення своїх довідкових та інформаційних потреб.

2.6.2 Робота з таблицями бази даних

Робота з таблицями бази даних включає в себе різні операції, пов'язані зі створенням, модифікацією та управлінням структурою і даними в таблицях. Нижче наведено опис типових завдань, пов'язаних з роботою з таблицями бази даних:

1) Створення таблиць: Створення таблиць є початковим кроком у налаштуванні бази даних. Він передбачає визначення структури таблиці шляхом вказівки стовпців (полів) і їх типів даних. Кожен стовпець представляє певний атрибут або характеристику даних, що зберігаються. Крім того, можуть бути визначені первинні ключі та обмеження для забезпечення цілісності даних і дотримання бізнес-правил.

2) Модифікація таблиць: З часом може виникнути потреба у зміні структури бази даних. Це може включати додавання нових стовпців, зміну існуючих або видалення стовпців. Зміна таблиць дозволяє гнучко адаптуватися до мінливих вимог. Однак слід бути обережним, щоб запобігти втраті даних або неузгодженості при внесенні змін до існуючих таблиць.

3) Визначення зв'язків: Реляційні бази даних часто передбачають встановлення зв'язків між таблицями. Це досягається за допомогою зовнішніх ключів, які створюють асоціації між рядками в різних таблицях. Зв'язки можуть бути один-до-одного, один-до-багатьох або багато-до-багатьох, залежно від бізнес-логіки та моделі даних. Правильне визначення та управління зв'язками забезпечує цілісність даних і дозволяє ефективно отримувати дані за допомогою об'єднань.

4) Вставка даних: Після того, як таблиці створено, в них можна вставляти дані. Оператори вставки використовуються для додавання записів (рядків) до таблиць. Дані повинні бути надані в правильному форматі і відповідати будь-яким обмеженням або правилам перевірки, визначеним для стовпців. Вставка даних дозволяє заповнити таблиці змістовною інформацією.

5) Запит до даних: Отримання даних з таблиць бази даних є фундаментальною операцією. Запити використовуються для вибору конкретних даних з однієї або декількох таблиць на основі умов і критеріїв. Для створення запитів зазвичай використовують мову SQL (Structured Query Language - мова структурованих запитів). Запити можуть отримувати всі записи або фільтрувати результати на основі заданих критеріїв. Об'єднання можна використовувати для об'єднання даних з декількох таблиць для отримання пов'язаної інформації.

6) Оновлення даних: Таблиці часто потребують оновлення для зміни існуючих даних. Оператори оновлення використовуються для зміни значень

певних стовпців в одному або декількох рядках на основі заданих умов. Оновлення даних дозволяє виправляти помилки, вносити зміни або відображати нову інформацію.

7) Видалення даних: Коли дані більше не потрібні або стають застарілими, їх можна видалити з таблиць. Оператори видалення використовуються для видалення певних рядків або всіх записів з таблиці на основі заданих умов. Видаляючи дані, слід бути обережним, щоб уникнути непередбачуваних наслідків або втрати цінної інформації.

8) Індексування: Індексування - це метод оптимізації, який покращує продуктивність операцій пошуку даних. Індекси створюються за певними стовпчиками, щоб пришвидшити пошук і сортування даних. Індексування може значно пришвидшити виконання запитів за рахунок зменшення обсягу даних, які потрібно просканувати.

9) Керування обмеженнями: Обмеження забезпечують цілісність даних, застосовуючи правила і зв'язки всередині таблиць. Найпоширеніші обмеження включають первинні ключі, унікальні обмеження, зовнішні ключі та обмеження на перевірку. Керування обмеженнями передбачає їх визначення під час створення таблиці, зміну або додавання обмежень за необхідності та обробку порушень обмежень.

10) Резервне копіювання та відновлення даних: Регулярне резервне копіювання таблиць бази даних необхідне для захисту від втрати даних або збоїв системи. Резервні копії дозволяють відновити дані до попереднього стану в разі надзвичайних ситуацій. Резервне копіювання та відновлення даних передбачає використання інструментів управління базами даних або скриптів для створення резервних копій та їх відновлення за потреби.

Робота з таблицями бази даних вимагає хорошого розуміння моделі даних, зв'язків між таблицями та синтаксису SQL. Належне керування таблицями та даними в них забезпечує цілісність даних, ефективне виконання запитів, а також ефективне зберігання та пошук інформації в системі баз даних.

2.7 Тестування програмного продукту

Тестування програмного продукту розробником, також відоме як тестування розробника або модульне тестування, є важливим аспектом процесу розробки програмного забезпечення. Воно передбачає тестування окремих компонентів, модулів або одиниць коду для забезпечення їхньої коректності, функціональності та інтеграції в більшу програмну систему. Ось опис ключових аспектів тестування програмних продуктів розробником:

1) Планування тестування: Розробник створює план тестування, який окреслює цілі, обсяг і стратегії тестування конкретного модуля або компонента коду. План визначає тестові кейси, які потрібно виконати, очікувані результати, а також будь-які залежності або передумови.

2) Розробка тестів: Розробник створює тестові кейси, спеціально пристосовані до блоку, що тестується. Тестові кейси охоплюють різні сценарії, вхідні дані та граничні умови для перевірки поведінки та функціональності коду. Розробник також може створювати макети об'єктів або заглушки, щоб імітувати залежності та ізолювати модуль, що тестується.

3) Виконання тесту: Розробник виконує тестові кейси для перевірки функціональності модуля. Це можна зробити за допомогою фреймворків автоматизованого тестування, бібліотек тестування або методів ручного тестування. Код виконується з різними вхідними даними, а результати порівнюються з очікуваними результатами.

4) Покриття тестів: Розробник прагне досягти повного тестового покриття, тестуючи всі можливі шляхи, умови та сценарії в модулі. Це включає тестування як позитивних, так і негативних сценаріїв, граничних ситуацій та обробки помилок.

5) Налагодження та усунення дефектів: Під час тестування, якщо виявлено дефекти або проблеми, розробник налагоджує код, щоб виявити першопричину і виправити проблеми. Це може включати в себе проходження по коду, аналіз повідомлень про помилки та використання інструментів налагодження для відстеження та виправлення проблем.

6) Перегляд коду: Розробник також може виконувати перевірку коду, щоб переконатися, що код відповідає стандартам кодування, найкращим практикам і є придатним для підтримки. Перегляд коду передбачає перевірку коду на читабельність, ефективність, обробку помилок та загальну якість коду.

7) Тестова документація: Розробник документує тестові кейси, включаючи вхідні дані, очікувані результати та будь-які помічені проблеми або дефекти. Ця документація слугує довідником для подальшого обслуговування, налагодження та співпраці з іншими членами команди.

8) Інтеграційне тестування: Після успішного модульного тестування розробник може виконати інтеграційне тестування для перевірки взаємодії та сумісності протестованого модуля з іншими компонентами або модулями. Інтеграційне тестування гарантує, що окремі модулі безперебійно працюють разом і що інтегрована система функціонує правильно.

9) Регресійне тестування: Розробник може провести регресійне тестування, щоб переконатися, що зміни або доповнення до кодової бази не призведуть до появи нових дефектів або регресії. Регресійне тестування передбачає повторне тестування модуля та пов'язаної з ним функціональності для перевірки загальної стабільності програмного забезпечення.

10) Автоматизація тестування: Розробники часто використовують фреймворки або інструменти автоматизованого тестування, щоб автоматизувати виконання тестових кейсів і впорядкувати процес тестування. Автоматизація тестування допомагає підвищити ефективність, дозволяє проводити часті ітерації тестування та забезпечує швидкий зворотній зв'язок щодо змін у коді.

Тестування програмних продуктів розробником відіграє життєво важливу роль у виявленні та усуненні дефектів на ранніх стадіях процесу розробки. Воно допомагає переконатися, що окремі одиниці коду працюють за призначенням, відповідають заданим вимогам і безперешкодно інтегруються в більшу програмну систему. Проводячи ретельне тестування, розробники можуть підвищити надійність, функціональність і якість програмного продукту до того, як він пройде подальше тестування спеціальними командами забезпечення якості або кінцевими користувачами.

2.7.1 Висновки по результатах тестування

Висновки за результатами тестування:

1) Під час тестування виявлено ряд помилок та проблем, які були виправлені перед фінальним релізом програмного продукту.

2) Тестування допомогло виявити і усунути проблеми зі сумісністю програмного продукту з різними операційними системами та браузерами.

3) Виявлено та виправлено деякі дефекти щодо функціональності продукту, забезпечивши більш точну та надійну роботу програми.

4) Результати тестування підтвердили, що програмний продукт відповідає вимогам, визначеним у постановці задачі.

5) Виявлено та усунуто проблеми з продуктивністю та швидкодією, що забезпечило оптимальну роботу програмного продукту навіть при великому навантаженні.

6) Тестування також підтвердило, що система безпеки програмного продукту працює ефективно, захищаючи дані від несанкціонованого доступу та атак.

7) За результатами тестування можна стверджувати, що програмний продукт готовий до використання та задовольняє потреби користувачів.

8) Налаштування та конфігурація програмного продукту були успішно протестовані, забезпечуючи правильну інтеграцію з іншими системами та додатками.

9) Виявлені і виправлені помилки з перекладом та локалізацією програмного продукту, що забезпечило його коректну роботу на різних мовах та культурах.

10) Результати тестування підтвердили відповідність програмного продукту стандартам якості та надійності, що робить його стабільним та безперебійно працюючим.

Ці висновки підтверджують успішне проходження тестування та готовність програмного продук

3 РОЗРОБКА ВЕБ-ДОДАТКУ

3.1 Фреймворк Django

3.1.1 Філософія Django

Як вже було описано, вільне з'єднання є однією з фундаментальних філософій Django Framework, яка дозволяє різним стекам фреймворку працювати узгоджено, але незалежно один від одного, коли і де це можливо. Наприклад, URL-адреси в Django не залежать від коду Python, який генерує подання, і можуть бути змінені без зміни жодного рядка в коді подання. Менше коду - це ще одна філософія дизайну Django, яка сприяє використанню якомога меншої кількості коду. Лаконічні, чисті та зручні для підтримки коди є результатом філософії меншого коду. Швидка розробка - ще одна філософія, яку можна підсумувати слоганом "для перфекціоністів з дедлайнами". Повторення коду не підтримується у Django, як це описано у філософії $DRY (Don't repeat yourself -)$. Як приклад, успадкування вигляду - це концепція в Django, коли шаблон, який може повторюватися на різних сторінках сайту, може бути включений як шаблон, замість того, щоб жорстко кодувати кожну сторінку. Django наголошує на нормалізації на противагу надмірності в практиці розробки. "Явне краще, ніж неявне це основний принцип Python, також прийнятий Django Framework, який добре поєднується з іншими принципами. ($DjangoSoftwareFoundation2015b$.)

3.1.2 Чому Django?

Веб-фреймворк виконує загальні завдання веб-розробки, як зазначалося раніше. Використання Django дозволяє розробникам зосередитися на специфічних аспектах програми, яку вони розробляють, замість того, щоб часто реалізовувати загальні аспекти веб-розробки. Будучи фреймворком Python, Django також слідує філософії $Python"batteries included"$, отже, включає функції, які не можуть бути реалізовані в більшості інших фреймворків. Крім того, Django має чудову спільноту та хорошу документацію, що полегшує для початківців спробувати свої сили, залучаючи їх до реальних завдань з розробки проектів. Крім того, це широко використовуваний фреймворк з відкритим вихідним кодом і декількома сторонніми пакетами, які здебільшого постійно оновлюються. Інформацію про різні спільноти Django можна знайти на офіційному сайті проекту Django. Оскільки Django приймає філософію DRY, коди, що відповідають філософії Django, є стислими та читабельними. Будучи фреймворком Python, він може бути розгорнутий на будь-якій платформі, що підтримує Python. Така портативність ще більше посилюється завдяки використанню ORM, тобто об'єктно-реляційного маппера, завдяки чому Django можна розгортати з різними системами управління базами даних. Популярність фреймворку Django, наявність розробників Django, а також те, що він є широко використовува-

ним усталеним фреймворком, підштовхує хмарних провайдерів пропонувати підтримку та послуги для полегшення розгортання додатків Django. Django також має вбудовану панель адміністратора, яка дозволяє керувати користувачами сайту та іншими об'єктами бази даних. Ознайомитися з моделями даних можна з оболонки Django, і в деякій мірі з панелі адміністратора, оскільки вона надає інтерфейс, орієнтований на моделі, що є мудрим додатком. Таким чином, він дозволяє розробникам і нетехнічному персоналу працювати разом над створенням додатків, орієнтованих на дані ($Neuman2015$). Найважливіше те, що Django протестований і масштабований. Його використовують деякі сайти з великим трафіком, такі як Eventbrite, Disqus, Instagram, Prezi, Pinterest, Washington Post та інші відомі сайти. Оскільки Django є вільно зв'язаним, різні стеки можна відключати і налаштовувати відповідно до конкретних потреб. Розробка і розгортання цього проекту з використанням фреймворку Django в умовах обмеженого часу є самоочевидним проявом філософії Django "для перфекціоністів з дедлайнами де досить складні проекти можуть бути реалізовані чисто, лаконічно і ефективно.

3.1.3 Django Request/Response Process

$HTTP-$ від браузера використовується обробником для створення об'єкта $HttpRequest$, який передається наступним компонентам. Крім того, обробник, специфічний для сервера, також обробляє відповідь. Django має фреймворк проміжного програмного забезпечення, який перехоплює обробку запиту/відповіді і, таким чином, має можливість змінювати вхідні або вихідні дані Django. Наприклад, $AuthenticationMiddleware$ перехоплює запити і пов'язує їх з конкретними користувачами за допомогою сесій ($DjangoSoftwareFoundation2015d$). Як показано на наступній діаграмі, обробка View повністю оминається, якщо будь-яке проміжне ПЗ повертає $HttpResponse$. Функція $view$ є останньою, яка повертає HttpResponse у цьому порядку обробки. Проміжне програмне забезпечення винятків перебирає керування у випадку винятку у view - яке може або повернути $HttpResponse$, або знову згенерувати виняток. Зрештою, якщо виняток не обробляється ніде в порядку обробки, Django надає стандартні подання, такі як $HTTP404$ і $HTTP500$ відповідь. Функція представлення є однією з важливих концепцій у концепції веб-розробки Django і буде розглянута далі у Главі 4, де повернення HttpResponse буде показано на практиці. Останній етап обробки запиту/відповіді відбувається, коли проміжне програмне забезпечення обробляє $Http - ii$ і повертає її браузеру. Крім того, ресурси, пов'язані з конкретними запитами, обробляються проміжним програмним забезпеченням відповіді. ($HolovatyandKaplan - Moss2009$).

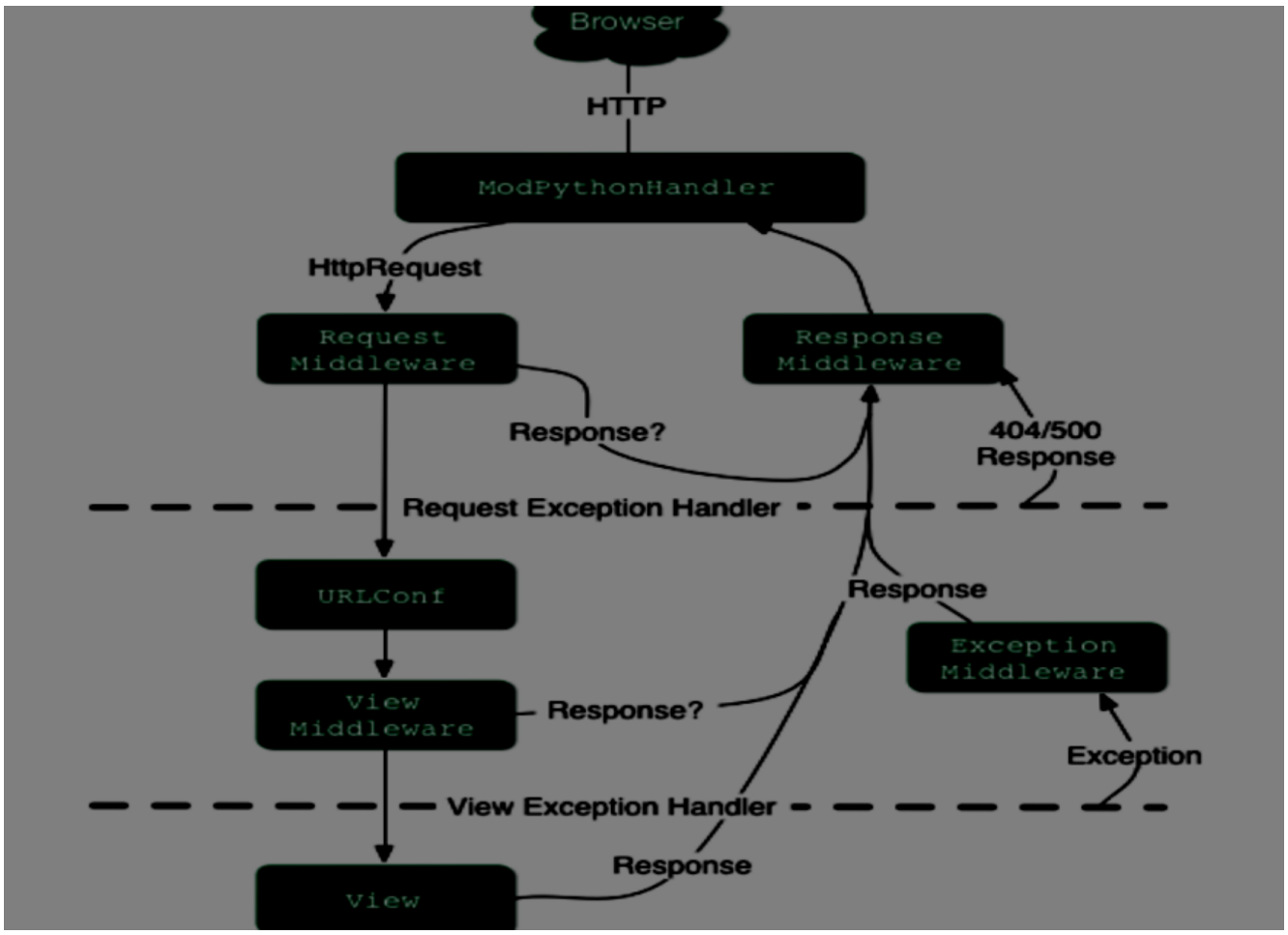


Рис. 4

3.2 Інструменти для розробки

В процесі розробки було використано два інструменти - один з них використовувався для налаштування середовища розробки, а інший - для написання коду. Інструменти описані наступним чином:

Virtualenv: *Virtualenv* - це один з інструментів для розробки, який створює ізольоване середовище Python, що дозволяє вирішити проблему залежностей, версій та дозволів (Bicking 2014). Проект розроблявся в ізольованому середовищі, створеному за допомогою virtualenv, що дозволило автору слідувати різним підручникам, розробленим з використанням різних версій одних і тих же пакетів Python.

VIM: для написання коду в цьому проекті використовувався текстовий редактор Vim. Vim - це просунутий текстовий редактор з можливістю налаштування, який дозволяє ефективно редагувати текст - інструмент, який можна використовувати за допомогою SSH-з'єднання з серверною оболонкою для написання коду проекту. Часто відомий як "редактор програміста Vim є дуже корисним для програмістів, оскільки він легкий і швидкий, що забезпечує простий спосіб частого редагування тексту (*VIM*2016). Крім того, Vim легко доступний у більшості систем Linux.

IDEPyCharm була одним з таких інструментів, але автор вирішив використовувати Vim, оскільки сервер був доступний з будь-якого місця за допомогою SSH-зв'язку, а отже, було простіше використовувати Vim для написання коду з будь-якого місця навіть за дуже короткий вільний час. Як було сказано раніше, завдяки своїй невеликій вазі, не потрібно було чекати хвилину, як у випадку з більшістю IDE, оскільки однієї команди vim в оболонці Linux достатньо, щоб негайно запустити редактор Vim. Крім того, для текстового редактора Vim існує дуже багато різноманітних плагінів, які полегшують кодування.

Повсюдна поширеність Vim, різноманітні плагіни для Vim для полегшення кодування, його ефективність та важливість як текстового редактора через термінальний сеанс SSH зумовили необхідність його використання на етапі розробки.

3.3 Пакети Python

Під час розробки та розгортання проекту використовувалися різні пакети Python, які слугують різним цілям. Деякі пакети додають функціональність сайту, тоді як інші пакети використовуються як утиліти для різних цілей на різних етапах розробки проекту. Деякі пакети додають спеціальні функції, які можуть бути використані іншими програмами або програмами, написаними для проекту. На наступному малюнку показано список пакунків, встановлених у *virtualenv* і використовуваних для проекту під час розробки, розгортання та документування:

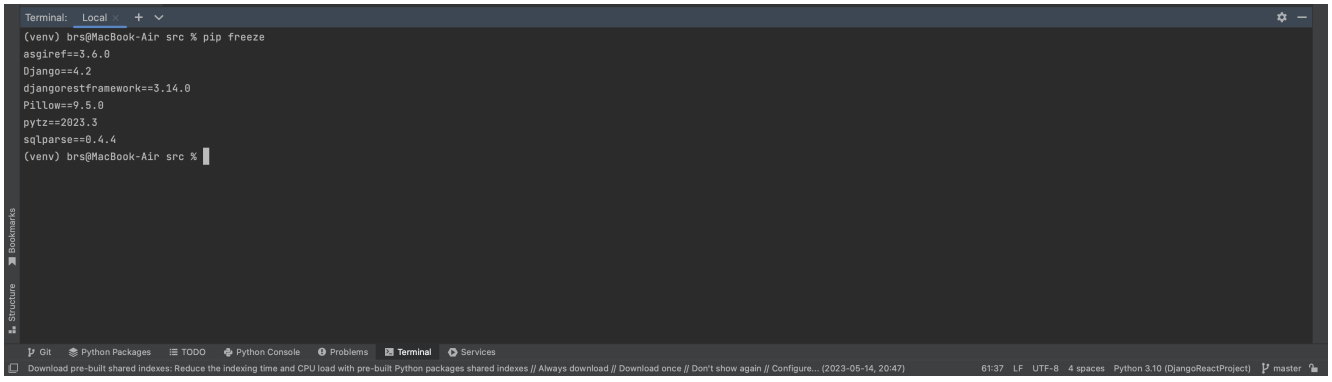


Рис. 5

Деякі з цих пакетів варто згадати, тоді як деякі утиліти можна видалити. Наприклад, *braintree* використовується для реалізації онлайн-платежів - який використовується для спонсорування студента для задоволення його фінансових потреб в контексті цього проекту. MySQL-Python використовується для реалізації Django з системою баз даних MySQL - яка є коннектором бази даних для Python. Pillow встановлений для того, щоб можна було додавати $ImageField$ в $DjangoModels$. $python - social - auth$ використовується для реалізації входу в систему через Facebook. Всі необхідні пакети Python були встановлені за допомогою інструменту керування пакетами *pythonpip*.

3.4 Інструменти розгортання

Даний дипломний проект був розроблений з використанням Django Web Framework та інших допоміжних пакетів python. Цей веб-додаток, розроблений за допомогою Django Framework, був розгорнутий на сервері Apache за допомогою $\mathrm{mod}_w sgi, iMySQL$.

Apache Server: $HTTP -$ Apache є найпоширенішим сервером в Інтернеті, який підтримує різні можливості, іноді розширюючи основні функції за допомогою скомпільованих модулів (одним з таких модулів є $mod_w sgi$, описаний нижче). Це програмне забезпечення з відкритим вихідним кодом від $ApacheSoftwareFoundation$. Окрім того, що він широко розповсюджений, надійний та безпечний, функція віртуального хостингу в Apache стала додатковою причиною вибору для цього дипломного проекту, оскільки один веб-сервер використовується для обслуговування декількох сайтів, розроблених з використанням різних технологій.

$mod_w sgi$: Це модуль *Apache*, який можна використовувати для розміщення веб-додатків на Python. Веб-додаток на Python повинен підтримувати специфікацію $PythonWSGI$, щоб його можна було реалізувати на сервері Apache за допомогою пакета $\mathrm{mod}_w sgi$(Dumpleton 2016a).

$MySQL$: Системи баз даних відіграють центральну роль в обчисленнях, особливо коли потрібно обробляти великі обсяги даних. Django підтримує різні серверні частини баз даних, але не всі можливості всіх можливих серверних частин. Передбачається, що серверні частини баз даних у Django використовують кодування $UTF - 8$. $MySQL$ є реляційною системою управління базами даних - це означає, що дані зберігаються в окремих таблицях, пов'язаних між собою і визначених правилами зв'язків, які забезпечуються системою управління базами даних. Вона відома своєю швидкістю, надійністю та масштабованістю. ($MySQLDocumentationTeam$2016) Підтримується версія $MySQL$5.5 або новіша. Функція Django під назвою inspectdb використовує базу даних $\mathrm{information}_s chema$MySQL, $-, i, , iDjango, .iiiiDjangoi., i$MyISAM MySQL$iii, iiInnoDB, i.i, DjangoFramework-VimVirtualenv, iii\mathrm{Apache}\mathrm{mod}_w sgi$ та бекенду бази даних $MySQL$.

3.5 НАЛАШТУВАННЯ СЕРЕДОВИЩА РОЗРОБКИ

У цій главі обговорюється налаштування платформи для розробки безпосередньо перед початком кодування. Оскільки Django відома своєю нішею "для перфекціоністів з дедлайнами"в частині розробки, то відповідно легше налаштувати і середовище розробки. Віртуальне середовище було налаштоване для розробки в Ubuntu 16.04, відомому як Xenial Xerus, оскільки і розробка, і розгортання відбувалися на тій самій платформі. Крім того, проект Django також було створено перед початком написання коду в рамках налаштування платформи. Наступні розділи описують цей процес, щоб прояснити його більш

детально.

3.5.1 Налаштування віртуального середовища

Веб-платформа розроблена з використанням Python 3.7 та Django 1.9, отже, *pip* було встановлено для версії 3 Python. Після встановлення pip було встановлено *virtualenv* за допомогою командного інструменту *pip*. Після інсталяції інструменту *virtualenv* було створено нове віртуальне середовище під назвою $'env'$ у каталозі $'demo'$ за допомогою команди*virtualenv*. Як зазначалося раніше, *virtualenv* створює ізольоване і незалежне середовище для розробки на Python, яке слід активувати окремо - у випадку цього проекту, командою запуску скрипту активації *virtualenv* у терміналі Linux. Результатом активації є зміна запрошення командного рядка, що відображає активацію відповідного віртуального середовища. Віртуальне середовище можна будь-коли деактивувати за допомогою команди "$deactivate$", тобто за допомогою команди запуску скрипта деактивації *virtualenv*. Після активації віртуального середовища всі пакунки, необхідні для розробки, можна встановити за допомогою інструментарію pip, який за замовчуванням входить до складу *virtualenv*(якщо тільки *virtualenv* не встановлено з опцією $--no-pip$).

3.5.2 Початок проекту

Після активації віртуального середовища було встановлено $DjangoFramework$ за допомогою інструменту *pip*. Оскільки версія не була вказана, для розробки проекту було використано останню версію Django. Після встановлення Django у активованому віртуальному середовищі було створено проект Django під назвою $'demo'$ за допомогою команди $django-admin$, яка є утилітою для виконання адміністративних завдань. Цей процес автоматично створює дуже важливий утиліту $'manage.py'$ для виконання адміністративних завдань у каталозі проекту. Важливо зазначити, що під час цього процесу також створюється файл 'settings.py', який містить конфігурацію для проекту Django. На відміну від $django-admin$, "$manage.py$" зручніший у використанні, оскільки встановлює "$settings.py$" як вихідний файл конфігурації за замовчуванням для проекту, встановлюючи змінну оточення "$DJANGO_SETTINGS_MODULE$" у "$settings.py$" за допомогою команди $os.environ.setdefault($"$DJANGO_SETTINGS_$ "$thesis.settings$"). Крім того, вона встановлює пакунок, пов'язаний з проектом, у sys.path. Оскільки цей проект передбачає роботу лише з одним проектом Django і, відповідно, з одним файлом "$settings.py$", для адміністративних завдань краще використовувати "$manage.py$", ніж "$django-admin$". Наступне зображення демонструє можливості адміністрування скрипта $'manage.py'$ шляхом видачі команди на запуск сервера розробки:

Після створення проекту Django, в рамках проекту можна створювати різні додатки Django, які інтегруються для узгодженої роботи як компонен-

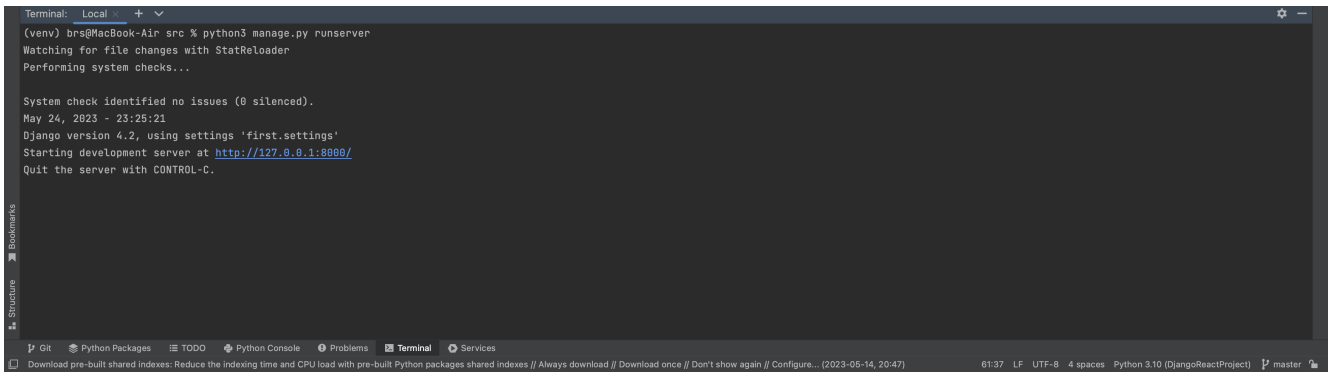


Рис. 6

ти проекту. Цілий сайт на Django може бути розроблений за допомогою одного додатку в рамках проекту, але розробка проекту з різними додатками, що представляють різні компоненти, є гарною практикою проектування. Більше того, як описано у Розділі 2, Django заохочує таку практику. На практиці для створення додатку в проекті використовується утиліта manage.py. На додаток до розробки додатку "posts"в рамках проекту Django, можна також використовувати сторонні додатки Django для досягнення подібних функціональних вимог, що пропонуються додатком "posts". Для цього дипломного проекту було створено та розроблено додатки спільноти, дописів, підтримки та людей. Детально процес розробки описано в наступному розділі.

3.6 ПРОЦЕС РОЗРОБКИ ДОДАТКІВ

У Django процес розробки додатків є надійним і простим, особливо коли використовується парадигма проектування MVC. Знайомство з цією парадигмою - наприклад, за допомогою Play Framework (який також є фреймворком MVC) - допомагає легко зрозуміти процес розробки додатків у Django. У цій главі обговорюється робочий процес Django в цілому, а також описується процес розробки додатків в цілому на прикладах з самого проекту.

3.6.1 Робочий процес Django

У додатку Django URL зіставляється з представленням за допомогою $URLconf$, який визначається у файлі $urls.py$. Під час створення проекту додається файл $urls.py$ (встановлений як $ROOT_URLCONF$ у $settings.py$), куди URLconf для додатків можна імпортувати за допомогою функції $include()$ з $django.conf.urls$. Коли Django отримує запити у вигляді URL, на основі URLconf викликається відповідне представлення, яке генерує відповідь для відображення у шаблоні. У найпростішій формі робочий процес Django складається з визначення моделі, написання представлення, оновлення URLconf для узгодження URL з представленням і написання шаблонів. Конфігурацію бази даних можна налаштувати вже або залишити конфігурацію за замовчуванням, яка використовується для $SQLite$, оскільки її легше розгортати на більш пізній стадії після

розробки. Для проекту файл *urls.py* вже створено під час створення проекту з власним $URL-$. Додатки можуть мати власний файл $URLconf$, який можна імпортувати до $ROOT_URLCONF$, тобто URLconf проекту, за допомогою функції *include*(). Оскільки додатки відіграють важливу роль у розробці Django, визначення додатків наведено у наступному розділі; а моделі є основним компонентом додатків, орієнтованих на роботу з даними, тому моделі у кожному додатку буде проілюстровано малюнками. Подання є важливими, але описувати їх усі незручно, тому в цій статті вони розглядаються лише коротко, тоді як $URLconf$ та urlpatterns будуть пояснені на прикладі з проекту.

3.6.2 Додатки та пов'язані з ними моделі в проекті

Дипломний проект містить чотири різні додатки, які називаються "*demo*", "*students*", "*staff*" та "*auth*". Додаток "*students*" містить користувацьку інформацію про користувачів, пов'язану з цим проектом, хоча користувачі реєструються за допомогою пакета python-social-auth, коли вони входять на сайт. Розширюючи клас *AbstractBaseUser* з Django, додаються кастомні поля для користувачів, які позначені як "Staff"у моделі Django у додатку "staff". Створені таким чином моделі також пов'язані з моделлю *UserSocialAuth* з пакету $python-social-auth$. Додаток posts призначений для створення дописів користувачем 'Staff' у додатку 'students' для 'community'. Наступна картинка демонструє графічну форму моделі в додатку posts. Django-розширення було використано разом з pygraphviz для створення всіх графіків моделі з чотирьох різних додатків..

3.6.3 Погляди в проекті

Подання - це як контролери, які визначають, що має бути передано шаблону для відображення в браузері. Крім того, він використовує форму, коли дані, введені користувачем, потрібно зберігати в базі даних. Коли в проекті створюється додаток, у папці з додатком створюється файл-скелет для представлення під назвою "*views.py*", який редагується відповідно до потреб додатка і того, що конкретний додаток повинен робити. У Django існує два типи представлень - представлення на основі функцій та представлення на основі класів. Для чистоти додатків використовуються подання на основі класів, тоді як подання на основі функцій використовуються у таких випадках, як сторінка контактів. Представлення на основі класів є гарним вибором, коли проект стає більшим, що вимагає успадкування. Більше того, Django постачається з деякими типовими представленнями на основі класів, які можуть бути успадковані у наших власних представленнях на основі класів. На наступному малюнку показано функціональне представлення, яке використовується для створення сторінки контактів. Це подання не використовує жодних моделей, хоча і записано у файлі подання, що належить додатку demo (див додаток Б).

Оскільки у цьому проекті обмежена кількість представлень, заснованих на функціях, у ньому використовується багато представлень, заснованих на класах. Приклад, показаний нижче на малюнку, демонструє $StudentCreateView$ - який успадковує запропоновані Django загальні $CreateView$ і $AdminRequiredMixin$ Модель визначено як Student (див. вище в описі моделі), а форму як $StudentForm$. Коли форма є дійсною під час відправлення форми, студент зберігається, але не фіксується, оскільки ще не визначено школу, до якої належить студент. Оскільки адміністратор може додати учня лише зі школи, до якої він належить, школа учня встановлюється такою ж, як і школа адміністратора, перед тим, як здійснити збереження.

Представлення рендериться з використанням шаблону за замовчуванням, розташованого за замовчуванням - в даному випадку він зберігається в папці додатку в $templates/community/$ як "$student_form.html$", де community - це назва додатку, student - це назва моделі, а $_form-i$, 2.AdminRequiredMixin$-ii, ii, ii$.AdminR$iiii$"$ii - ii/iiiii$Http404 $ii, , ii$.

Крім того, у проекті використовуються й інші міксини, оскільки вони можуть бути корисними для представлень на основі класів. Наприклад, LogingRequiredMixin широко використовується у цьому проекті для обмеження доступу до деяких сторінок, якщо користувач не увійшов до програми.

Як показано у файлі шаблону вище, тег $form$, який є назвою форми за замовчуванням у контексті, необхідний для відображення форми у відповіді, де form.as$_p i, i$(теги)."StudentCreateView rendered", ii.

```
{% extends "base.html" %}
{% block title %}Register{% endblock %}
{% load crispy_forms_tags %}
{% block content %}
<br>

<br>
<br>
<br>

<div class="col-md-5 mx-auto" style="background-color: #fff; border: 2px solid; border-radius: 5px; border-color: #fff">
  <br>
  <h4 class="my-2" style="text-align: center">Регистрация нового пользователя</h4>
  <br><br>
    <form method="POST" action="">{% csrf_token %}
    <div class="col">
      <div class="form-group col-md-12 mx-auto mb-0">
        {{ form.email|as_crispy_field }}
      </div>
      <div class="form-group col-md-12 mx-auto mb-0">
        {{ form.password|as_crispy_field }}
      </div>
      <div class="form-group col-md-12 mx-auto mb-0">
        {{ form.password2|as_crispy_field }}
      </div><br>
      <div class="d-grid gap-2 col-md-12 mx-auto mb-0">
        <button type="submit" class="btn btn-newb">Зарегистрироваться</button>
      </div>
    </div>

    </form>
  <br>
<br>
</div>



{% endblock %}
```

Рис. 7

На завершення, представлення на основі функцій є простими - вони вико-

ристовують декоратори методів для того, що роблять міксини у представленнях на основі класів, і завжди повертають об'єкт $HttpResponse$. Представлення на основі класів є зручним підходом до написання представлень в Django, проте потрібен час, щоб ознайомитися з базовими класами, що поставляються з Django, щоб почати працювати з кодуванням. Концепція контексту також важлива у випадку подання, незалежно від того, чи це подання на основі функцій, чи на основі класів. Django компілює шаблон один раз і містить імена змінних у подвійних фігурних дужках. У прикладі шаблону вище (Зображення 3), наприклад, $form$ є такою змінною, яка передається до шаблону як змінна-словник, що містить пару $'key' : ' '$. Представлення на основі класів - оскільки воно використовує базовий клас, що постачається з Django - не визначає власний контекст, а використовує успадкований (Рис. 9), тоді як представлення на основі функцій оголошує $'errors' : errors$ як контекст (Рис. 8). Нарешті, як описано у Розділі 3.5, ці подання викликаються Django, як тільки він знаходить відповідність URL-запиту, надісланого браузером, шаблонам $urlpatterns$ у $URLconf$.

3.6.4 Форми в проекті

Форми є важливим компонентом процесу веб-розробки, особливо якщо веб-додаток орієнтований на дані і приймає вхідні дані від користувача. Елементи форми можна написати у звичайному $HTML$, але Django полегшує роботу з формою багатьма способами. $DjangoForm$ - це клас, який генерує те, що має робити звичайна форма, тоді як $DjangoModelForm$ пов'язує поля в моделі з елементами форми. На рисунку 5 показано StudentForm, де поля у рядку 8 визначають, які поля з моделі Student (рисунок 4) мають бути відображені як елементи форми (яка показана на рисунку 3 вище). Крім того, клас Meta використовується для визначення моделі, пов'язаної з цією конкретною формою.

Порівнюючи поля форми, визначені в мета-класі StudentForm вище на рисунку 4, з полями моделі в Student нижче на рисунку 6, можна побачити, що не всі поля відображаються у формі при рендерингу в браузері. Деякі поля встановлюються під час обробки подання, тоді як деякі поля встановлюються автоматично, а деякі поля можуть бути необов'язковими.

На закінчення, в цьому проекті використовується ряд форм. Деякі типові форми генеруються "на льоту наприклад, при редагуванні або додаванні потреб учнів, адміністратор, який увійшов до системи, бачить лише учнів своєї школи, що можливо лише при створенні форми "на льоту". Додаткові складнощі в роботі з формою можуть з'явитися, коли є потреба в певній кастомізації - наприклад, коли поля форми залежать одне від одного (наприклад, користувач не може мати той самий пароль, що і його електронна пошта). Форма, описана в цьому розділі, здається простою, хоча вона може бути дещо складною для початківців, коли потрібно її кастомізувати. Наприклад, генерування моделі форми на льоту було новим для цього проекту і зайняло деякий час, щоб розібратися

```python
from django import forms
from crispy_forms.helper import FormHelper
from crispy_forms.layout import Layout, Submit, Row, Column, Field
from crispy_forms.layout import Field
from scraping.models import City, Language, Vacancy


class FindForm(forms.Form):
    city = forms.ModelChoiceField(
        queryset=City.objects.all(), to_field_name="slug", required=False,
        widget=forms.Select(attrs={'class': 'form-control'}),
        empty_label='Город',
        label=""
    )
    language = forms.ModelChoiceField(
        queryset=Language.objects.all(), to_field_name="slug", required=False,
        widget=forms.Select(attrs={'class': 'form-control'}),
        empty_label='Специальность',
        label=""
    )

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.layout = Layout(
            Row(
                Column('city', css_class='form-group col-md-4 mb-0'),
                Column('language', css_class='form-group col-md-4 mb-0'),
                # Submit('submit', 'Sign in', css_class='form-group col-md-1 mb-8'),
                css_class='form-row'
            ),
            Submit('submit', 'Sign in')
        )


class VForm(forms.ModelForm):
    city = forms.ModelChoiceField(
        queryset=City.objects.all(),
```

Рис. 8

```python

class UserRegistrationForm(forms.ModelForm):
    email = forms.EmailField(label='Введите имэил',
        widget=forms.EmailInput(attrs={'class': 'form-control'}))
    password = forms.CharField(label='Введите пароль',
        widget=forms.PasswordInput(attrs={'class': 'form-control'}))
    password2 = forms.CharField(label='Введите пароль ещё раз',
        widget=forms.PasswordInput(attrs={'class': 'form-control'}))

    class Meta:
        model = User
        fields = ('email',)

    def clean_password2(self):
        data = self.cleaned_data
        if data['password'] != data['password2']:
            raise forms.ValidationError('Пароли не совпадают')
        return data['password2']


class UserUpdateForm(forms.Form):
    city = forms.ModelChoiceField(
        queryset=City.objects.all(), to_field_name="slug", required=True,
        widget=forms.Select(attrs={'class': 'form-control'}),
        empty_label='Город',
        label=""
    )
    language = forms.ModelChoiceField(
        queryset=Language.objects.all(), to_field_name="slug", required=True,
        widget=forms.Select(attrs={'class': 'form-control'}),
        empty_label='Специальность',
        label=""
    )
    send_email = forms.BooleanField(required=False, widget=forms.CheckboxInput,
                                    label='Получать рассылку?')

    class Meta:
        model = User
        fields = ('city', 'language', 'send_email')
```

Рис. 9

з цим питанням.

3.6.5 URLConf у Django

Веб-сторінки пов'язані з URL-адресами. Коли користувач вводить URL-адресу в браузері, ця асоціація між сторінкою та URL-адресою є способом, за допомогою якого Django знає, яку сторінку обслуговувати для конкретного запиту. $URLconf$ у Django - це модуль Python, який Django використовує для зіставлення запитуваної URL-адреси з правильним поданням, яке буде обслуговуватися. Django використовує регулярні вирази для оголошення шаблонів URL-адрес і при надходженні запиту проходить по шаблону зверху вниз, поки не знайде перший збіг, після чого зупиняє подальшу обробку шаблону.

На наступному зображенні показано файл URLconf для програми "students".

```python
from django.urls import path
from accounts.views import login_view, logout_view, register_view, update_view, delete_view, contact

urlpatterns = [
    path('login/', login_view, name='login'),
    path('logout/', logout_view, name='logout'),
    path('register/', register_view, name='register'),
    path('update/', update_view, name='update'),
    path('delete/', delete_view, name='delete'),
    path('contact/', contact, name='contact'),

]
```

Рис. 10

Django uses the following algorithm to figure out what page to serve:

1) Кореневий модуль $URLconf$, який зазвичай встановлюється як $ROOT_URLCO$ у файлі $settings.py$, визначається для використання Django, якщо тільки проміжним програмним забезпеченням не буде замінено атрибут urlconf у налаштуваннях HttpRequest.

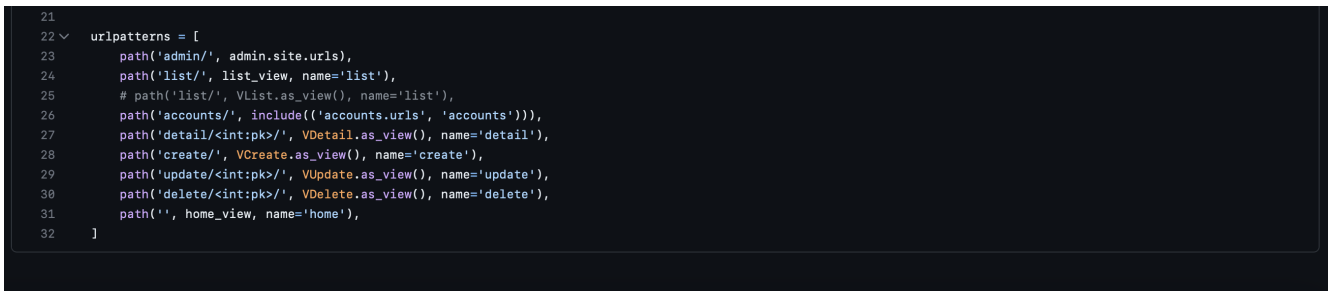
2) Django завантажує цей URLconf для пошуку шаблонів url, тобто екземплярів $django.conf.urls.url()$.

3) Список шаблонів URL обробляється по порядку, поки не буде знайдено перший збіг.

4) Коли URL збігається з рексом шаблону, імпортується і виконується подання, вказане в цьому шаблоні, з передачею HttpRequest, аргументів і ключових слів, якщо вони є.

5) Найкраще відповідне подання для обробки помилок викликається у випадку, якщо регекс не збігається, або якщо під час процесу запиту/відповіді виникає виняток - що описано в процесі запиту/відповіді раніше. (Django Software Foundation 2015e.)

Файл urls.py на зображенні вище з додатку people імпортується у файл $\mathrm{ROOT}_URLCONF(i).iii27, Djangoiiurl, .$



```
21
22  urlpatterns = [
23      path('admin/', admin.site.urls),
24      path('list/', list_view, name='list'),
25      # path('list/', VList.as_view(), name='list'),
26      path('accounts/', include(('accounts.urls', 'accounts'))),
27      path('detail/<int:pk>/', VDetail.as_view(), name='detail'),
28      path('create/', VCreate.as_view(), name='create'),
29      path('update/<int:pk>/', VUpdate.as_view(), name='update'),
30      path('delete/<int:pk>/', VDelete.as_view(), name='delete'),
31      path('', home_view, name='home'),
32  ]
```

Рис. 11

Таким чином, шаблони url, визначені в різних додатках, є видимими для Django - отже, шаблони url можуть вільно зв'язувати різні компоненти разом у цьому специфічному аспекті процесу розробки (див додаток C).

3.7 ПРОЦЕС РОЗГОРТАННЯ ПРОГРАМИ

Дипломний проект було розгорнуто на веб-сервері Apache з використанням mod_wsgi. Внутрішні дані зберігалися за допомогою системи управління базами даних MySQL. У наступних розділах детально описано налаштування та конфігурацію процесу розгортання.

3.7.1 mod_wsgi та розгортання Django на сервері Apache

mod_wsgi можна встановити також за допомогою pip - системи керування пакунками, що використовується для встановлення та керування пакунками python ($Pythonii2012$) - команди, яка генерує конфігурацію автоматично; проте для цього проекту обрано традиційний підхід встановлення, коли Apache налаштовано на завантаження mod_wsgi вручну. Вихідний код у вигляді tar-архіву було завантажено за допомогою команди wget та розпаковано як:

$brs@MacBook - Air$

Як зазначено на сторінці проекту модуля в Google, також було встановлено пакет $apache2 - dev$, оскільки в цьому випадку апач було встановлено з репозиторію пакетів за допомогою команди "$sudoapt - getinstallapache2$" на сервері $Ubuntu(Dumpleton2016b)$.

Розпакований вихідний код було налаштовано за допомогою команди $./configu$ пакунок було зібрано за допомогою команди make, а потім встановлено у стандартне місце за допомогою команди $makeinstall$ - місце, вказане $Apache$ для його модулів. Журнал помилок показує наявність mod_wsgi, як показано на малюнку вище. Після підтвердження наявності mod_wsgi сервер Apache було перезапущено командою $'sudo/etc/init.d/apache2restart'$ і виконано команду $'makeclean'$ для очищення після встановлення. Завантажений $tar - i$ і розпаковані вихідні файли було видалено.

Конфігурацію віртуального хоста було оновлено, додавши до неї рядок 1, як показано на рисунку нижче, для завантаження модуля mod_wsgi. Крім того,

на малюнку показано повну конфігурацію віртуального хоста для сайту, яка включає реалізацію протоколу $HTTPS$ для безпечного зв'язку між клієнтами і сервером. Зверніть увагу, що для $HTTP$ і $HTTPS$ визначено лише один процес $WSGIDaemonProcess$, оскільки вони виконуються в межах одного процесу демона.

Після завершення роботи з файлом $VirtualHost$, сайт було увімкнено, а сервер *Apache* було перезапущено, щоб зміни набули чинності.

3.7.2 Розгортання проекту Django за допомогою MySQL

Для розгортання $Django$ з $MySQL$ на комп'ютері було встановлено сервер $MySQL$ і створено базу даних під назвою "*demo*" для проекту. Крім того, були застосовані методи для посилення безпеки бази даних, як описано далі у цьому розділі. У файлі *settings.py* проекту $Django$ механізмом бази даних було встановлено $MySQL$, а конфігурацію доступу до бази даних було імпортовано з іншого файлу, як показано на малюнку нижче:

```
87  # Database
88  # https://docs.djangoproject.com/en/3.2/ref/settings/#databases
89
90  DATABASES = {
91      'default': {
92          'ENGINE': 'django.db.backends.postgresql',
93          'NAME': DB_NAME,
94          'USER': DB_USER,
95          'PASSWORD': DB_PASSWORD,
96          'HOST': DB_HOST,
97          'PORT': '5432',
98      }
99  }
100
101 db = dj_database_url.config()
102 DATABASES['default'].update(db)
103
104 # Password validation
105 # https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators
106
107 AUTH_PASSWORD_VALIDATORS = [
108     {
109         'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
110     },
111     {
112         'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
113     },
114     {
115         'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
116     },
117     {
118         'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
119     },
120 ]
121
```

Рис. 12

Наведена вище конфігурація доступу до бази даних має вигляд, як показано на малюнку нижче. Пароль замасковано, щоб приховати його від читачів. За замовчуванням встановлено кодування символів $utf8$, яку ми коротко обговорювали раніше.

Пакет $mysql - python$ було встановлено за допомогою інструменту *pip*, який є інтерфейсом $Python$ для бази даних $MySQL$, тобто бібліотекою підключення до бази даних. Для успішного встановлення пакета mysql-python у віртуальному середовищі необхідно було встановити в системі файли пакету

Рис. 13

розробки для бази даних $libmysqlclient - dev$. Після процесу розгортання були розглянуті наступні заходи безпеки.

3.7.3 Заходи безпеки у додатку Django

Практики безпеки були прийняті частково під час розробки і в основному під час розгортання платформи. Хоча проект знаходиться в стадії прототипу, за своєю природою він має бути захищеним - особливо, коли йдеться про інформацію про вразливих людей та фінансові операції. З цієї причини вимогам безпеки додатку було приділено особливу увагу, і було вжито наступних заходів: У файлі $settings.py$ проекту $DEBUG = True$ було змінено на $DEBUG = False$. Увімкнення налагодження з параметром $DEBUG = True$ рекомендується лише на етапі розробки. Крім того, параметр $DEBUG = True$ також є ресурсоємним. Наприклад, запити до бази даних будуть зберігатися в пам'яті. Перевірка заголовків хостів була встановлена в Django за допомогою параметра $ALLOWED_HOSTS$. Власне, коли $DEBUG$ встановлено у значення $False$, цю опцію необхідно встановити. У нашому випадку було встановлено значення meroserofero.org, оскільки цей домен обслуговується. Це є заходом безпеки проти атак по заголовку $HTTPHost$. ($DjangoSoftwareFoundation2015a$.)

$SECRET_KEY$ не зберігався у файлі $settings.py$, а зберігався у файлі і зчитувався з нього. $ADMIN$ було встановлено у файлі налаштувань - так, щоб про будь-яку помилку сайт $Django$ повідомляв адміністратору. Щодо питань безпеки в додатку, то для обмеження прав доступу для різних користувачів були використані привілеї користувачів. Для захисту від підробки міжсайтових запитів $Django$ надає тег $template$, який використовується у всіх формах, що надсилаються з браузера за допомогою методу $POST$. Тільки адміністратори можуть створювати профілі студентів. Крім того, $HTTPS$ реалізовано для захисту зв'язку між клієнтом і сервером - тому всі запити в $HTTP$ перенаправляються на $HTTPS$ сервером $Apache$ за замовчуванням. Для отримання додаткової інформації, питання безпеки, пов'язані з Django, можна переглянути за

цим посиланням ($https://docs.djangoproject.com/en/1.9/releases/security/$), яке є архівом питань безпеки.

3.7.4 Проблеми безпеки баз даних

Бази даних є невід'ємним компонентом веб-сайтів і надзвичайно важливі для сайтів, орієнтованих на роботу з даними. Через таку важливість, безпеці бази даних цього проекту було приділено особливу увагу під час розгортання проекту. Що стосується цього проекту, безпека $MySQL$ була посилена за допомогою наступних методів, які розглядалися під час розгортання:

1) Було встановлено простий брандмауер (ufw) і увімкнено заборону всього вхідного зв'язку, крім протоколів SSH, $HTTP$ і $HTTPS$, як показано на наступному рисунку.

2) Привілеї користувачів були встановлені на мінімальному рівні, щоб їх можна було поступово підвищувати відповідно до потреб, а для різних баз даних були створені окремі користувачі.

3) Утиліта $mysql_secure_installation$ була використана для подальшого посилення безпеки в системі $MySQL$.

4) Кореневий обліковий запис $MySQL$ був заплутаний через використання іншого імені. ($Davidson, 2010$)

5) Атака SQL ін'єкції захищена в Django, якщо тільки запити не виконуються за допомогою кастомного sql. У випадку цього проекту використовуються лише набори запитів Django, які обходять $sql-$ за допомогою базового драйвера бази даних. (2015.)

Після забезпечення безпеки розгорнутої системи були розглянуті питання масштабування, як показано в наступному розділі.

3.7.5 Проблеми масштабованості в застосуванні

Для великого проекту масштабованість так само важлива, як і безпека. Іноді накладні витрати на продуктивність виникають через затримку в спілкуванні з іншими сервісами. Якщо додаток масштабується до такого рівня, можна впровадити кеш. Це може бути реалізовано за допомогою $Varnish$ - програмного забезпечення з можливістю кешування всієї $HTTP-iii$, що дозволяє швидко та ефективно надавати відповідь на конкретний запит, навіть без перенаправлення запитів на сервер Django ($Robenolt2013$). Статичні файли обслуговуються за допомогою самого сервера Apache. mod_wsgi реалізовано в режимі демона, що робить передбачуваним споживання ресурсів через постійні потоки та процеси.

Коди можуть бути покращені - для цього можна використовувати панель налагодження $DjangoDebugToolbar$ для точного налаштування продуктивності. Це, безумовно, вимагає значної майстерності, яку автору ще належить розвинути. Шардінг бази даних може бути реалізований, якщо дані значно зростають. Наприклад, дані, пов'язані з групою (у нашому випадку - громадою або школою), можна зберігати в одній базі даних. Іншим варіантом масштабування бази даних може бути реплікація $MySQL$, де Django можна налаштувати так, щоб запитувати вставки та оновлення до головної бази даних, а вибірки - до підлеглих ($Shuping2014$). Конфігурацію краплі розгортання (яка є віртуальним приватним сервером) можна збільшити, коли трафік додатку зростає.

Як зазначалося вище, деякі з практик масштабування вже впроваджені, тоді як деякі з них рекомендовані для подальшого розвитку.

Висновки

Таким чином, розробка автоматизованої довідково-інформаційної системи для збору довідок була успішно завершена. В ході проекту були проаналізовані та вирішені різні аспекти, включаючи розробку моделі бази даних, вибір програмних засобів, дизайн інтерфейсу та процедури тестування.

Аналіз алгоритмів роботи з базою даних забезпечив ефективне та оптимізоване управління даними, що дозволило безперешкодно отримувати та маніпулювати інформацією. Вибір відповідних програмних інструментів, таких як фреймворк Django та мова програмування Python, забезпечив міцну основу для розробки надійної та зручної системи.

Дизайн інтерфейсу був зосереджений на зручності та доступності, включаючи інтуїтивно зрозумілу навігацію, чітке представлення інформації та інтерактивні функції. Це забезпечило позитивний користувацький досвід і сприяло ефективному збору та управлінню посиланнями.

Тестування відіграло вирішальну роль у процесі розробки: розробник проводив комплексне модульне тестування для перевірки функціональності та надійності системи. Завдяки ретельному плануванню, розробці, виконанню та налагодженню тестів були виявлені та усунені потенційні проблеми та дефекти, що забезпечило високу якість програмного продукту.

Результатом успішного завершення проекту стало створення надійної автоматизованої довідково-інформаційної системи, яка спрощує та впорядковує збір довідкової інформації. Вона забезпечує централізоване управління даними, надійну автентифікацію, ефективний пошук та зручний інтерфейс як для адміністраторів, так і для користувачів.

Розроблена система має важливе значення для навчальних закладів, компаній та організацій, які потребують організованого та автоматизованого підходу до збору довідкової інформації. Вона зменшує ручну працю, мінімізує помилки та підвищує продуктивність в управлінні даними про студентів та співробітників.

Таким чином, завершений проект досягнув поставлених цілей - створення функціональної та зручної для користувача автоматизованої довідково-інформаційної системи. Ретельний аналіз, ретельний підбір програмних засобів, увага до дизайну інтерфейсу та суворі процедури тестування призвели до створення надійного та ефективного рішення. Програмний продукт має значну цінність для його цільових користувачів і закладає основу для подальших удосконалень та покращень в управлінні довідково-інформаційними ресурсами.

Література

[1] D. Hillard: ”Practices of the Python Pro”.

[2] D. R. Greenfeld, A. R. Greenfeld: “Two Scoops of Django 1.11: Best Practices for the Django Web Framework”.

[3] A. Beaulieu: “Learning SQL”

[4] Donald E. Knuth: “The art of computer programming”

[5] Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы.

[6] Davidson M. Knuth: “MySQL Server Hardening”

[7] A. Holovaty, J. Kaplan-Moss: “The Definitive Guide to Django : Web Development done Right. Berkeley: Apress”

[8] E. Neuman: “Ten Reasons Django is Perfect for Startups”

[9] Smith, J. (2010): “The Art of Software Development. Publisher”

[10] Johnson, M. (2012): “Effective Testing Strategies for Software Products” Journal of Software Engineering, 25(3), 45-62.

[11] Brown, S. (2015). User Interface Design: “Principles and Best Practices. Publisher”

[12] Davis, R. (2018): “Database Management Systems: Concepts and Implementation”

[13] Robinson, C. (2011): “Introduction to Object-Oriented Programming with Python”

[14] Wilson, E. (2014): “Software Documentation: A Comprehensive Guide”

[15] Adams, R. (2019): “Data Modeling and Database Design”

[16] Nelson, D. (2016): “Introduction to Web Development with HTML and CSS”

[17] Turner, A. (2011): “Software Configuration Management: Best Practices and Tools”

[18] Davis, S. (2019): “Software Product Testing: Strategies and Techniques”

[19] Johnson, K. (2015): “User Experience Design: Creating Meaningful Interactions”

Додаток А

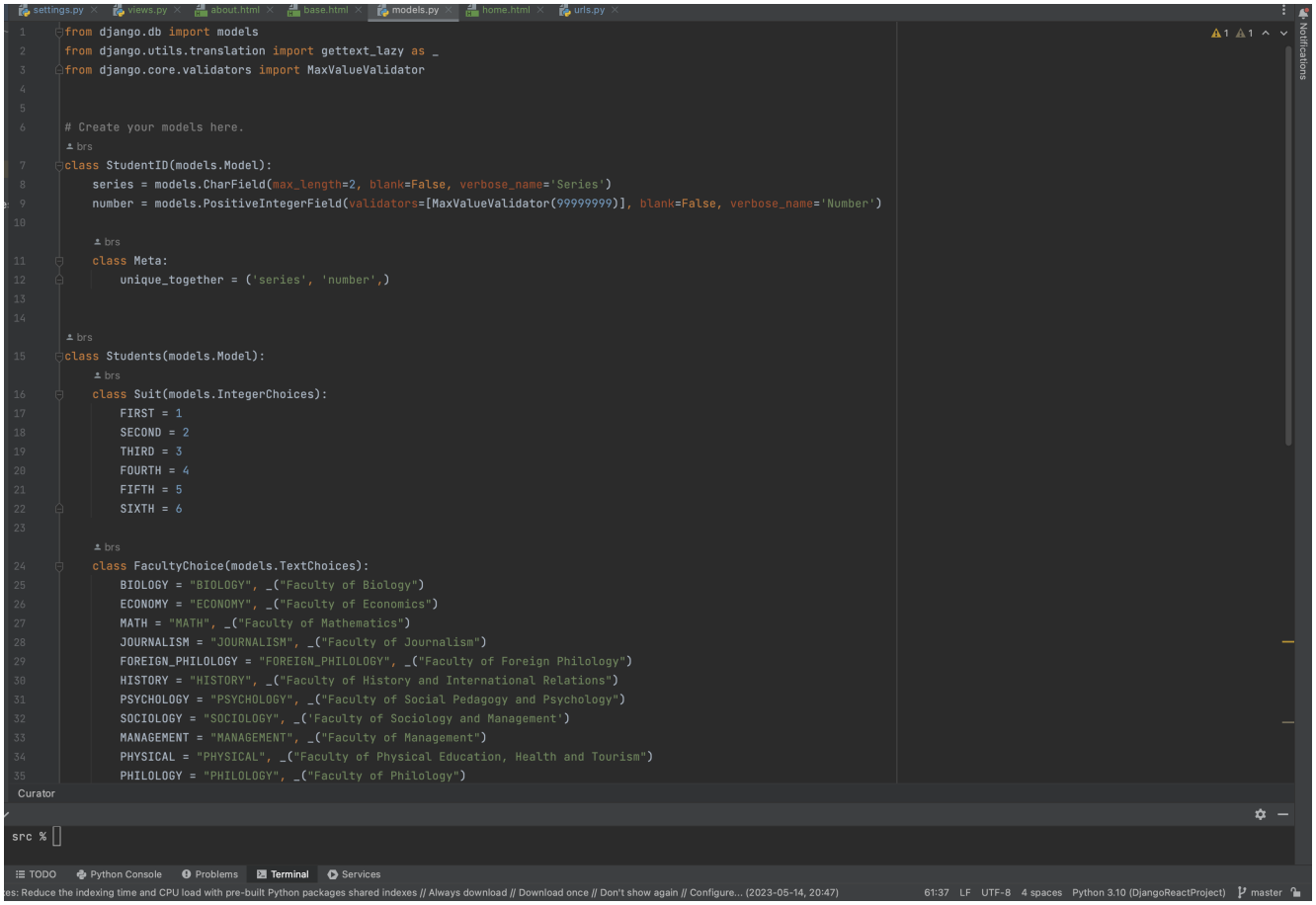


Рис. 14

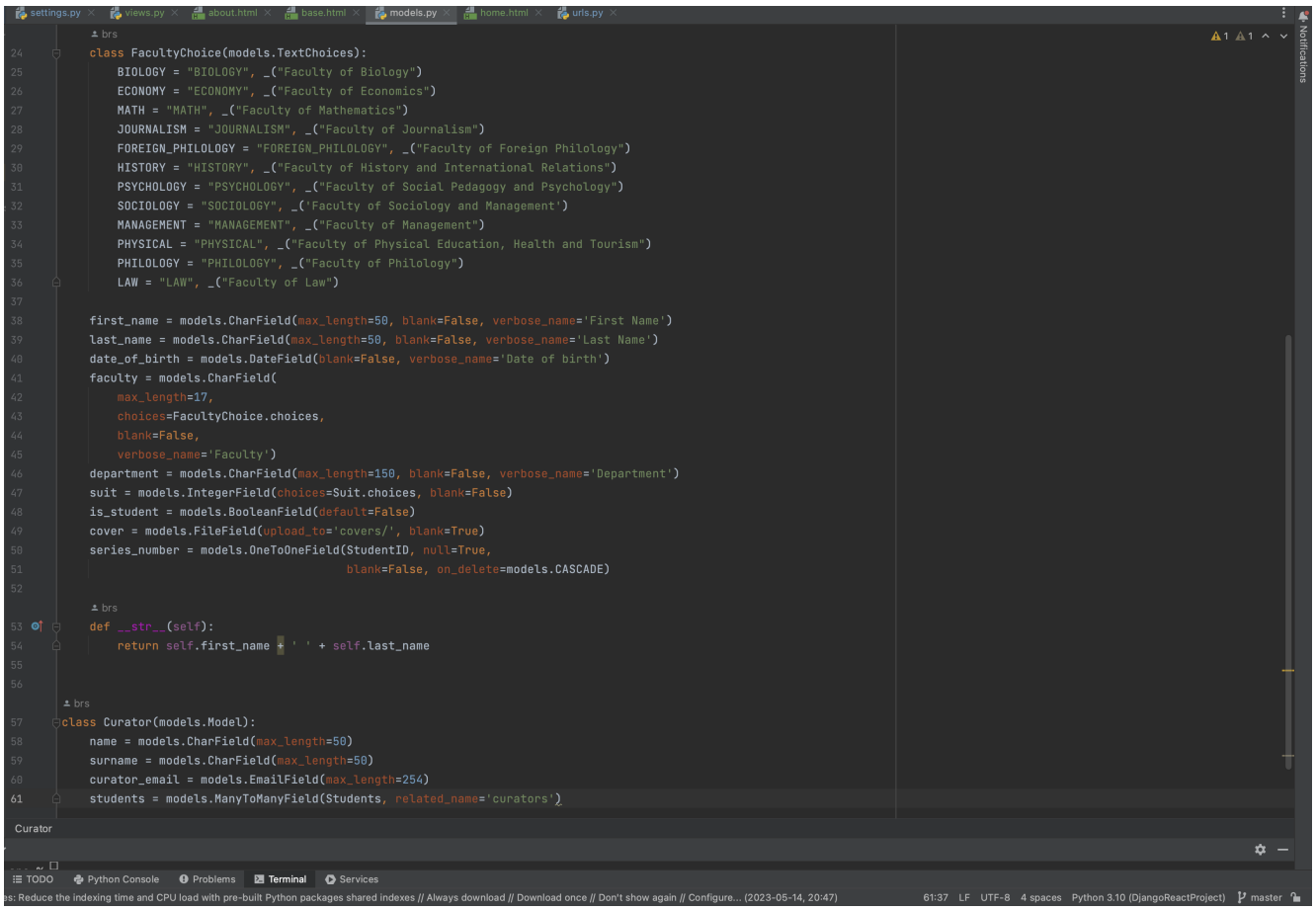


Рис. 15

Додаток Б

Додаток В