МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА
Відокремлений структурний підрозділ «Фаховий коледж Чернівецького національного університету імені Юрія Федьковича»

Природниче відділення
Циклова комісія комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

зі спеціальності *123 Комп'ютерна інженерія*

підготовки за освітньо-професійною програмою «*Комп'ютерна інженерія*»

на тему: «**Комп'ютерної системи забезпечення роботи куратора академічної групи»**

Виконав (ла):
студент (ка) 4-го курсу *Молдован Олександр Віталійович*
(прізвище, ім'я та по батькові) (підпис)

Керівник:
Плахов О.О. _____
(науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
Букурос О.В. _____
(науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

До захисту допущено:

Протокол засідання циклової комісії № _

від „___"_____ 2024 р.

Голова циклової комісії _____ Олександр ТАЩУК

Чернівці, 2024

Завдання та календарний план роботи

Міністерство освіти і науки України
Чернівецький національний університет імені Юрія Федьковича
Відокремлений структурний підрозділ «Фаховий коледж
Чернівецького національного університету імені Юрія Федьковича»

Затверджую
Голова циклової комісії,
Олександр ТАЩУК.

_____

«____» _____2023 р.

Завдання

На кваліфікаційну роботу	Молдован Олександр Віталійович
Тема кваліфікаційної роботи	Комп'ютерна система забезпечення роботи куратора академічної групи
Постановка завдання в короткій формі	Створення системи відеонагляду в локальній мережі за допомогою Cisco Packet Tracer
Вихідні дані (2-3 адреси сайтів, матеріали яких рекомендує керівник кваліфікаційної роботи)	

Календарний план роботи

Дата отримання версії звіту керівником	Етап виконання роботи	Виконання (зазначає керівник)
10.10.23	Отримання завдання до кваліфікаційної роботи.	
07.11.23	Огляд літератури, присвяченої створенню системи.	
18.12.23	Аналіз завдання. Оформлення розділу 1.	
15.01.24	Вибір технічної складової.	
22.02.24	Написання теоретичного матеріалу по використовуваних пристроях. Оформлення розділу 2.	
28.03.24	Опис передачі даних в мережі. Проектування та розробка програмної частини, підключення. Оформлення розділу 3.	
23.04.24	Моделювання системи. Оформлення розділу 4.	

15.05.24	Редагування та оформлення кваліфікаційної роботи.	
30.05.24	Подання роботи на перевірку Інтернет-сервісом Unicheck.	
02.06.24	Подання роботи на рецензію	
06.06.24	Представлення роботи на засіданні Циклової комісії	
за графіком	Захист кваліфікаційної роботи	

Дата видачі завдання _____

Студент Святослав БАБІЙЧУК

Керівник Олександр ПЛАХОВ

Анотація

Молдован Олександр Віталійович Кваліфікаційна робота присвячена розробці Комп’ютерної системи забезпечення роботи куратора академічної групи з використанням мови програмування Python і бібліотеки Tkinter для створення графічного інтерфейсу користувача.

Метою даної роботи є створення інструменту, який полегшить управління академічною діяльністю студентів, забезпечить централізоване зберігання даних та покращить комунікацію між куратором і студентами.

Система надає можливість зберігати інформацію про студентів, їхні оцінки, відвідуваність та зворотній зв'язок в одному місці. Вона дозволяє швидко отримувати доступ до необхідних даних, автоматизувати рутинні завдання та забезпечує гнучке управління розкладом занять. Крім того, система дозволяє студентам залишати зворотній зв'язок, що покращує комунікацію та сприяє більш ефективному вирішенню виникаючих проблем.

У роботі розглянуто основні функціональні можливості системи, такі як додавання і видалення студентів, запис і моніторинг оцінок та відвідуваності, створення і зміна розкладу занять. Також проведено порівняння з аналогічними рішеннями на ринку, такими як Blackboard, Moodle та Google Classroom, що дозволило визначити унікальні переваги і недоліки запропонованої системи.

Впровадження даної системи управління студентами може значно покращити організацію навчального процесу, підвищити ефективність роботи куратора і створити більш структуроване і сприятливе навчальне середовище для студентів. Висновки, зроблені на основі аналізу функціональних можливостей системи та її впливу на академічну діяльність, свідчать про доцільність використання такого інструменту в умовах сучасної освіти.

ЗМІСТ

5.3Внесення необхідних змін та вдосконалень на основі результатів тестування.

6. Оцінка результатів 31-37

6.1Аналіз функціональності та продуктивності розробленої системи.

6.2Порівняння її характеристик з аналогічними рішеннями на ринку.

6.3Оцінка впливу системи на роботу куратора та академічної групи в цілому.

7. Висновки 38-39

Узагальнення отриманих результатів та висунення висновків щодо досягнень.

Обговорення можливостей подальшого розвитку системи та напрямків подальших досліджень у даній області.

8. Список використаних джерел 40

Перелік літератури, статей, сайтів та інших джерел, які використовувалися при написанні роботи.

9. Додатки 40-64

Додаткові матеріали, які можуть бути корисними для розуміння роботи або реалізації системи: код програми, додаткові дані тощо.

Вступ

Сучасна освітня система постійно розвивається та змінюється під впливом технологій та суспільних потреб. Завдяки цьому виникає необхідність впровадження ефективних інструментів та технологій для полегшення процесів керування академічними групами. У цьому контексті виникає велика потреба у розробці комп'ютерної системи, яка забезпечить кураторів академічних груп набором інструментів для виконання їхніх повсякденних обов'язків та оптимізації процесів керування.

Куратор академічної групи відіграє ключову роль у формуванні та підтримці успішної освітньої траєкторії студентів. Він відповідає за організацію навчального процесу, взаємодію зі студентами та вирішення різних адміністративних питань. Однак, ці обов'язки часто вимагають значних зусиль та часу, і можуть бути ускладнені великою кількістю студентів у групі та потребою у систематизації та обробці великого обсягу інформації.

У зв'язку з цим, розробка комп'ютерної системи, спрямованої на автоматизацію та оптимізацію роботи куратора академічної групи, є актуальною та перспективною задачею. Така система має потенціал покращити ефективність роботи куратора, забезпечити збереження та доступ до важливої інформації, спростити процеси взаємодії зі студентами та підвищити загальний рівень організації навчального процесу.

Ця дипломна робота спрямована на розробку та аналіз комп'ютерної системи для куратора академічної групи з метою покращення ефективності його роботи та забезпечення оптимального управління навчальним процесом. У наступних розділах будуть розглянуті деталі проектування, реалізації та оцінки результатів цієї системи.

- **Визначення мети та завдань дослідження**

Мета дослідження полягає у розробці та апробації комп'ютерної системи, яка забезпечить кураторам академічних груп ефективні інструменти для виконання їхніх повсякденних обов'язків та оптимізації процесів керування навчальним процесом. Головною метою є підвищення продуктивності та ефективності роботи кураторів, спрощення їхніх завдань та покращення якості керування академічними групами.

Для досягнення поставленої мети перед дослідженням стоять наступні завдання:

1. **Аналіз потреб та вимог кураторів**: Провести детальне дослідження ролі та обов'язків кураторів академічних груп, ідентифікувати основні вимоги до комп'ютерної системи з їхнього боку.

2. **Проектування комп'ютерної системи**: Розробити архітектуру та функціональні вимоги до системи, враховуючи результати аналізу потреб кураторів.

3. **Реалізація системи**: Розробити та налагодити програмне забезпечення, яке відповідає вимогам проекту та забезпечує необхідний функціонал.

4.**Тестування та валідація**: Провести тестування системи з метою виявлення та виправлення можливих помилок, а також перевірки відповідності її функціоналу вимогам та очікуванням користувачів.

5. **Оцінка ефективност**: Здійснити оцінку продуктивності та ефективності роботи системи з використанням визначених метрик та порівняння її з аналогічними рішеннями на ринку.

6. **Апробація системи**: Провести експериментальне впровадження системи в реальне навчальне середовище з метою збору фідбеку та оцінки її придатності у реальних умовах.

Огляд літератури

Огляд літератури - це ключовий розділ дипломної роботи, де аналізується існуючі джерела, що стосуються теми дослідження.

Функціональність систем для керування навчальними групами може включати різноманітні можливості, спрямовані на полегшення та оптимізацію різних аспектів організації навчального процесу. Ось кілька ключових функцій, які можуть бути реалізовані в таких системах:

1. Облік відвідування студентів:

- Система може дозволяти кураторам вести облік відвідування студентів на заняттях. Це включає в себе ведення списків присутності, відмітки про відсутність та причини відсутності.

2. Планування занять:

- Система може надавати можливість планувати та розкладати заняття для кожної навчальної групи. Це може включати розподіл занять за днями тижня, визначення типу заняття (лекція, практика, семінар тощо) та призначення викладачів.

3. Обмін повідомленнями:

- Система може забезпечувати можливість обміну повідомленнями між кураторами та студентами. Це може бути використано для сповіщення про зміни в розкладі, нагадування про важливі події або просто для зв'язку між учасниками навчального процесу.

4. Ведення журналів та оцінювання:

- Система може дозволяти кураторам вести журнали успішності студентів та здійснювати оцінювання їхньої академічної діяльності. Це включає в себе записи про результати тестів, лабораторних робіт, контрольних та інших видів оцінювання.

5. Моніторинг успішності студентів:

- Система може надавати інструменти для моніторингу успішності студентів та відстеження їхнього прогресу в навчанні. Це може включати статистику по оцінкам, аналіз активності студентів та звіти про їхні досягнення.

6. Автоматизація адміністративних процесів:

- Система може допомагати кураторам в автоматизації адміністративних процесів, таких як формування звітів, відправлення листів, підготовка документації та інші рутинні завдання.

Ці функції допомагають кураторам та іншим учасникам навчального процесу ефективно організовувати та керувати навчальними групами, забезпечуючи оптимальні умови для навчання.

Інтерфейс користувача (UI) є ключовим аспектом будь-якої комп'ютерної системи для керування навчальними групами, оскільки від нього залежить зручність та ефективність використання системи кураторами, студентами та іншими користувачами. Ось деякі аспекти:

1. Зручність використання:

- Інтерфейс повинен бути інтуїтивно зрозумілим та легким у використанні для користувачів будь-якого рівня комп'ютерної грамотності. Елементи керування, такі як кнопки, меню та форми, повинні бути розміщені логічно та доступні згідно з контекстом використання.

2. Навігація:

- Система повинна мати чітку та зручну систему навігації, яка дозволяє користувачам швидко переходити між різними розділами та функціями. Меню, панелі інструментів та інші елементи навігації повинні бути легко доступні та логічно організовані.

3. Пристосованість до користувача:

- Інтерфейс повинен мати можливості для персоналізації та налаштування залежно від потреб конкретних користувачів. Наприклад, кожен користувач

може мати можливість налаштувати свій профіль, вибрати переваги щодо відображення інформації та інші параметри.

4. Візуалізація даних:

- Система повинна ефективно візуалізувати навчальну інформацію та дані, щоб користувачі могли швидко зрозуміти ситуацію та приймати інформовані рішення. Графіки, діаграми, таблиці та інші візуальні елементи можуть бути використані для цієї цілі.

5. Мобільність:

- Урахуйте можливість використання системи на мобільних пристроях, таких як смартфони та планшети. Інтерфейс повинен бути адаптивним та оптимізованим для різних розмірів екрану та специфіки використання на дотикових пристроях.

6. Доступність:

- Впевніться, що інтерфейс відповідає стандартам доступності, щоб забезпечити можливість використання системи для користувачів з обмеженими можливостями.

Аналіз інтерфейсу користувача допоможе забезпечити, комп'ютерна система для керування буде зручною, ефективною та легко засвоюваною користувачами.

Звичайно, ось таблиця, яка відображає можливості інтеграції для комп'ютерних систем управління навчальними групами:

Можливість інтеграції	Опис
LMS (Learning Management Systems)	Інтеграція з платформами для управління навчанням, такими як Moodle, Blackboard, Canvas тощо.

	Дозволяє обмінюватися даними про курси, користувачів та оцінки
SIS (Student Information Systems)	| Інтеграція з системами інформації про студентів, які містять дані про реєстрацію, графік навчання, академічну історію тощо. Дозволяє автоматизувати процеси обліку студентів та їхніх даних.
Електронні журнали та оцінки	Інтеграція з електронними системами для ведення журналів та оцінювання студентів. Дозволяє автоматизувати передачу даних про успішність студентів та оцінки.
Платформи для відео-конференцій	Інтеграція з платформами для відео-конференцій, такими як Zoom, Microsoft Teams, Google Meet тощо. Дозволяє організувати

<table>
<tr><td colspan="2"></td><td>віртуальні зустрічі та онлайн-уроки.</td></tr>
<tr><td colspan="2">Електронний документообіг</td><td rowspan="2">Інтеграція з системами електронного документообігу для обміну документами та інформацією між користувачами системи. Дозволяє спростити процеси обміну документами та координації робіт.</td></tr>
<tr><td>Системи для тестування та оцінювання</td><td>Інтеграція з системами для проведення тестів та оцінювання знань, такими як Respondus, ExamSoft, Quizzes у Google Classroom тощо. Дозволяє автоматизувати процеси тестування та оцінювання студентів.</td></tr>
</table>

Схема безпеки та конфіденційності даних може включати такі елементи:

1. Автентифікація та авторизація:

- Використання механізмів аутентифікації, таких як паролі, біометричні дані або двофакторна аутентифікація, для перевірки ідентичності користувачів. Авторизація визначає права доступу до конкретних ресурсів чи функцій системи на основі ролей користувачів.

2. Шифрування даних:

- Використання методів шифрування для захисту конфіденційної інформації під час передачі через мережу та зберігання в базах даних. Це може включати шифрування транспортного каналу (SSL/TLS) та шифрування даних у спокійному стані (AES, RSA тощо).

3. Моніторинг та аудит:

- Встановлення систем моніторингу та аудиту, які відстежують доступ до даних та дії користувачів у системі. Це дозволяє виявляти неправомірні дії та вчасно реагувати на можливі загрози безпеці даних.

4. Фізична безпека інфраструктури:

- Забезпечення фізичної безпеки серверних приміщень, дата-центрів та інших компонентів інфраструктури, що забезпечують роботу системи. Це може включати контроль доступу, відеоспостереження, пожежну сигналізацію тощо.

5. Заходи захисту від вторгнень (IDS/IPS):

- Використання систем виявлення та запобігання вторгненням для виявлення та блокування спроб несанкціонованого доступу до системи або спроб злому.

6. Політики безпеки та регулярне оновлення:

- Розроблення та виконання строгих політик безпеки, які включають у себе встановлення паролів, регулярну зміну ключів шифрування, обмеження доступу до даних лише необхідним користувачам, а також регулярні оновлення систем та програмного забезпечення для виправлення виявлених уразливостей.

Ці елементи допомагають створити комплексну систему захисту даних в комп'ютерних системах управління навчальними групами, яка гарантує їхню безпеку та конфіденційність.

Припустимо, що розробка системи буде виконуватися з використанням сучасних технологій програмування та баз даних, а також враховуватиме потреби у безпеці та масштабованості. Орієнтовні витрати на розробку такої системи можуть становити від $50,000 до $200,000 і більше, залежно від рівня складності та функціональності.

Тут також важливо врахувати витрати на обслуговування та підтримку системи, які можуть становити приблизно 20-30% від вартості розробки щорічно.

В доларах:

- Розробка: $50,000 - $200,000+
- Обслуговування та підтримка: $10,000 - $60,000+ на рік

В гривнях (з урахуванням середнього курсу обміну):

- Розробка: приблизно 1,250,000 - 5,000,000+ гривень
- Обслуговування та підтримка: приблизно 250,000 - 1,500,000+ гривень на рік

Звичайно, ці цифри можуть варіюватися в залежності від конкретних умов та вимог проекту тому це чисто приблизна грошова еквівалентність розрахована по розрахунках.

Зазвичай доступність системи вимірюється у відсотках і розраховується як час, протягом якого система була доступна для користувачів, відносно загального часу. Наприклад, 99,9% доступності означає, що система може бути недоступною протягом 8,76 годин на рік.

Щоб забезпечити високу доступність системи, можна використовувати такі практики як:

1. Дублювання серверів та мережевих компонентів: Використання дублювання для забезпечення резервних екземплярів системи, які автоматично активуються у разі відмови основної системи.

2. Балансування навантаження: Розподіл навантаження між кількома серверами для забезпечення оптимального використання ресурсів та запобігання перевантаженням.

3. Регулярне планування та технічне обслуговування: Проведення регулярного технічного обслуговування та планових перерв у роботі системи для запобігання виникненню непередбачених проблем

4. Резервне копіювання даних: Забезпечення резервного копіювання даних для відновлення системи у випадку її відмови або втрати даних.

5. Моніторинг та автоматизація: Використання систем моніторингу для виявлення проблем та автоматичного відновлення системи в разі відмови.

Аналіз потреб куратора

Як я бачу роль куратора якщо б мені довелося ним бути з досліджень та спроб в навчальному закладі:

Як куратор академічної групи, моєю основною метою було забезпечення успіху та добробуту студентів під час їхнього навчання. Це вимагало багато відповідальності та різноманітних дій, включаючи:

1. Підтримка студентів: Я проводив індивідуальні консультації зі студентами, допомагаючи їм у вирішенні академічних та особистих питань. Це включало як розмови про навчальні труднощі, так і підтримку у вирішенні особистих проблем.

2. Організація подій та заходів: Я був відповідальний за організацію та проведення різноманітних заходів для студентів, включаючи тематичні лекції, культурні заходи та спортивні події. Ці заходи створювали сприятливу атмосферу спільноти та сприяли підтримці студентського духу.

3. Моніторинг успішності студентів: Я відстежував успішність студентів та реагував на їхні академічні труднощі, надаючи додаткову підтримку та рекомендації для поліпшення їхньої навчальної діяльності.

4. Співпраця з батьками: Я забезпечував взаємодію з батьками студентів, повідомляючи їх про академічну успішність та поведінку своїх дітей, відповідаючи на їхні запитання та розглядаючи можливі питання.

5. Підтримка психологічного клімату: Я надавав психологічну підтримку студентам у складних життєвих ситуаціях, допомагаючи їм розуміти та подолати стресові ситуації та виклики.

6. Взаємодія з адміністрацією та викладачами: Я співпрацював з адміністрацією та викладачами для вирішення академічних та організаційних питань, забезпечуючи ефективну роботу групи.

Повсякденні обов'язки:

Мої повсякденні обов'язки включали підготовку до зустрічей та заходів, проведення консультацій зі студентами, ведення документації та звітів про мою роботу, а також постійну взаємодію зі студентами та колегами. Це вимагало хорошої організації часу та здатності працювати в різних ситуаціях,

Але Звичайно, розглянемо роль куратора академічної групи та його повсякденні обов'язки у комп'ютерному плані:

1. Система керування студентською інформацією: Куратор використовує спеціалізовані програмні засоби для зберігання та оновлення інформації про студентів, таку як особисті дані, академічний прогрес, контактні дані тощо.

2. Електронна пошта та комунікація: Він використовує електронну пошту та спеціалізовані платформи для комунікації зі студентами, надсилає оголошення, інформує про події та розповсюджує корисну інформацію.

3. Відстеження академічного прогресу: Куратор використовує електронні засоби для відстеження успішності студентів, оцінює їхній академічний прогрес, вносить дані про оцінки та вирішує питання, пов'язані з успішністю.

4. Планування та організація заходів: Він використовує спеціалізовані програми для планування та організації заходів для студентів, створення розкладів, сповіщення про події та управління ресурсами.

5. Документування та звітність: Куратор веде електронну документацію про свою роботу, включаючи записи про консультації, заходи та взаємодію зі студентами, а також генерує звіти для адміністрації.

При розробці комп'ютерної системи для куратора академічної групи необхідно врахувати різноманітні завдання та функції, щоб забезпечити ефективну роботу та підтримку різних аспектів діяльності куратора та студентів. Ось деякі з них які вдалось знайти :

1. Керування інформацією про студентів: Збереження та оновлення персональних даних студентів, включаючи контактну інформацію, академічний прогрес, спеціальні потреби тощо.

2. Розклад та планування подій: Створення розкладу зустрічей, консультацій, тематичних заходів, важливих дат тощо, інтеграція з можливістю резервування приміщень та нагадувань.

3. Електронна пошта та комунікація: Платформа для ведення електронної переписки зі студентами, викладачами та батьками, а також сповіщень про важливі події та оголошення.

4. Моніторинг академічного прогресу: Відстеження успішності студентів, введення та оновлення оцінок, створення звітів та аналіз академічного прогресу.

5. Управління заходами та заходи підтримки: Реєстрація на заходи, організація та планування тематичних заходів, вебінарів, консультацій та інших активностей.

6. Система звітності: Ведення документації та статистичних даних щодо роботи куратора, генерація звітів для адміністрації та інших зацікавлених сторін.

7. Безпека та конфіденційність даних: Захист персональних даних студентів та інших конфіденційних інформаційних ресурсів, використання механізмів автентифікації та авторизації.

8. Моніторинг та аналіз даних: Забезпечення можливості аналізу та відстеження різних параметрів роботи системи для подальшого вдосконалення та оптимізації.

9. Інтеграція з іншими системами: Забезпечення можливості обміну даними з іншими системами університету, такими як система електронного навчання, истема обліку фінансів тощо.

Рисунок 3.1. Куратор функції

Проектування комп'ютерної системи

Розробка комп'ютерної системи для нашої академічної групи включає кілька важливих кроків. По-перше, необхідно визначити, що повинна робити система, наприклад, які функції вона повинна мати, які вимоги до продуктивності та безпеки тощо. Далі створюється архітектура системи, обираються відповідні технології та інструменти і починається розробка та тестування коду.

Потім система впроваджується, будується необхідна інфраструктура і (за необхідності) здійснюється міграція даних зі старої системи. Важливо здійснювати постійний моніторинг системи, надавати технічну підтримку користувачам та оновлювати систему на основі відгуків користувачів. На кожному етапі співпраця та комунікація між залученими сторонами має важливе значення для забезпечення найкращого можливого результату.

Перш за все, ми визначаємо, що нам потрібно. Наприклад, ми хочемо створити систему управління студентськими даними, яка дозволить кураторам:

- Зберігати інформацію про студентів.
- Відслідковувати успішність.
- Надавати студентам доступ до їхніх оцінок.

Архітектурне та детальне проектування

Вибираємо архітектуру системи: наприклад, веб-додаток з базою даних. Ми вирішуємо використовувати Django (Python) для бекенду та SQLite для бази даних, оскільки це простий та ефективний варіант для нашої мети.

Розробка коду починається з створення базових моделей даних. Ось приклад коду на Python з використанням Django для створення моделі студента:

```python
from django.db import models

class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    date_of_birth = models.DateField()
    email = models.EmailField(unique=True)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

Рисунок 4.1. Імя фамілія

Тестування

Після створення моделей та інших компонентів ми проводимо тестування. Наприклад, пишемо тести для перевірки створення студента:

```python
from django.test import TestCase
from .models import Student

class StudentModelTest(TestCase):
    def test_student_creation(self):
        student = Student.objects.create(
            first_name="Іван",
            last_name="Іванов",
            date_of_birth="2000-01-01",
            email="ivan@example.com"
        )
        self.assertEqual(student.first_name, "Іван")
        self.assertEqual(student.last_name, "Іванов")
```

Рисунок 4.2 Студентна модель

Впровадження

Після успішного тестування ми впроваджуємо систему, налаштовуючи сервер та базу даних. Наприклад, розгортаємо наш Django-додаток на Heroku або іншу платформу для розгортання.

Підтримка та обслуговування

Після запуску системи ми забезпечуємо її безперебійну роботу, моніторимо продуктивність та надаємо підтримку користувачам. Наприклад, регулярно перевіряємо логи та відгуки користувачів, виправляємо помилки та додаємо нові функції на основі зворотного зв'язку.

Документація

Ми створюємо документацію для користувачів і розробників. Наприклад, інструкції для користувачів можуть містити кроки для перегляду оцінок, а технічна документація описує архітектуру системи та інтерфейси API.

Приклад для користувачів

Користувачі (студенти) можуть увійти в систему та переглянути свої оцінки:

```python
from django.shortcuts import render
from .models import Student

def student_grades(request, student_id):
    student = Student.objects.get(id=student_id)
    grades = student.grade_set.all()  # Припустимо, що є модель Grade
    return render(request, 'student_grades.html', {'student': student, 'grades': grades})
```

Рисунок 4.3 Модель

Цей простий код демонструє, як студент може переглянути свої оцінки на веб-сторінці.

Як Куратор академічної групи, я обрав технологію та інструменти для створення системи управління даними про студентів на основі простоти використання, ефективності та надійності.

Для програмування я обрала мову Python, тому що вона проста у вивченні, має багато бібліотек і фреймворків, активну спільноту і хорошу документацію.

Веб-фреймворк, який ми використовуємо, - це Django, який дозволяє швидко розробляти веб-додатки, має вбудовані інструменти адміністрування, підтримує інтеграцію з базами даних за допомогою ORM і включає механізми автентифікації та авторизації.

В якості бази даних ми обрали SQLite. Вона проста у налаштуванні та використанні, підходить для невеликих проектів і не вимагає окремого серверного програмного забезпечення.

Інтерфейс використовує HTML, CSS і JavaScript для створення структурованих і стилізованих веб-сторінок, додаючи інтерактивність і динамічні елементи.

Для контролю версій ми використовуємо Git, оскільки він дозволяє відстежувати зміни коду, працювати над проектами разом, а також зберігати та ділитися кодом за допомогою таких сервісів, як GitHub.

Середовищем розробки ми обрали Visual Studio Code. Це середовище легко налаштовується, підтримує розширення для різних мов програмування та має вбудовані інструменти для налагодження.

```python
from django.db import models

class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    date_of_birth = models.DateField()
    email = models.EmailField(unique=True)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

Рисунок 4.4 Опис моделі

Реалізація системи

1. **Визначення вимог**

Перш ніж ми почали розробку системи, ми ретельно подумали про те, що саме нам потрібно. Ми хотіли мати зручний спосіб зберігати інформацію про наших студентів, їх оцінки і контактні дані. Також нам потрібно було зручно відслідковувати розклад занять і відсутності студентів.

2. **Вибір технологій**

Для створення цієї системи ми обрали Python, бо це мова програмування, яка дозволяє нам легко створювати програми і працювати з базами даних. Наше середовище розробки - Visual Studio Code - допомагає нам писати і перевіряти наш код. Ми також використовуємо Django, що є фреймворком для веб-розробки, який дозволяє нам швидко створювати веб-сайти і зручно працювати з базами даних.

3**. Розробка системи**

Після вибору технологій ми почали створювати нашу систему. Ми написали код, який дозволяє додавати нових студентів, зберігати їхні дані та переглядати розклад занять. Тепер ми можемо легко вносити зміни в інформацію про студентів і перевіряти, які заняття вони пропускають.

4. **Тестування**

Перед тим, як наша система стала доступною для використання, ми провели тестування, щоб переконатися, що все працює як потрібно. Ми перевірили кожну частину програми, щоб переконатися, що вона виконує свої функції правильно. Наприклад, ми переконалися, що можемо додавати нових студентів і зберігати їхні дані без помилок.

5. **Впровадження**

Коли ми були впевнені, що наша програма працює як потрібно, ми її запустили. Тепер кожен куратор може використовувати цю систему для ведення обліку своєї групи студентів. Вони можуть легко додавати нових студентів і переглядати їхні дані.

6. **Підтримка і покращення**

Після запуску ми продовжуємо підтримувати нашу систему. Якщо щось не працює, ми виправляємо це якомога швидше. Також ми постійно шукаємо способи, як можна покращити нашу систему, щоб вона стала ще зручнішою для кураторів і студентів.

Проведення фази розробки, включаючи програмування основних функцій системи.

Як куратор академічної групи, я хотів би розповісти про процес розробки основних функцій системи управління даними студентів.

Ми почали з визначення основних вимог до системи, включаючи можливість додавання, перегляду та редагування даних студентів. Для створення зручного та ефективного інтерфейсу ми обрали мову програмування Python та веб-фреймворк Django; завдяки Django ми змогли швидко реалізувати такі моделі даних, як студенти з полями для особистих даних та контактної інформації з полями для особистих даних та контактної інформації.

Після створення базової моделі ми реалізували можливість додавання нових студентів до бази даних та зберігання їхніх даних. Наша мета - зробити процес додавання та редагування даних студентів максимально зручним та безпечним.

Інтегрувавши ці функції в систему та протестувавши їх, ми змогли виявити та усунути потенційні проблеми ще до того, як система була запущена в експлуатацію. Ми прийняли методологію проактивного тестування, щоб переконатися, що система працює правильно, перш ніж вона буде запущена.

Цей процес розробки гарантує, що наша система управління даними студентів є надійною та відповідає всім потребам нашої академічної команди.

```
from app.models import Student

def add_student(first_name, last_name, date_of_birth, email):
    student = Student.objects.create(first_name=first_name, last_name=last_name, date_of_birth=date_of_birth, email=email)
    return student
```

Рисунок 5.1 Процес розробки

Тестування розробленої системи на відповідність вимогам та виявлення помилок:

Як куратор нашої академічної групи, я хотів би розповісти вам про процес тестування нашої новоствореної системи управління студентськими даними.

Я ретельно протестували кожну частину системи, щоб переконатися, що вона відповідає нашим вимогам. По-перше, ми протестували основні функції, які ми визначили на етапі планування: додавання, редагування та видалення даних про студентів та управління розкладом занять.

Під час тестування ми активно співпрацювали з розробниками, щоб виявити потенційні проблеми. Дуже важливо було переконатися, що дані зберігаються та обробляються в системі правильно. Виявивши помилки, ми змогли швидко їх виправити та покращити загальну якість продукту.

Тестування також включало перевірку безпеки системи: ми ретельно відстежували дані учнів, щоб переконатися, що вони захищені від потенційних загроз, таких як SQL-ін'єкції та інші атаки. Для перевірки системи на вразливості були використані спеціальні інструменти, а виявлені проблеми були своєчасно виправлені.

Все це важливо для забезпечення безпеки, надійності та ефективності системи управління даними про учнів. Наша мета - забезпечити, щоб усі користувачі могли використовувати систему з упевненістю, що їхні дані захищені та доступні, коли їм це потрібно.

```python
from django.test import TestCase
from app.models import Student
from datetime import date

class StudentModelTestCase(TestCase):
    def test_add_student(self):
        Student.objects.create(first_name="John", last_name="Doe", date_of_birth=date(2000, 1, 1), email="john.doe@example.c
        student = Student.objects.get(first_name="John")
        self.assertEqual(student.last_name, "Doe")
```

Рисунок 5.2 Дата народження

Після розробки основних функцій ми перейшли до етапу тестування системи. Ми створили тести, щоб переконатися, що кожна частина програми працює як потрібно. Наприклад, ми перевірили, чи правильно додаються нові студенти до бази даних і чи зберігаються їхні дані без помилок.

Під час тестування нашої системи управління студентськими даними ми зосередилися на декількох основних аспектах для переконання у відповідності системи вимогам і виявлення потенційних помилок.

Функціональне тестування

1. Додавання нового студента:

- Тестовий сценарій:Введення нових особистих даних студента через інтерфейс користувача.

- Очікуваний результат: Студент має бути успішно доданий до бази даних і збережений з правильними даними.

- Приклад коду:

```
def test_add_student():
    student = add_student("John", "Doe", "2000-01-01", "john.doe@example.com")
    assert student is not None
```

Рисунок 5.3 Додавання студента

2. Редагування студента:

- Тестовий сценарій:Зміна контактної інформації і збереження змін.

- Очікуваний результат: Зміни повинні бути успішно внесені до запису студента в базі даних.

- Приклад коду:

```
def test_edit_student():
    student = Student.objects.get(email="john.doe@example.com")
    student.email = "john.smith@example.com"
    student.save()
    updated_student = Student.objects.get(id=student.id)
    assert updated_student.email == "john.smith@example.com"
```

Рисунок 5.4 Редагування студента

Ітеграційне тестування

3. Тестування API для додавання студента:

- Тестовий сценарій: Відправлення POST-запиту на API для створення нового студента.

- Очікуваний результат: Отримання статусу 201 (створено) і наявність нового запису у базі даних.

Прикладкоду:

```
def test_api_student_creation():
    response = client.post('/api/students/',
    {'first_name': 'Jane', 'last_name': 'Smith', 'date_of_birth': '1998-05-15', 'email': 'jane.smith@example.com'})
    assert response.status_code == 201
```

Рисунок 5.5 Тест джимейла

Тестування безпеки

4. Перевірка захисту від SQL-ін'єкцій:

- Тестовий сценарій: Введення SQL-запиту в поле вводу для додавання студента.

- Очікуваний результат:Система має коректно обробити введені дані і уникнути виконання небажаних SQL-операцій.

- Приклад коду:

```
def test_sql_injection_protection():
    malicious_input = "'; DROP TABLE students; --"
    response = client.post('/add_student/', {'name': malicious_input})
    assert "Error" not in response.content
```

Рисунок 5.6 Безпека

Ці тести допомогли нам переконатися в правильності роботи системи перед її впровадженням. Кожен з них був ретельно спроектований для перевірки конкретної функціональності або захисту системи, щоб забезпечити високу якість нашого програмного забезпечення.

Тепер прейдемо до основни коду який в нас є:

```
def create_widgets(self):
    self.load_button = tk.Button(self, text="Load Students from CSV", command=self.load_students)
    self.load_button.pack(pady=10)

    self.create_csv_button = tk.Button(self, text="Create and Save CSV", command=self.create_and_save_csv)
    self.create_csv_button.pack(pady=10)

    self.search_button = tk.Button(self, text="Search Student by ID", command=self.search_student_by_id)
    self.search_button.pack(pady=10)

    self.delete_button = tk.Button(self, text="Delete Student by ID", command=self.delete_student_by_id)
    self.delete_button.pack(pady=10)

    self.schedule_button = tk.Button(self, text="Show Schedule", command=self.show_schedule_window)
    self.schedule_button.pack(pady=10)

    self.output_text = tk.Text(self, wrap=tk.WORD)
    self.output_text.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)
```

Рисунок 5.7 кнопки

На цьому скріншоті ми бачимо як створювались та показані кнопки нашого програмного коду.Кожна з них відповідає за конкретну дію

Load Students from CSV відповідає за список наших студетів їх айді та день народження.

```
def load_students(self):
    csv_file = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
    if csv_file:
        group_id = 'A1'
        if not self.curator_system.get_group(group_id):
            group = Group(group_id)
            self.curator_system.add_group(group)

        self.curator_system.load_students_from_csv(group_id, csv_file)
        self.show_students()
```

Рисунок 5.8 додавання студента

Create and Save CSV-відповідає за додавання студента до нашої групи в цій строкі ми надаємо студенту айді,вписуємо його ім'я та фамілію,та дату його народження.

```python
def create_and_save_csv(self):
    self.create_csv_window = tk.Toplevel(self)
    self.create_csv_window.title("Create CSV")
    self.create_csv_window.geometry("400x300")

    tk.Label(self.create_csv_window, text="ID:").pack(pady=5)
    self.id_entry = tk.Entry(self.create_csv_window)
    self.id_entry.pack(pady=5)

    tk.Label(self.create_csv_window, text="Name:").pack(pady=5)
    self.name_entry = tk.Entry(self.create_csv_window)
    self.name_entry.pack(pady=5)

    tk.Label(self.create_csv_window, text="Birthdate (YYYY-MM-DD):").pack(pady=5)
    self.birthdate_entry = tk.Entry(self.create_csv_window)
    self.birthdate_entry.pack(pady=5)

    self.add_student_button = tk.Button(self.create_csv_window, text="Add Student", command=self.add_student_to_list)
    self.add_student_button.pack(pady=10)

    self.save_csv_button = tk.Button(self.create_csv_window, text="Save CSV", command=self.save_csv)
    self.save_csv_button.pack(pady=10)
```

Рисунок 5.9 айді

Search Student by ID- Відповідно легко зрозуміти що ця кнопочка відповідає за пошук нашого студента по його айді.

```python
def search_student(self):
    student_id = self.search_id_entry.get()
    group = self.curator_system.get_group('A1')
    if group:
        student = group.get_student(student_id)
        if student:
            self.output_text.delete( index1: 1.0, tk.END)
            self.output_text.insert(tk.END,  chars: f"ID: {student.student_id}, Name: {student.name}, Birthdate: {student.birthdate}\n")
            self.search_window.destroy()
        else:
            messagebox.showerror( title: "Error",  message: "Student not found!")
    else:
        messagebox.showerror( title: "Error",  message: "Group not found!")
```

Рисунок 5.10 відрахування

Delete Student by ID- Ось це кнопка домінантна (надає змогу відрахувати студента за його айді)

```
def delete_student(self):
    student_id = self.delete_id_entry.get()
    group = self.curator_system.get_group('A1')
    if group:
        student = group.get_student(student_id)
        if student:
            group.remove_student(student_id)
            self.curator_system.save_students_to_csv('A1')
            self.output_text.delete( index1: 1.0, tk.END)
            self.output_text.insert(tk.END,  chars: f"Student with ID {student_id} has been deleted.\n")
            self.delete_window.destroy()
        else:
            messagebox.showerror( title: "Error",  message: "Student not found!")
    else:
        messagebox.showerror( title: "Error",  message: "Group not found!")
```

Рисунок 5.11 графік пар

Show Schedule- Показує графік пар для групи на всю неділлю

```
if __name__ == "__main__":
    curator_system = CuratorSystem()

    # Add class schedules for each day from Monday to Friday
    days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
    subjects = [
        ('9:00', 'Mathematics'),
        ('11:00', 'Physics'),
        ('13:00', 'Chemistry'),
        ('9:00', 'Biology'),
        ('11:00', 'Geography'),
        ('13:00', 'History'),
        ('9:00', 'Literature'),
        ('11:00', 'Art'),
        ('13:00', 'Physical Education'),
        ('9:00', 'Computer Science'),
        ('11:00', 'Music'),
        ('13:00', 'Philosophy'),
        ('9:00', 'Economics'),
        ('11:00', 'Psychology'),
        ('13:00', 'Sociology')
    ]
```

Рисунок 5.12 Уроки

Оцінка результатів

Програма вийшла доволі складною але якщо ще прикласти зусиль та доробити візуальну частину та трошки модернізувати код то ця програма може спокійно стати робочим варіантом для кураторів академічних груп.Вона облегшить роботу як для студентів так і для їх крураторів.

реалізує систему управління студентами для куратора академічної групи з використанням мови програмування Python і бібліотеки Tkinter для створення графічного інтерфейсу користувача (GUI). Давайте оцінимо його за такими критеріями:

1. Функціональність

Код надає основні функції для управління студентами і розкладом:

Завантаження студентів з CSV-файлу.

Додавання і збереження студентів у CSV-файл.

Пошук студента за ID.

Видалення студента за ID.

Відображення списку студентів.

Відображення розкладу занять.

2. Структура і читабельність коду

Код добре організований з чіткими класами: Student, Group, Schedule, CuratorSystem та Application.

Використання методів у класах спрощує розуміння і підтримку коду.

Коментарі відсутні, що трохи ускладнює розуміння для тих, хто вперше стикається з кодом.

3. Гнучкість і розширюваність

Клас Student включає методи для додавання оцінок, запису відвідуваності та додавання відгуків, що дозволяє легко розширювати функціональність.

Клас Group управляє списком студентів, надаючи методи для додавання, видалення і пошуку студентів.

Клас Schedule дозволяє додавати заняття в розклад і отримувати його.

Клас CuratorSystem об'єднує групи і розклад, що робить систему більш модульною і легкою для розширення.

4. Користувацький інтерфейс

Інтерфейс на Tkinter надає основні функції для взаємодії з користувачем.

Присутня можливість завантажувати і зберігати дані через діалогові вікна.

Інтерфейс простий і інтуїтивно зрозумілий, але може бути покращений для більш приємного візуального сприйняття.

Порівння коду

Порівняння характеристик даної системи управління студентами з аналогічними рішеннями на ринку є важливим для розуміння її переваг і недоліків. Розглянемо основні характеристики, функціональні можливості та порівняємо їх із деякими популярними аналогами, такими як Blackboard, Moodle і Google Classroom.

1. Функціональність

Наша система

- Управління студентам: додавання, видалення, пошук студентів, запис відвідуваності.

- Управління оцінками: додавання оцінок, розрахунок середнього балу.

- Розклад: створення і відображення розкладу занять, можливість його зміни.

- Інтерфейс: графічний інтерфейс на Tkinter, простий і інтуїтивно зрозумілий.

Blackboard

- Управління студентами: повне управління студентами, групами, списками відвідуваності.

- Управління оцінками: детальне управління оцінками, автоматичний розрахунок, аналітика.

- Розклад: інтеграція з календарями, створення та управління розкладом.

- Інтерфейс: багатофункціональний, підтримує налаштування та адаптацію під користувача.

Moodle

- Управління студентами: можливість додавання, видалення та управління студентами та групами.

- Управління оцінками: система оцінок, зворотний зв'язок, аналітика успішності.

- Розклад: інтеграція з календарями, модулі для управління розкладом.

- Інтерфейс: модульний, гнучкий, підтримує налаштування та адаптацію.

Google Classroom

- Управління студентами: інтеграція з Google Workspace, управління групами.

- Управління оцінками: базові можливості оцінювання, зворотний зв'язок.

- Розклад: інтеграція з Google Calendar, створення і управління розкладом.

- Інтерфейс: простий, інтуїтивно зрозумілий, інтеграція з іншими сервісами Google.

2. Переваги і недоліки

Наша система

- Переваги:

- Простота і інтуїтивно зрозумілий інтерфейс.

- Можливість локального зберігання даних.

- Гнучкість і легкість у використанні для невеликих груп.

- Недоліки:

- Обмежена функціональність у порівнянні з комерційними рішеннями.

- Відсутність підтримки онлайн-інтеграцій і колаборaційних інструментів.

- Відсутність мобільного додатку.

Blackboard

- Переваги:

- Багатофункціональність і можливість налаштування під конкретні потреби.

- Підтримка великої кількості користувачів і великих груп.

- Інтеграція з іншими освітніми і корпоративними системами.

- Недоліки:

- Висока вартість.

- Вимогливість до технічних ресурсів.

- Складність в освоєнні для нових користувачів.

Moodle

-Переваги:

- Безкоштовність і відкритий вихідний код.

- Модульність і гнучкість у налаштуванні.

- Велика спільнота користувачів і розробників.

- Недоліки:

- Необхідність технічних знань для налаштування і адміністрування.

- Можливість уповільненої роботи при великій кількості користувачів.

- Відсутність повноцінної підтримки з боку розробників.

Google Classroom

- Переваги:

- Простота у використанні.

- Інтеграція з іншими сервісами Google.
- Безкоштовність для навчальних закладів.

- Недоліки:
- Обмежена функціональність у порівнянні з комерційними рішеннями.
- Залежність від Google Workspace.
- Відсутність можливості локального зберігання даних.

Наша система є досить простою і зручною для невеликих груп і локального використання. Вона не має багатих функціональних можливостей комерційних рішень, таких як Blackboard, але є гнучкою і легкою у використанні. Для більш великих і комплексних потреб можуть підійти більш потужні системи, такі як Moodle або Google Classroom, які мають більше функцій і інтеграцій, але можуть вимагати більших технічних знань і ресурсів.

Оцінка впливу системи на роботу куратора та академічної групи

Запропонована система управління студентами може мати значний вплив на роботу куратора та академічної групи в цілому

1. Ефективність управління даними

Покращення організації даних

Система дозволяє централізовано зберігати і управляти даними про студентів, їхні оцінки, відвідуваність та зворотній зв'язок. Це зменшує необхідність використання паперових записів або різноманітних електронних таблиць, що покращує організацію даних і зменшує ризик їх втрати.

Швидкий доступ до інформації

Куратор може легко отримувати доступ до інформації про студентів, що дозволяє швидко реагувати на запити студентів та адміністрації. Пошук за ID студента спрощує процес отримання необхідних даних, що економить час.

2. Покращення комунікації та зворотнього зв'язку

Зворотній зв'язок від студентів

Функція додавання зворотнього зв'язку дозволяє студентам залишати свої коментарі та пропозиції, що може допомогти куратору вчасно виявляти проблеми і реагувати на них. Це сприяє покращенню взаємодії між студентами та куратором.

3. Управління академічною успішністю

Моніторинг оцінок

Система дозволяє куратору легко додавати і відстежувати оцінки студентів з різних предметів. Це допомагає вчасно виявляти студентів, які мають проблеми з навчанням, та надавати їм необхідну підтримку.

Аналіз успішності

Функція розрахунку середнього балу дозволяє швидко оцінювати загальний рівень успішності студентів з окремих предметів. Це сприяє більш ефективному плануванню навчального процесу і коригуванню підходів до навчання.

4. Управління розкладом

Планування занять

Можливість створення і зміни розкладу занять дозволяє куратору більш гнучко планувати навчальний процес, враховуючи потреби студентів та доступність ресурсів. Це сприяє більш ефективному використанню навчального часу.

Інформування студентів

Відображення розкладу занять у зручному форматі допомагає студентам бути в курсі змін і планувати свій час більш ефективно. Це знижує кількість пропущених занять і покращує дисципліну.

5. Зниження навантаження на куратора

Автоматизація рутинних завдань

Система автоматизує багато рутинних завдань, таких як запис оцінок, відвідуваності та зворотнього зв'язку. Це дозволяє куратору зосередитися на більш важливих аспектах роботи, таких як індивідуальна робота зі студентами та планування навчального процесу.

Ефективне управління групою

Можливість централізованого управління групою дозволяє куратору більш ефективно координувати дії студентів, відстежувати їх успішність і своєчасно вносити необхідні корективи.

Впровадження даної системи управління студентами може значно покращити ефективність роботи куратора та академічної групи в цілому. Вона сприяє покращенню організації даних, зниженню навантаження на куратора, покращенню комунікації та моніторингу успішності студентів. Зручний доступ до інформації та можливість гнучкого управління розкладом дозволяють створити більш ефективне і гнучке навчальне середовище.

Висновки

Наша система є простим і ефективним рішенням для управління студентами та розкладом у невеликих академічних групах. Запропонована система управління студентами має значний потенціал для покращення роботи куратора та академічної групи в цілому. Вона забезпечує централізоване зберігання даних про студентів, їхні оцінки, відвідуваність та зворотний зв'язок, що суттєво покращує організацію даних і зменшує ризик їх втрати. Завдяки цьому куратору легше й швидше отримувати доступ до необхідної інформації, що сприяє оперативному реагуванню на запити студентів та адміністрації.

Функція додавання зворотного зв'язку дозволяє студентам залишати свої коментарі та пропозиції, що покращує комунікацію між студентами та куратором. Це допомагає вчасно виявляти проблеми і вирішувати їх, створюючи більш сприятливе навчальне середовище. Можливість додавання і відстеження оцінок студентів з різних предметів дозволяє куратору ефективніше контролювати академічну успішність студентів, своєчасно надаючи їм необхідну підтримку.

Система також надає інструменти для створення і зміни розкладу занять, що дозволяє більш гнучко планувати навчальний процес. Це особливо важливо в умовах змінного навчального середовища, де необхідно оперативно реагувати на різні обставини. Відображення розкладу занять у зручному форматі допомагає студентам бути в курсі змін і планувати свій час більш ефективно, що знижує кількість пропущених занять і покращує дисципліну.

Автоматизація рутинних завдань, таких як запис оцінок, відвідуваності та зворотного зв'язку, значно знижує навантаження на куратора, дозволяючи йому зосередитися на більш важливих аспектах роботи, таких як індивідуальна робота зі студентами та планування навчального процесу. Це сприяє зниженню ймовірності помилок і підвищенню загальної ефективності.

Порівняння з аналогічними рішеннями на ринку, такими як Blackboard, Moodle і Google Classroom, показує, що наша система є гнучкою, простою у використанні і підходить для невеликих академічних груп. Вона не має всіх можливостей комерційних рішень, але забезпечує необхідну функціональність для ефективного управління студентами та навчальним процесом. Впровадження такої системи може значно покращити організацію навчального процесу, підвищити академічну успішність студентів і створити більш структуроване і ефективне навчальне середовище.

Загалом, дана система є важливим інструментом для куратора і може значно полегшити його роботу, сприяючи підвищенню ефективності управління академічною групою. Це, в свою чергу, позитивно впливає на навчальний процес і створює сприятливі умови для успішного навчання студентів.

Список використаних джерел

Blackboard Inc. "Blackboard Learn: Education Technology & Services".
[blackboard.com](https://www.blackboard.com)

Moodle Pty Ltd. "Moodle - Open-source learning platform". moodle.org

Google LLC."Google Classroom". classroom.google.com

Python Software Foundation. "Tkinter - Python interface to Tcl/Tk". [docs.python.org](https://docs.python.org/3/library/tkinter.html).

Guido van Rossum and the Python community. "Python Programming Language".
[python.org](https://www.python.org)

Csv Module - Python Standard Library Documentation. [docs.python.org](https://docs.python.org/3/library/csv.html)

Tkinter Documentation by Fredrik Lundh.
[effbot.org](http://effbot.org/tkinterbook/)

Datetime Module - Python Standard Library Documentation. [docs.python.org](https://docs.python.org/3/library/datetime.html)

Wiki- (https://uk.wikipedia.org/wiki/)

Samoosvita – (https://stlnau.in.ua/samoosvita/item/2018/nmc180425.pdf)

ДОДАТКИ

Додаток А

- **CSV**

```
import csv
import os
from random import randint
from datetime import date, timedelta

# Функция для генерации случайных дат рождения
def random_birthdate(start_year=1995, end_year=2005):
    start_date = date(start_year, 1, 1)
    end_date = date(end_year, 12, 31)
    delta = end_date - start_date
    random_days = randint(0, delta.days)
    birthdate = start_date + timedelta(days=random_days)
    return birthdate.strftime('%Y-%m-%d')

# Список украинских имен и фамилий
names = [
    'Іван Іванов', 'Ольга Петрова', 'Сергій Сидоров', 'Анна Павлова', 'Дмитро Козлов',
```

'Марія Смирнова', 'Микола Волков', 'Олена Федорова', 'Олексій Морозов', 'Тетяна Іванова',

'Андрій Петров', 'Ірина Сидорова', 'Володимир Кузнєцов', 'Наталія Михайлова', 'Кирило Павлов',

'Світлана Новикова', 'Юрій Лебедєв', 'Катерина Сергєєва', 'Олег Зайцев', 'Анастасія Ніколаєва',

'Михайло Попов', 'Людмила Соколова', 'Олександра Орлова', 'Віктор Романов', 'Єлизавета Степанова',

'Ігор Макаров', 'Валерія Дмитрієва', 'Павло Єгоров', 'Надія Васильєва', 'Максим Богданов'

]

```
# Генерация списка из 30 студентов
students = []
for i in range(1, 31):
    student_id = str(i)
    name = names[i-1]
    birthdate = random_birthdate()
    students.append({'ID': student_id, 'Name': name, 'Birthdate': birthdate})
```

```
# Указываем путь к файлу
directory = r'C:\Users\Le_II\Desktop'
csv_file_name = 'students.csv'
csv_file_path = os.path.join(directory, csv_file_name)

try:
    with open(csv_file_path, mode='w', newline='') as csvfile:
        fieldnames = ['ID', 'Name', 'Birthdate']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()
        for student in students:
            writer.writerow(student)

    print(f"Файл {csv_file_path} створено.")
except PermissionError:
    print(f"Не вдалося відкрити файл {csv_file_path}. Будь ласка, закрийте його в інших програмах і спробуйте знову.")
except Exception as e:
    print(f"Сталася помилка: {e}")
```

- **Основний код**

```
import csv
from datetime import datetime, date
import tkinter as tk
from tkinter import filedialog, messagebox, ttk


class Student:
    def __init__(self, student_id, name, birthdate):
        self.student_id = student_id
        self.name = name
        self.birthdate = datetime.strptime(birthdate, '%Y-%m-%d').date()
        self.grades = {}
        self.attendance = {}
        self.feedback = []

    def add_grade(self, subject, grade):
        if subject not in self.grades:
            self.grades[subject] = []
        self.grades[subject].append(grade)
```

```
    def get_average_grade(self, subject):
        if subject in self.grades and self.grades[subject]:
            return sum(self.grades[subject]) / len(self.grades[subject])
        return None

    def record_attendance(self, date, present):
        self.attendance[date] = present

    def add_feedback(self, message):
        self.feedback.append(message)

    def get_feedback(self):
        return self.feedback

class Group:
    def __init__(self, group_id):
        self.group_id = group_id
        self.students = {}

    def add_student(self, student):
        self.students[student.student_id] = student
```

					Комп'ютерна система для забезпечення роботи куратора академічної групи	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Молдован.О.В.*						
Перевір.		*Плахов О.О*				*Арк.* 1	*Аркушів* 1	
Т. Контр.								
Реценз.						*Циклова комісія Комп'ютерної інженерії*		
Н. Контр.								
Затверд.								

```
    def get_student(self, student_id):
        return self.students.get(student_id, None)

    def remove_student(self, student_id):
        if student_id in self.students:
            del self.students[student_id]

    def get_all_students(self):
        return self.students.values()

class Schedule:
    def __init__(self):
        self.timetable = {}

    def add_class(self, day, time, subject):
        if day not in self.timetable:
            self.timetable[day] = []
        self.timetable[day].append((time, subject))

    def get_schedule(self):
        return self.timetable
```

```
    def update_class(self, day, old_time, new_time, new_subject):
        if day in self.timetable:
            for i, (time, subject) in enumerate(self.timetable[day]):
                if time == old_time:
                    self.timetable[day][i] = (new_time, new_subject)
                    break

class CuratorSystem:
    def __init__(self):
        self.groups = {}
        self.schedule = Schedule()

    def add_group(self, group):
        self.groups[group.group_id] = group

    def get_group(self, group_id):
        return self.groups.get(group_id, None)
```

```
def load_students_from_csv(self, group_id, csv_file):
    group = self.get_group(group_id)
    if group:
        try:
            with open(csv_file, newline='') as csvfile:
                reader = csv.DictReader(csvfile)
                for row in reader:
                    student = Student(row['ID'], row['Name'], row['Birthdate'])
                    group.add_student(student)
            self.csv_file = csv_file  # Save the CSV file path for later use
        except Exception as e:
            messagebox.showerror("Error", f"Failed to read CSV file: {e}")

def save_students_to_csv(self, group_id):
    group = self.get_group(group_id)
    if group and hasattr(self, 'csv_file'):
        try:
            with open(self.csv_file, mode='w', newline='') as csvfile:
                fieldnames = ['ID', 'Name', 'Birthdate']
                writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

```
                writer.writeheader()
                for student in group.get_all_students():
                    writer.writerow({
                        'ID': student.student_id,
                        'Name': student.name,
                        'Birthdate': student.birthdate.strftime('%Y-%m-%d')
                    })
            except Exception as e:
                messagebox.showerror("Error", f"Failed to save CSV file: {e}")

    def add_class_to_schedule(self, day, time, subject):
        self.schedule.add_class(day, time, subject)

    def get_schedule(self):
        return self.schedule.get_schedule()

    def update_class_in_schedule(self, day, old_time, new_time, new_subject):
        self.schedule.update_class(day, old_time, new_time, new_subject)
```

```
class Application(tk.Tk):
    def __init__(self, curator_system):
        super().__init__()
        self.curator_system = curator_system
        self.title("Curator System")
        self.geometry("800x600")

        self.create_widgets()
        self.show_schedule()

    def create_widgets(self):
        self.load_button = tk.Button(self, text="Load Students from CSV", command=self.load_students)
        self.load_button.pack(pady=10)

        self.create_csv_button = tk.Button(self, text="Create and Save CSV", command=self.create_and_save_csv)
        self.create_csv_button.pack(pady=10)

        self.search_button = tk.Button(self, text="Search Student by ID", command=self.search_student_by_id)
        self.search_button.pack(pady=10)
```

```
        self.delete_button = tk.Button(self, text="Delete Student by ID",
command=self.delete_student_by_id)
        self.delete_button.pack(pady=10)

        self.schedule_button = tk.Button(self, text="Show Schedule",
command=self.show_schedule_window)
        self.schedule_button.pack(pady=10)

        self.update_schedule_button = tk.Button(self, text="Update Schedule",
command=self.update_schedule_window)
        self.update_schedule_button.pack(pady=10)

        self.output_text = tk.Text(self, wrap=tk.WORD)
        self.output_text.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)

    def load_students(self):
        csv_file = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
        if csv_file:
            group_id = 'A1'
            if not self.curator_system.get_group(group_id):
                group = Group(group_id)
                self.curator_system.add_group(group)
```

					Комп'ютерна система для забезпечення роботи куратора академічної групи	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Молдован.О.В.*						
Перевір.		*Плахов О.О*						
Т. Контр.						*Арк.* 1	*Аркушів* 1	
Реценз.								
Н. Контр.						*Циклова комісія Комп'ютерної інженерії*		
Затверд.								

```
            self.curator_system.load_students_from_csv(group_id, csv_file)
            self.show_students()

        def create_and_save_csv(self):
            self.create_csv_window = tk.Toplevel(self)
            self.create_csv_window.title("Create CSV")
            self.create_csv_window.geometry("400x300")

            tk.Label(self.create_csv_window, text="ID:").pack(pady=5)
            self.id_entry = tk.Entry(self.create_csv_window)
            self.id_entry.pack(pady=5)

            tk.Label(self.create_csv_window, text="Name:").pack(pady=5)
            self.name_entry = tk.Entry(self.create_csv_window)
            self.name_entry.pack(pady=5)

            tk.Label(self.create_csv_window, text="Birthdate (YYYY-MM-
DD):").pack(pady=5)
            self.birthdate_entry = tk.Entry(self.create_csv_window)
            self.birthdate_entry.pack(pady=5)
```

```
        self.add_student_button = tk.Button(self.create_csv_window, text="Add Student", command=self.add_student_to_list)
        self.add_student_button.pack(pady=10)

        self.save_csv_button = tk.Button(self.create_csv_window, text="Save CSV", command=self.save_csv)
        self.save_csv_button.pack(pady=10)

        self.students = []

    def add_student_to_list(self):
        student_id = self.id_entry.get()
        name = self.name_entry.get()
        birthdate = self.birthdate_entry.get()

        if not student_id or not name or not birthdate:
            messagebox.showerror("Error", "Please fill all fields!")
            return
```

```
            try:
                datetime.strptime(birthdate, '%Y-%m-%d')
            except ValueError:
                messagebox.showerror("Error", "Incorrect date format! Use YYYY-MM-DD.")
                return

            self.students.append({'ID': student_id, 'Name': name, 'Birthdate': birthdate})
            messagebox.showinfo("Success", "Student added successfully!")
            self.id_entry.delete(0, tk.END)
            self.name_entry.delete(0, tk.END)
            self.birthdate_entry.delete(0, tk.END)

        def save_csv(self):
            csv_file = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files", "*.csv")])
            if csv_file:
                try:
                    with open(csv_file, mode='w', newline='') as csvfile:
                        fieldnames = ['ID', 'Name', 'Birthdate']
                        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

					Комп'ютерна система для забезпечення роботи куратора академічної групи	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Молдован.О.В.*						
Перевір.		*Плахов О.О*				*Арк.* 1	*Аркушів* 1	
Т. Контр.								
Реценз.						*Циклова комісія Комп'ютерної інженерії*		
Н. Контр.								
Затверд.								

```
                writer.writeheader()
                for student in self.students:
                    writer.writerow(student)

                messagebox.showinfo("Success", "CSV file saved successfully!")
                self.create_csv_window.destroy()
            except Exception as e:
                messagebox.showerror("Error", f"Failed to save CSV file: {e}")

        def search_student_by_id(self):
            self.search_window = tk.Toplevel(self)
            self.search_window.title("Search Student")
            self.search_window.geometry("400x200")

            tk.Label(self.search_window, text="Enter Student ID:").pack(pady=5)
            self.search_id_entry = tk.Entry(self.search_window)
            self.search_id_entry.pack(pady=5)

            self.search_button = tk.Button(self.search_window, text="Search",
command=self.search_student)
            self.search_button.pack(pady=10)
```

					Комп'ютерна система для забезпечення роботи куратора академічної групи	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Молдован.О.В.*						
Перевір.		*Плахов О.О*				*Арк.* 1	*Аркушів* 1	
Т. Контр.								
Реценз.						*Циклова комісія Комп'ютерної інженерії*		
Н. Контр.								
Затверд.								

```
        def search_student(self):
            student_id = self.search_id_entry.get()
            group = self.curator_system.get_group('A1')
            if group:
                student = group.get_student(student_id)
                if student:
                    self.output_text.delete(1.0, tk.END)
                    self.output_text.insert(tk.END, f"ID: {student.student_id}, Name:
{student.name}, Birthdate: {student.birthdate}\n")
                    self.search_window.destroy()
                else:
                    messagebox.showerror("Error", "Student not found!")
            else:
                messagebox.showerror("Error", "Group not found!")

        def delete_student_by_id(self):
            self.delete_window = tk.Toplevel(self)
            self.delete_window.title("Delete Student")
            self.delete_window.geometry("400x200")
```

```
        tk.Label(self.delete_window, text="Enter Student ID:").pack(pady=5)
        self.delete_id_entry = tk.Entry(self.delete_window)
        self.delete_id_entry.pack(pady=5)

        self.delete_button = tk.Button(self.delete_window, text="Delete",
command=self.delete_student)
        self.delete_button.pack(pady=10)

    def delete_student(self):
        student_id = self.delete_id_entry.get()
        group = self.curator_system.get_group('A1')
        if group:
            student = group.get_student(student_id)
            if student:
                group.remove_student(student_id)
                self.curator_system.save_students_to_csv('A1')
                self.output_text.delete(1.0, tk.END)
                self.output_text.insert(tk.END, f"Student with ID {student_id} has
been deleted.\n")
```

					Комп'ютерна система для забезпечення роботи куратора академічної групи	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Молдован.О.В.*						
Перевір.		*Плахов О.О*						
Т. Контр.						*Арк.* 1	*Аркушів* 1	
Реценз.								
Н. Контр.						*Циклова комісія Комп'ютерної інженерії*		
Затверд.								

```
                self.delete_window.destroy()
            else:
                messagebox.showerror("Error", "Student not found!")
        else:
            messagebox.showerror("Error", "Group not found!")

    def show_students(self):
        group = self.curator_system.get_group('A1')
        if group:
            self.output_text.delete(1.0, tk.END)
            for student in group.get_all_students():
                self.output_text.insert(tk.END, f"ID: {student.student_id}, Name:
{student.name}, Birthdate: {student.birthdate}\n")

    def show_schedule(self):
        self.output_text.delete(1.0, tk.END)
        schedule = self.curator_system.get_schedule()
        today = datetime.now().strftime('%A')
```

```
        for day, classes in schedule.items():
            if day == today:
                self.output_text.insert(tk.END, f"Day: {day}\n", 'highlight')
            else:
                self.output_text.insert(tk.END, f"Day: {day}\n")
            for time, subject in classes:
                self.output_text.insert(tk.END, f"  {time}: {subject}\n")

        self.output_text.tag_configure('highlight', background='yellow')

    def show_schedule_window(self):
        schedule_window = tk.Toplevel(self)
        schedule_window.title("Schedule")
        schedule_window.geometry("400x400")

        schedule_text = tk.Text(schedule_window, wrap=tk.WORD)
        schedule_text.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)
```

```
        schedule = self.curator_system.get_schedule()
        for day, classes in schedule.items():
            schedule_text.insert(tk.END, f"Day: {day}\n")
            for time, subject in classes:
                schedule_text.insert(tk.END, f"  {time}: {subject}\n")

    def update_schedule_window(self):
        self.update_window = tk.Toplevel(self)
        self.update_window.title("Update Schedule")
        self.update_window.geometry("400x300")

        tk.Label(self.update_window, text="Day:").pack(pady=5)
        self.day_entry = tk.Entry(self.update_window)
        self.day_entry.pack(pady=5)

        tk.Label(self.update_window, text="Old Time:").pack(pady=5)
        self.old_time_entry = tk.Entry(self.update_window)
        self.old_time_entry.pack(pady=5)

        tk.Label(self.update_window, text="New Time:").pack(pady=5)
        self.new_time_entry = tk.Entry(self.update_window)
        self.new_time_entry.pack(pady=5)
```

```
        tk.Label(self.update_window, text="New Subject:").pack(pady=5)
        self.new_subject_entry = tk.Entry(self.update_window)
        self.new_subject_entry.pack(pady=5)

        self.update_class_button = tk.Button(self.update_window, text="Update
Class", command=self.update_class)
        self.update_class_button.pack(pady=10)

    def update_class(self):
        day = self.day_entry.get()
        old_time = self.old_time_entry.get()
        new_time = self.new_time_entry.get()
        new_subject = self.new_subject_entry.get()

        if not day or not old_time or not new_time or not new_subject:
            messagebox.showerror("Error", "Please fill all fields!")
            return
```

					Комп'ютерна система для забезпечення роботи куратора академічної групи	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Молдован.О.В.*						
Перевір.		*Плахов О.О*				*Арк.* 1	*Аркушів* 1	
Т. Контр.								
Реценз.						*Циклова комісія Комп'ютерної інженерії*		
Н. Контр.								
Затверд.								

```
            self.curator_system.update_class_in_schedule(day, old_time, new_time,
new_subject)
            messagebox.showinfo("Success", "Class updated successfully!")
            self.update_window.destroy()
            self.show_schedule()

        if __name__ == "__main__":
            curator_system = CuratorSystem()

            # Add class schedules for each day from Monday to Friday
            days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
            subjects = [
```

```
    ('9:00', 'Mathematics'),
    ('11:00', 'Physics'),
    ('13:00', 'Chemistry'),
    ('9:00', 'Biology'),
    ('11:00', 'Geography'),
    ('13:00', 'History'),
    ('9:00', 'Literature'),
    ('11:00', 'Art'),
    ('13:00', 'Physical Education'),
    ('9:00', 'Computer Science'),
    ('11:00', 'Music'),
    ('13:00', 'Philosophy'),
    ('9:00', 'Economics'),
    ('11:00', 'Psychology'),
    ('13:00', 'Sociology')
]
```

```
        for i, day in enumerate(days):
            curator_system.add_class_to_schedule(day, subjects[i*3][0], subjects[i*3][1])
            curator_system.add_class_to_schedule(day, subjects[i*3+1][0], subjects[i*3+1][1])
            curator_system.add_class_to_schedule(day, subjects[i*3+2][0], subjects[i*3+2][1])

        app = Application(curator_system)
        app.mainloop()
```

					Комп'ютерна система для забезпечення роботи куратора академічної групи	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Молдован.О.В.*						
Перевір.		*Плахов О.О*				*Арк.* 1	*Аркушів* 1	
Т. Контр.								
Реценз.						*Циклова комісія Комп'ютерної інженерії*		
Н. Контр.								
Затверд.								