ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для вивчення курсу з кібербезпеки мовою Python з використанням фреймворку Django»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Денис КЕДА
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Денис КЕДА

Керівник: Ігор АВЕРІЧЕВ
к.е.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кеді Денису Олеговичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку для вивчення курсу з кібербезпеки мовою Python з використанням фреймворку Django»
керівник кваліфікаційної роботи к.е.н., доцент кафедри ІПЗ Ігор АВЕРІЧЕВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні дані, технічна документація.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області.
2. Теоретичні основи розробку мобільного застосунку.
3. Розробка мобільного застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма послідовності.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження.

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих алгоритмів та методів для створення мобільного застосунку з курсом з кібербезпеки	14.03-20.03.2024	
4	Проектування мобільного застосунку для вивчення курсу з кібербезпеки	21.03-31.03.2024	
5	Програмна реалізація мобільного застосунку	01.04-21.04.2024	
6	Тестування мобільного застосунку	22.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Денис КЕДА
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор АВЕРІЧЕВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 59 стор., 7 табл., 7 рис., 15 джерел.

Мета роботи – поліпшення доступності процесу навчання кібербезпеки шляхом створення інтерактивного мобільного застосунку.

Об'єкт дослідження – процес навчання курсу з кібербезпеки.

Предмет дослідження – мобільний застосунок для навчання курсу з кібербезпеки.

Короткий зміст роботи: В роботі проаналізовано алгоритми та методи для створення мобільного застосунку з курсом з кібербезпеки, реалізованого мовою Python та базованого на фреймворку Django. Проаналізовано основні вимоги до застосунку, специфіку навчального контенту. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: можливість реєструватися та авторизуватися користувачам, переглядати навчальні матеріали курсу з кібербезпеки, проходити вікторини для перевірки знань, пройти підсумковий тест по завершенню курсу та переглядати статистику та прогрес користувача у вивченні курсу з кібербезпеки. Проведено функціональне тестування додатку. В роботі використано фреймворк Django для розробки серверної частини, який забезпечує швидку розробку веб-застосунків з використанням мови програмування Python та надійну підтримку безпеки.

Сферою використання застосунку є навчання та підвищення обізнаності у сфері кібербезпеки через мобільний додаток.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, КІБЕРБЕЗПЕКА, PYTHON, DJANGO.

ЗМІСТ

ВСТУП

У світі, де все більше людей користуються Інтернетом, питання кібербезпеки стає дедалі більш актуальним. Хакери постійно розробляють нові способи атак, щоб отримати доступ до конфіденційної інформації, пошкодити комп'ютерні системи або викрасти гроші. Тому важливо, щоб люди знали, як захистити себе від цих загроз.

Існує безліч способів вивчити кібербезпеку, таких як читання книг, проходження курсів або перегляд онлайн-туторіалів. Проте, мобільні додатки стають все більш популярними для навчання, завдяки своїй зручності та доступності. Можна використовувати мобільні додатки для вивчення кібербезпеки в будь-який час і в будь-якому місці, що робить їх ідеальним вибором для зайнятих людей.

Розробка мобільного додатку для вивчення курсу з кібербезпеки може допомогти людям освоїти основи цієї галузі. Додаток містить інформацію про різні види кіберзагроз, такі як фішинг, шкідливі програми та DDoS-атаки. Крім того, він може навчити користувачів захищати себе від цих загроз за допомогою таких методів, як складні паролі, двофакторна аутентифікація та регулярні оновлення програмного забезпечення.

Отже, розробка мобільного додатку для вивчення курсу з кібербезпеки мовою Python з використанням фреймворку Django може стати цінним інструментом для людей, які бажають дізнатися більше про цю важливу тему.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний огляд кібербезпеки

Захист інформаційних систем, мереж та даних від несанкціонованого доступу, використання, розголошення, порушень, змін або знищення є ключовим аспектом кібербезпеки. Це актуальне завдання для індивідуумів, компаній та урядових організацій, оскільки атаки кіберзлочинців можуть призвести до серйозної шкоди, включаючи викрадення особистих даних, завади в роботі систем або навіть фізичні пошкодження.

На сьогодні існує різноманітні види кіберзагроз, включаючи**:**

1. Шкідливе програмне забезпечення, яке призначене для завдання шкоди комп'ютерній системі, таке як віруси, хробаки, троянські коні, шпигунське та зловмисне програмне забезпечення.
2. Атака фішингу, що полягає у шахрайстві для отримання особистих даних, таких як паролі або номери кредитних карток, часто виконується за допомогою підроблених електронних листів або веб-сайтів.
3. Крадіжка даних, що включає у себе заволодіння особистою інформацією, такою як номери соціального страхування або кредитних карток, шляхом кібератак, фізичних втручань або соціальної інженерії.
4. Відмова в обслуговуванні (DoS), яка полягає в переповненні веб-сайтів або сервісів трафіком, що робить їх недоступними для користувачів.
5. Людська помилка, яка є однією з основних причин кіберзлочинів, і може включати в себе використання слабких паролів або ненавмисний перехід за сумнівними посиланнями.

Для захисту від цих загроз важливо:

- Використовувати складні паролі, що містять різноманітні символи і цифри, і уникати повторного використання одного пароля для різних сервісів.

- Регулярно оновлювати програмне забезпечення, включаючи операційні системи та веб-браузери, щоб мати найновіші заходи безпеки.
- Встановлювати антивірусне та антишпигунське програмне забезпечення та забезпечувати його регулярне оновлення.
- Бути обережними при кліканні на посилання та завантаженні веб-сторінок або файлів, особливо з ненадійних джерел.
- Тримати особисті дані в безпеці, обмежуючи їх розголошення в Інтернеті та використовуючи надійні методи захисту, такі як унікальні паролі.

Кібербезпека є постійною проблемою, і важливо бути завжди в курсі останніх загроз та вживати заходів для захисту себе.

1.2 Огляд існуючих методів вивчення кібербезпеки

Забезпечення безпеки в Інтернеті потребує постійного удосконалення та реагування на нові загрози. Для цього необхідно детально вивчати наявні методи та практики. Існує безліч способів вивчення кібербезпеки, кожен з яких має свої переваги та недоліки. У цьому розділі ми проаналізуємо традиційні та інноваційні методи вивчення кібербезпеки.

Традиційні методи навчання:

1. Уроки та дискусії: цей класичний підхід включає виклад матеріалу викладачем, за яким слідує обговорення на групових зустрічах. Переваги цього методу включають чітку структуру, можливість отримати пояснення від професіоналів та спілкування з однокурсниками. Однак недоліками є пасивність студентів, обмежений час на практичні вправи та відсутність індивідуального підходу.
2. Самостійна робота: вивчення матеріалів з підручників, методичних посібників, онлайн-курсів та інших джерел. Переваги цього методу включають гнучкість, можливість навчатися у власному темпі та докладно вивчати цікаві теми. Проте недоліки включають відсутність зворотного

зв'язку з викладачем, складнощі самостійного засвоєння складних тем та можливість помилок у розумінні матеріалу.

3. Лабораторні роботи: виконання практичних завдань на комп'ютері під керівництвом викладача або лаборанта. Переваги цього методу включають набуття практичних навичок, закріплення теоретичних знань та можливість отримання допомоги у вирішенні проблем. Однак недоліки включають обмежений час на виконання завдань, необхідність комп'ютера та програмного забезпечення.

Інноваційні методи навчання:

1. Онлайн-курси: вивчення матеріалу за допомогою платформ дистанційного навчання. Переваги цього методу включають гнучкість, доступність з будь-якої точки світу, можливість навчатися у власному темпі та наявність інтерактивних елементів. Недоліки включають відсутність прямого контакту з викладачем, складнощі самостійної мотивації та можливість технічних проблем.
2. Ігри та симуляції: вивчення матеріалу за допомогою ігрових платформ, що імітують реальні сценарії кібербезпеки. Переваги цього методу включають інтерактивність, захопливість та можливість отримання практичного досвіду у безпечному середовищі. Недоліки включають обмеженість тематики, складнощі розробки якісних ігрових платформ та можливість відволікання від навчання.
3. Віртуальна та доповнена реальність: вивчення матеріалу за допомогою технологій віртуальної та доповненої реальності, що створюють імітацію реального світу. Переваги цього методу включають реалістичність, отримання практичного досвіду в онлайн середовищі та підвищення мотивації до навчання. Недоліки включають високу вартість обладнання, складнощі розробки віртуальних середовищ та можливість виникнення кінетозу.

Вибір методу навчання кібербезпеки залежить від цілей, стилю навчання студента, наявних ресурсів та інших факторів. Традиційні методи, такі як уроки та

дискусії, підходять для отримання базових знань та навичок. Інноваційні методи, такі як онлайн-курси, ігри та симуляції, можуть бути ефективними для засвоєння складних тем та отримання практичного досвіду.

1.3 Переваги використання мобільних застосунків для навчання

Мобільні застосунки для навчання стають дедалі популярнішими завдяки численним перевагам, які вони пропонують учням. Ось деякі з ключових переваг:

1. Доступність: мобільні застосунки роблять навчання більш доступним, оскільки можна мати доступ до них на будь-якому смартфоні чи планшеті. Це особливо зручно для студентів зі складним графіком або проживанням у віддалених районах. Вони надають можливість вчитися в будь-який час і в будь-якому місці. Також важливо зазначити, що багато з таких застосунків доступні безкоштовно або за низьку ціну, порівняно з традиційними методами навчання, які часто вимагають великих витрат на книги та курси.
2. Гнучкість: мобільні застосунки дозволяють учням навчатися у власному темпі та в зручний для них час. Вони можуть переглядати матеріали курсу, проходити вікторини та виконувати завдання в будь-який час, коли їм це зручно.
3. Персоналізація: багато мобільних застосунків для навчання пропонують персоналізований досвід навчання, який адаптований до потреб індивідуального учня. Це може включати рекомендації щодо курсу, адаптивні тести та персоналізований зворотний зв'язок.
4. Інтерактивність: мобільні застосунки можуть зробити навчання більш інтерактивним та захоплюючим за допомогою ігор, вікторин, симуляцій та інших інтерактивних елементів. Це може допомогти учням краще засвоїти матеріал і зробити процес навчання більш приємним.
5. Зручність використання: мобільні застосунки, зазвичай є легкими у використанні та навігації, що робить їх зручними для користувачів усіх рівнів технічної підготовки.

6. Соціальна взаємодія: деякі мобільні застосунки для навчання пропонують функції соціальної взаємодії, які дозволяють учням спілкуватися з однокласниками, обговорювати навчальні матеріали та співпрацювати над проектами. Це може допомогти учням відчути себе більш згуртованими та підтримуваними, а також покращити їх загальний досвід навчання.
7. Офлайн-доступ: деякі мобільні застосунки для навчання дозволяють учням завантажувати навчальні матеріали для доступу в режимі офлайн. Це може бути корисно для учнів, які не мають постійного доступу до Інтернету.
8. Аналітика та відстеження: мобільні застосунки для навчання можуть надавати учням та викладачам дані про аналітику та відстеження, які можуть допомогти відстежувати прогрес, визначати області, які потребують покращення, та вносити необхідні корективи у навчальний процес.
9. Портативність: мобільні пристрої мають невеликі розміри та легкі, що робить їх портативними, що дозволяє учням навчатися в будь-якому місці.

Мобільні застосунки для навчання пропонують низку переваг, які роблять їх цінним інструментом для учнів усіх віків та рівнів освіти. Завдяки своїй доступності, гнучкості, персоналізації, інтерактивності, доступності, зручності використання, соціальній взаємодії, офлайн-доступу, аналітиці та відстеженню, а також портативності, мобільні застосунки можуть зробити навчання більш ефективним, приємним та доступним.

1.4 Визначення мети та завдань дослідження

Мета цього дослідження полягає в створенні мобільного додатка для навчання кібербезпеці з використанням Python та фреймворку Django. Мобільний додаток має стати зручним, інтерактивним та інформативним інструментом для всіх, хто цікавиться основами кібербезпеки.

Для досягнення цієї мети необхідно виконати наступні завдання:

1. Проаналізувати існуючі мобільні додатки для навчання кібербезпеці: оцінити можливості, дизайн та користувацький інтерфейс існуючих додатків.

2. Визначити потреби та вимоги цільової аудиторії: провести дослідження, щоб з'ясувати, що саме потрібно користувачам від мобільного додатка з курсу для вивчення кібербезпеки.
3. Розробити концепцію мобільного додатка: визначити функціонал, дизайн та технологічний стек додатка.
4. Спроектувати та розробити мобільний додаток: реалізувати додаток з використанням Django, дотримуючись принципів мобільного дизайну.
5. Заповнити додаток навчальним контентом: розробити та включити до додатка текстові матеріали, інтерактивні завдання та тести з кібербезпеки.
6. Провести тестування мобільного додатка: перевірити додаток на помилки та оцінити його зручність та ефективність навчання.

Результатом дослідження буде:

- Мобільний додаток для навчання кібербезпеці на базі Python та Django.
- Навчальний контент у доступному та інтерактивному форматі.
- Інструмент для вивчення кібербезпеки для широкого кола користувачів.

Наукова новизна полягає в розробці такого додатка, що поєднує зручність мобільних пристроїв з інтерактивністю та інформативністю навчального контенту.

Практичне значення дослідження полягає в тому, що цей додаток стане корисним інструментом для підвищення рівня обізнаності в кібербезпеці серед користувачів, тим самим допомагаючи зменшити ризики кіберзлочинів. Шляхом надання доступу до навчального контенту та інтерактивних завдань з кібербезпеки, додаток сприятиме усвідомленню загроз та методів захисту, забезпечуючи користувачам необхідні знання для ефективного захисту їхніх особистих даних та пристроїв в онлайн середовищі.

1.5 Аналіз аналогів

Для оцінки програмних засобів для вивчення курсу з кібербезпеки, можна розглянути такі платформи, як CyberSmart, SANS Institute та Чемпіони кібербезпеки.

CyberSmart - це веб-платформа, яка пропонує навчання та підготовку з кібербезпеки для людей на різних рівнях. Вміст платформи розроблений фахівцями у галузі кібербезпеки і включає широкий спектр тем, таких як основи кібербезпеки, етичне хакерство, тестування на проникнення та багато іншого. На рисунку 1.1 показано інтерфейс додатку.

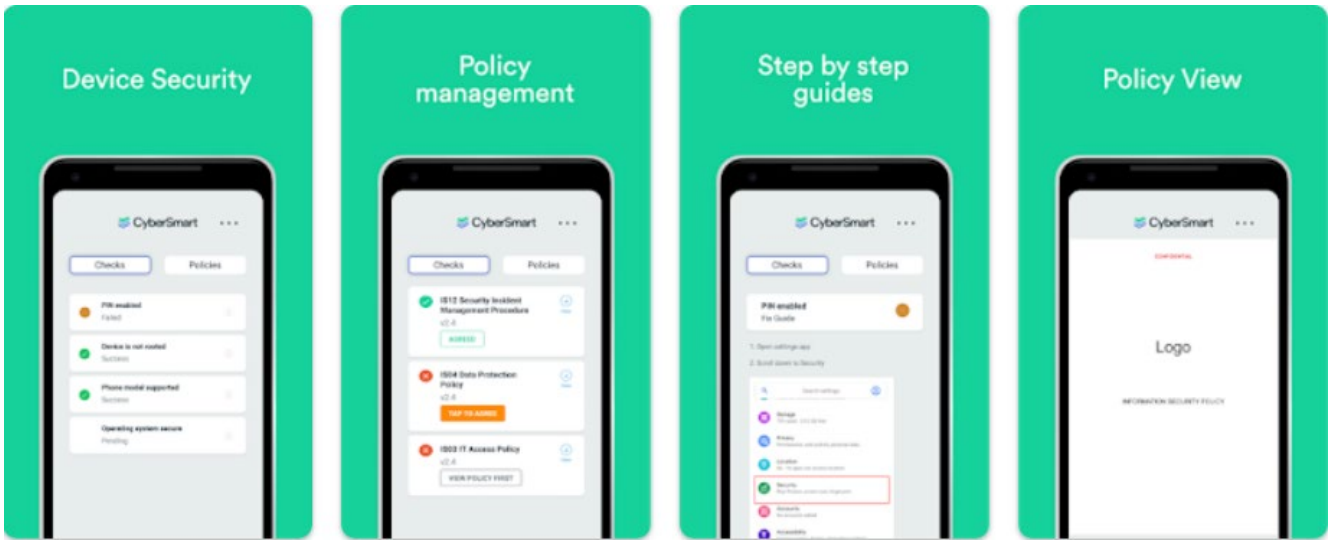


Рис. 1.1 Інтерфейс додатку CyberSmart

SANS Institute - це організація без прибуткової мети, яка пропонує широкий спектр курсів та сертифікацій у галузі кібербезпеки. Її курси вважаються одними з найкращих у цьому сегменті та спрямовані на професіоналів у галузі кібербезпеки. Інтерфейс додатку рисунок 1.2.

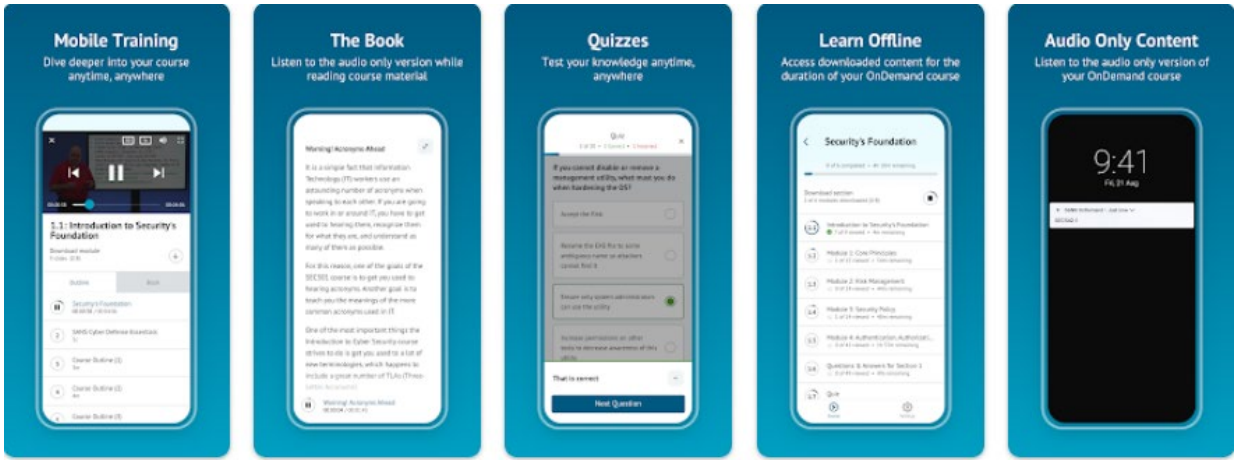


Рис. 1.2 Інтерфейс додатку SANS Institute

«Чемпіонат з кібербезпеки» - це освітньо-превентивна гра у форматі вікторини, створена Кіберполіцією Сумської області спільно з громадськими та урядовими органами регіону. Гру можна грати через мобільний додаток для Android. Інтерфейс гри рисунок 1.3.

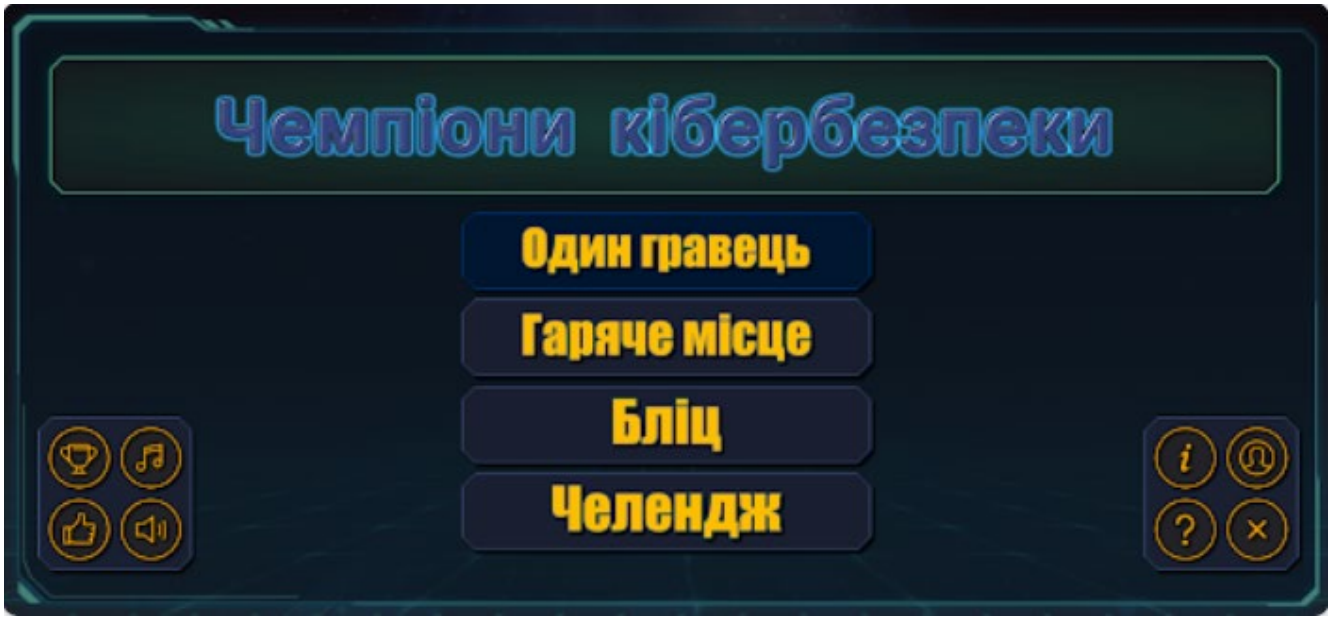


Рис. 1.3 Інтерфейс гри «Чемпіонат з кібербезпеки»

Результати аналізу основних характеристик розглянутих додатків подано в таблиці 1.1.

Таблиця 1.1

Порівняльна таблиця мобільних додатків для вивчення кібербезпеки

Характеристика	CyberSmart	SANS Institute	Чемпіони з кібербезпеки	CyberSavvy
Інтерактивний навчальний контент	-	+	+	+
Відстеження прогресу	-	-	-	+
Надання теоретичної інформації з кібербезпеки	+	+	-	+
Тести для перевірки знань	-	+	-	+
Підтримка української мови	+	-	+	+

2. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Опис застосованих технологій і мов програмування

Вибір мови програмування та фреймворку є важливим етапом розробки будь-якого мобільного застосунку. Мовою програмування, обраною для розробки мобільного застосунку, є Python.

Python – це програмна мова, яку легко освоїти і яка пропонує високорівневі ефективні структури даних. Завдяки простому, але потужному підходу до об’єктно-орієнтованого програмування, Python стає привабливим вибором для багатьох розробників. Його витончений синтаксис і динамічна типізація, а також можливість інтерпретації роблять його ідеальним для написання скриптів і швидкої розробки додатків у різних сферах на різних платформах. Крім того, Python має активну та обширну спільноту користувачів, а також чудово структуровану та доступну документацію. Python також має багато бібліотек та фреймворків, які можна використовувати для розробки мобільних застосунків, включаючи Django.

Python має численні переваги, серед яких простота - він вважається однією з найпростіших мов програмування для вивчення, що особливо привабливо для початківців; читабельність - чіткий та лаконічний синтаксис Python робить код зрозумілим та легким у сприйнятті; універсальність - Python чудово підходить для створення різноманітних програмних продуктів, від веб-застосунків до наукових обчислень; спільнота - наявність великої та активної спільноти розробників дозволяє отримати підтримку та допомогу швидко та легко; і наявність великої кількості бібліотек та фреймворків, які розширюють можливості Python і полегшують розв'язання різноманітних завдань.

Python знаходить застосування у різних галузях, включаючи розробку веб-застосунків за допомогою фреймворків, таких як Django та Flask, наукові обчислення, де широко використовуються бібліотеки, такі як NumPy та SciPy,

машинне навчання, де Python є основним вибором завдяки бібліотекам, таким як TensorFlow та scikit-learn, розробку ігор, де використовуються бібліотеки, наприклад PyGame, а також для автоматизації завдань завдяки його простоті та гнучкості.

Фреймворком, обраним для розробки мобільного застосунку, є Django. Django - це фреймворк веб-розробки на базі Python, який пропонує широкий спектр функцій для створення веб-застосунків. Django простий у використанні та має велику та активну спільноту, а також чітку та зрозумілу документацію. Django також має багато бібліотек та розширень, які можна використовувати для розробки мобільних застосунків.

Django - безкоштовний інструмент для створення веб-застосунків на Python, який володіє широкими можливостями для створення веб-застосунків. Він славиться своєю простотою використання, масштабованістю та надійністю. Django користується популярністю серед багатьох компаній та організацій, у тому числі Instagram, Pinterest та Mozilla.

Переваги Django:

- Простота: django легко вивчати та використовувати, що робить його прекрасним вибором для розробників з досвідом у Python.
- Масштабованість: django можна застосовувати для створення веб-застосунків будь-якого масштабу, починаючи від невеликих сайтів до великих корпоративних додатків.
- Безпека: у django є вбудовані функції безпеки, які допомагають захистити ваші веб-застосунки від потенційних загроз.
- Гнучкість: django можна налаштувати для створення рознообразних веб-застосунків з різноманітним функціоналом і можливостями.
- Активна спільнота: у django є широка та активна спільнота розробникїv, що означає, що у разї потреби легко можна отримати пїдтримку і пораду.

Деякі з можливостей Django:

- Система управління: у django є вбудована система управління, яка дозволяє легко керувати вмістом вашого веб-сайту.

- Система шаблонів: django має систему шаблонів, яка дає можливість створювати динамічні веб-сторінки без необхідності писати складний HTML-код.
- Підтримка баз даних: django підтримує ряд баз даних, таких як MySQL, PostgreSQL та SQLite.
- Система автентифікації та авторизації: у django існує вбудована система автентифікації та авторизації, яка дозволяє контролювати доступ до вашого веб-застосунку. Ви можете створювати користувачив, групи та надавати їм розличний рівень доступу до застосунку.
- Підтримка REST API: у django є вбудована підтримку REST API для легкої розробки web-API. Це дає можливість іншим програмам спілкуватися з вашим застосунком через HTTP-запити.
- Розширюваність: django можна розширити за допомогою сторонніх бібліотек та пакетів. Це дає можливість добавляти нові функції та можливості до вашого застосунку.

Python та Django представляють собою потужні та універсальні інструменти, що можна використовувати для створення різноманітних мобільних додатків. Вони легкі у використанні, функціональні, мають активну спільноту та зрозумілу документацію. Python та Django також надають можливість розробникам створювати додатки для мобільних пристроїв. з різноманітними функціями та можливостями (рис. 2.1).

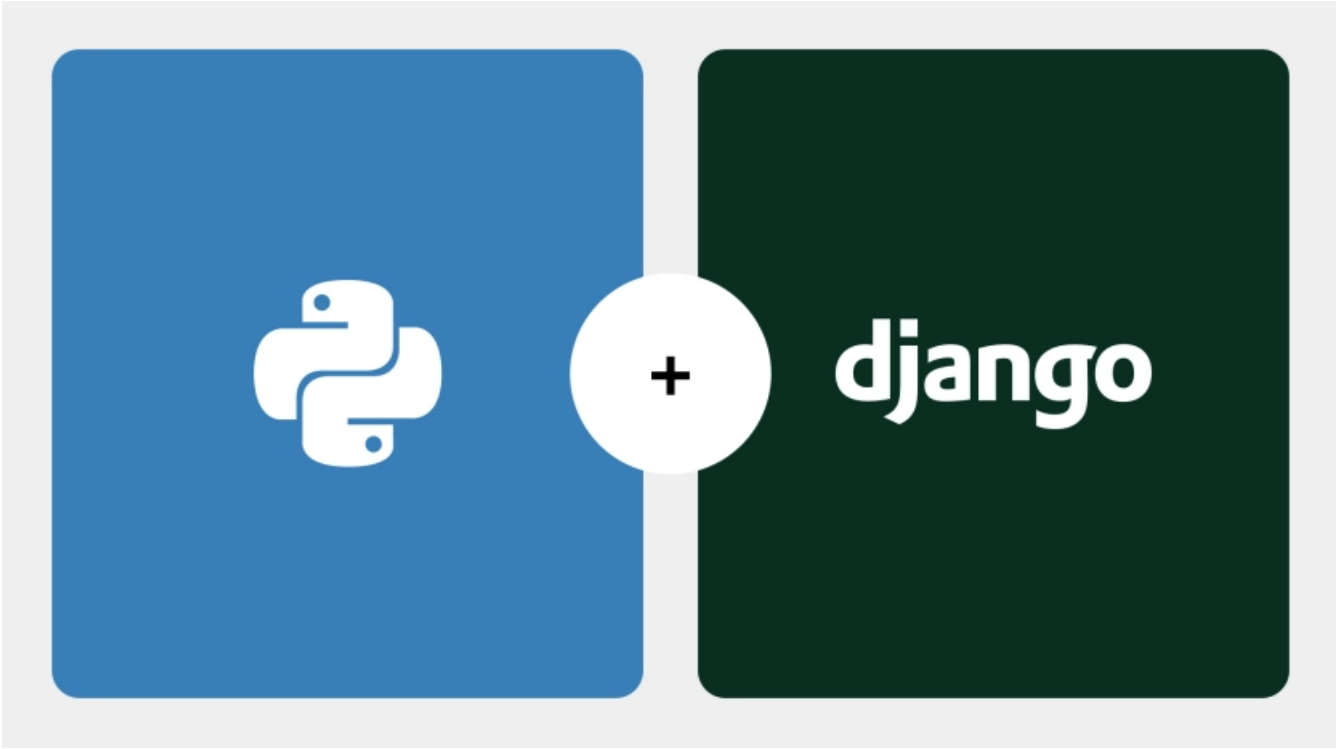


Рис. 2.1 Потужна комбінація для розробки різноманітних додатків

Django REST Framework (DRF) - це ефективний та гнучкий засіб для створення веб-API з використанням Django. Він забезпечує зручні засоби для створення RESTful API, підтримує функції аутентифікації, авторизації, серіалізації, валідації та інші можливості. Завдяки DRF, розробка потужних і гнучких веб-API з використанням Django процес стає швидким і простим.

Django REST Framework (DRF) спрощує процес створення RESTful API, дотримуючись принципів REST, що забезпечує чіткість, послідовність і зручність використання вашого API. DRF надає потужні інструменти для серіалізації даних між Python і форматами, такими як JSON, XML та YAML, що полегшує обмін даними. Він підтримує різні механізми аутентифікації та авторизації, дозволяючи контролювати доступ до вашого API та дії користувачів. Крім того, DRF забезпечує можливості фільтрації та пагінації результатів запитів, роблячи API більш масштабованим та зручним для користувачів. Завдяки чудовій документації, DRF допомагає швидко розпочати роботу та створювати складні API.

Також, Django REST Framework (DRF) забезпечує швидкість розробки завдяки своїм зручним інструментам та функціям, дозволяючи швидко створювати

веб-API. Він також розроблений для масштабованості, що дозволяє обробляти великі обсяги трафіку, та є дуже гнучким інструментом, який можна використовувати для створення різноманітних веб-API. Крім того, DRF має велику та активну спільноту розробників, готових допомогти у разі виникнення проблем.

У цілому, Django REST Framework є відмінним інструментом для Python-розробників, які прагнуть створювати потужні, гнучкі та масштабовані веб-API (рис. 2.2).

Рис. 2.2 Логотип Django REST Framework

SQLite — це легка реляційна система управління базами даних з відкритим вихідним кодом, призначена для зберігання, організації та доступу до даних. Її часто називають вбудованою СУБД, оскільки вона не потребує окремого сервера для роботи. SQLite легко інтегрується у ваш додаток, що робить її ідеальною для мобільних додатків, вбудованих систем та інших проектів, де важливі компактність і простота.

SQLite має декілька ключових характеристик: вона легка, оскільки має дуже маленький розмір файлу, що ідеально підходить для пристроїв з обмеженою пам'яттю; проста у використанні та налаштуванні, не потребує спеціальних знань чи сервера; потужна, підтримуючи функції реляційних баз даних, такі як транзакції, індекси та підзапити; гнучка, оскільки підходить для мобільних, настільних додатків та вбудованих систем; надійна, з репутацією стійкої до збоїв СУБД; і, нарешті, безкоштовна та з відкритим кодом, що дозволяє вільне використання та розповсюдження (рис. 2.3).

Рис. 2.3 Логотип SQLite

Завдяки багатьом перевагам, SQLite стала однією з найбільш популярних систем управління базами даних у всьому світі. Вона використовується в мільйонах додатків, серед яких Firefox, Chrome, WhatsApp, Twitter і Facebook.

Декілька прикладів використання SQLite:

- Зберігання даних користувачів у мобільних додатках.
- Відстеження даних про продажі у веб-додатках.
- Зберігання кешованих даних на локальному пристрої.
- Створення вбудованих систем.

- Прототипування баз даних.

Kivy - це безкоштовна платформа Python з відкритим кодом, яка використовується для розробки різноманітних програм. Вона дозволяє розробляти програми для мобільних пристроїв на платформах Android та iOS, використовуючи код Python, та настільні програми з графічним інтерфейсом користувача для операційних систем Linux, macOS та Windows. Крім того, Kivy підтримує мультитач та інші сенсорні введення, що робить її ідеальним вибором для створення програм для сенсорних екранів та інтерактивних інсталяцій.

Переваги використання Kivy очевидні. По-перше, це швидка розробка завдяки декларативному синтаксису, який дозволяє легко створювати прототипи та розробляти інтерфейси користувача. Крім того, Kivy - це кросплатформовий інструмент, що дозволяє запускати програми на різних платформах без необхідності змінювати код. Важливою перевагою є також те, що Kivy - це безкоштовний та відкритий проект, що робить його доступним для всіх. І нарешті, за Kivy стоїть активна спільнота розробників, яка надає підтримку та розвиває додаткові бібліотеки та інструменти.

Kivy застосовується у різних сферах: від розробки мобільних ігор до створення бізнес-додатків. Він відмінно підходить для створення 2D-ігор для мобільних пристроїв, а також для прототипування та розробки MVP у сфері бізнесу. Освітні програми та інтерактивні навчальні ігри також легко реалізуються за допомогою Kivy. Крім того, цей інструмент можна використовувати для створення захоплюючих мистецьких інсталяцій та візуальних ефектів, розширюючи можливості творчості (рис. 2.4).

Рис. 2.4 Логотип Kivy

2.2 Структура та архітектура мобільного застосунку

Front-end інтерфейс користувача (UI) для нашого проекту розроблений з використанням Kivy, кроссплатформного фреймворку GUI для Python. Цей інтерфейс включає кілька ключових компонентів, які забезпечують зручну навігацію користувача та доступ до усіх необхідних функцій.

Перший компонент - це головний екран, на якому відображається список курсів, доступних для користувачів. Це дозволяє їм швидко знайти потрібний матеріал та переходити до вивчення обраних курсів.

Другий компонент - екран курсу, на якому відображені модулі, уроки, тести та інші навчальні матеріали, пов'язані з обраним курсом. Це забезпечує користувачам повну інформацію про кожен курс і можливість переходу до різних його частин.

Третій компонент - екран уроку, де користувачі можуть переглядати текст уроків, зображення та інші навчальні матеріали, які стосуються конкретного уроку. Це дозволяє їм зосередитися на засвоєнні матеріалу без зайвих відволікань.

Четвертий компонент - екран тесту, де користувачі мають змогу відповідати на питання з множинним вибором, перевіряти свої знання та отримувати пояснення до правильних та неправильних відповідей.

Нарешті, п'ятий компонент - це екран профілю, де відображена інформація про користувача, його прогрес у курсі та інші налаштування. Це дозволяє користувачам відстежувати свій прогрес та налаштовувати інтерфейс під свої потреби.

Усі ці компоненти розроблені з урахуванням зручності для користувача, інтуїтивної зрозумілості та навігаційної логіки, щоб забезпечити максимальний комфорт під час використання системи.

Інтеграція з API back-end є ключовим елементом системи, який забезпечує безперервний обмін даними між front-end і back-end. Ця взаємодія реалізована через REST API, яке надає Django. Цей механізм дозволяє front-end отримувати різноманітну інформацію з back-end, включаючи навчальні матеріали, дані користувачів та результати тестів.

Завдяки інтеграції з API, front-end має зручний доступ до всієї необхідної інформації для ефективного функціонування. Наприклад, користувачі можуть швидко отримувати доступ до навчальних ресурсів, переглядати свої особисті дані та аналізувати результати своїх тестів. Все це забезпечує зручний та ефективний користувацький досвід.

Крім того, front-end має можливість взаємодії з back-end в обох напрямках. Окрім отримання даних, front-end може також передавати інформацію до back-end. Наприклад, користувачі можуть надсилати свої відповіді на тести або змінювати налаштування свого облікового запису. Це забезпечує повний цикл обміну даними між front-end і back-end, що дозволяє системі працювати якісно та ефективно з урахуванням потреб користувачів.

Back-end. Django REST API використовується для створення RESTful API для back-end. Ця система API розробляється з метою надання зручних та ефективних методів взаємодії між front-end та базою даних. Зокрема, API забезпечує доступ до

різноманітних ресурсів, які включають в себе інформацію про курси, навчальні матеріали, користувачів, результати тестів та оцінки.

Для зручності користувачів, API надає ряд кінцевих точок, що дозволяє взаємодіяти з різними аспектами системи. Наприклад, для отримання доступу до інформації про курси та навчальні матеріали передбачені відповідні кінцеві точки. Користувачі можуть реєструватися, автентифікуватися та здійснювати оновлення налаштувань через відповідні кінцеві точки, призначені для управління користувачами.

Додатково, API забезпечує можливість отримання тестів, запису результатів та отримання пояснень до них через відповідні кінцеві точки для тестів. Крім того, інформація про прогрес користувачів у курсі та їхні оцінки доступні через спеціальні кінцеві точки для оцінок.

Для забезпечення безпеки, API використовує механізм JSON Web Tokens (JWT). Це дозволяє здійснювати безпечну автентифікацію та авторизацію користувачів, зменшуючи ризик несанкціонованого доступу до системи та забезпечуючи конфіденційність даних.

База даних SQLite обумовлюється як ключовий компонент нашого застосунку, який служить для зберігання різноманітної інформації. Основні дані, які включені до цієї бази, охоплюють інформацію про курси та навчальні матеріали, необхідні для користувачів. Це включає в себе не лише опис курсів, а й доступ до матеріалів, їх структуру та оновлення.

Далі, база даних SQLite відіграє важливу роль у зберіганні даних користувачів. Це включає їхні особисті дані, такі як імена, паролі та електронні адреси. Крім того, база даних відслідковує прогрес у навчанні, зберігаючи інформацію про те, які курси вони відвідують, і як далеко вони продвинулися у кожному курсі.

Окрім цього, SQLite також використовується для зберігання результатів тестів та оцінок, які користувачі проходять в рамках курсів. Це дозволяє нам ефективно відстежувати успішність кожного користувача і надавати їм відповідні рекомендації та підказки.

Загалом, SQLite обрана як база даних через її легкість та портативність, що робить її ідеальним варіантом для мобільного застосунку. Це забезпечує ефективну та надійну роботу застосунку, дозволяючи зберігати та керувати великою кількістю різноманітних даних.

Система автентифікації та авторизації в нашому проекті реалізована з використанням Django REST framework та JSON Web Tokens (JWT). Django REST framework використовується для побудови цієї системи з метою забезпечення безпеки та ефективності. JWT використовується для генерації та зберігання токенів автентифікації, які надають користувачам доступ до API back-end. Це забезпечує зручність та безпеку процесу автентифікації, дозволяючи користувачам отримати доступ до системи лише за умови правильного введення імен користувачів та паролів.

Після успішної аутентифікації в системі користувачам виданий JWT, який вони зможуть використовувати для доступу до API протягом обмеженого періоду часу. Це дозволяє забезпечити безпеку та контроль над доступом до ресурсів, оскільки токени активні лише на обмежений час та потребують періодичного оновлення. Такий підхід допомагає уникнути можливих загроз безпеці, пов'язаних з несанкціонованим доступом до системи.

В рамках цієї системи користувачі мають можливість реєстрації та входу до системи, використовуючи свої імена користувачів та паролі. Це забезпечує зручність та простоту використання системи для кінцевих користувачів, дозволяючи їм легко використовувати сервіс та отримувати доступ до необхідних функцій і можливостей. Таким чином, за допомогою Django REST framework та JWT ми побудували потужну, безпечну та зручну систему автентифікації та авторизації для нашого проекту.

Система управління навчальним контентом (Learning Management System, LMS) забезпечує організацію, розповсюдження та контроль за навчальними матеріалами і процесами. Готове LMS-рішення передбачає наявність вже розробленого програмного забезпечення, яке може бути інтегроване з back-end на базі Django. Це дозволяє забезпечити ефективне управління курсами та

навчальними матеріалами, а також відстежувати прогрес користувачів. Окрім цього, система надає можливість виставляти оцінки та забезпечувати зворотний зв'язок для користувачів.

Готове LMS-рішення має безліч переваг, зокрема швидкість впровадження та готовий функціонал. Воно ідеально підходить для організацій, яким потрібен стандартний набір можливостей для управління навчальними процесами. Інтеграція з Django дозволяє забезпечити надійність роботи та гнучкість масштабування системи у відповідності до ростучих потреб користувачів.

Однак, у деяких випадках готове LMS-рішення може не відповідати всім специфічним вимогам організації. У такій ситуації може бути доцільним розробити власну систему управління навчальним контентом. Розробка власного LMS потребуватиме додаткового часу та ресурсів для створення унікального функціоналу, який повністю відповідатиме потребам організації. Інтеграція цього рішення з back-end Django дозволить створити гнучку та адаптовану систему, яка забезпечить усі необхідні можливості для ефективного навчання.

Система оцінювання складається з двох основних компонентів: Django REST framework та front-end. Django REST framework відповідає за створення API, яке дозволяє створювати, отримувати та оновлювати оцінки. Це забезпечує гнучку та зручну роботу з даними, які стосуються оцінювання.

На front-end покладено завдання відображення тестів, вікторин, практичних завдань та інших оцінок для користувачів. Завдяки зручному інтерфейсу користувачі можуть легко знаходити та проходити різноманітні оцінювання, що полегшує процес навчання та самоперевірки.

Користувачі мають змогу проходити тести, вікторини та практичні завдання, після чого система записує їхні результати. Це дозволяє зберігати прогрес кожного користувача, надаючи можливість відстежувати свої досягнення та робити висновки щодо необхідності подальшого навчання.

Архітектура застосунку. Клієнт-серверна архітектура є основою побудови даного застосунку. Це означає, що функціональні можливості системи розподіляються між двома основними компонентами: клієнтом і сервером. Клієнт

представляє собою інтерфейс, з яким взаємодіє користувач, тоді як сервер обробляє запити клієнтів, виконуючи необхідні операції та повертаючи результати.

Front-end застосунку працює на мобільних пристроях користувачів. Це означає, що мобільний додаток, встановлений на смартфоні чи планшеті, виконує роль клієнта. Цей мобільний додаток надає користувачам зручний інтерфейс для взаємодії із системою, відправляє запити на сервер та відображає отримані дані. Такий підхід забезпечує зручність і доступність для користувачів, які можуть користуватися застосунком у будь-якому місці і в будь-який час.

Back-end застосунку розміщено на сервері. Це означає, що вся основна логіка обробки даних, а також зберігання та управління базою даних, виконується на серверній стороні. Сервер приймає запити від мобільних клієнтів, обробляє їх, здійснює необхідні обчислення та операції, і відправляє відповіді назад до клієнтів. Така структура дозволяє централізовано управляти системою, що значно спрощує її обслуговування та оновлення.

Використання клієнт-серверної архітектури забезпечує масштабованість та надійність застосунку. Масштабованість означає, що систему можна легко розширювати, збільшуючи кількість серверів або покращуючи їх продуктивність, щоб обслуговувати більшу кількість користувачів. Надійність забезпечується завдяки тому, що сервери можуть використовуватися для реалізації резервних копій даних, а також для забезпечення безперервної роботи системи у випадку збою окремих компонентів. Це дозволяє гарантувати стабільну роботу застосунку навіть за високих навантажень.

Взаємодія між front-end і back-end здійснюється через REST API. Це означає, що front-end буде звертатися до back-end за допомогою HTTP-запитів, щоб отримати або відправити необхідні дані. Відповіді на ці запити будуть містити інформацію у форматі, який front-end зможе легко обробити і відобразити користувачеві.

Використання REST API також дозволить чітко розподілити відповідальність між front-end та back-end. Front-end буде відповідати за представлення даних та взаємодію з користувачем, тоді як back-end буде обробляти запити, виконувати

бізнес-логіку та керувати базою даних. Такий поділ забезпечить легкість в обслуговуванні та розвитку застосунку, а також дозволить різним командам розробників працювати над своїми частинами проекту незалежно одна від одної.

Таким чином, розробка мобільного застосунку з використанням Django як основного фреймворку для back-end та REST API для взаємодії з front-end створить ефективну та гнучку систему. Це дозволить забезпечити високу продуктивність, масштабованість та зручність у використанні як для кінцевих користувачів, так і для розробників.

2.3 Дизайн інтерфейсу користувача

Метою створення інтерфейсу користувача є забезпечення зручного, інтуїтивно зрозумілого та естетично привабливого дизайну, що сприятиме ефективному навчанню користувачів.

При створенні інтерфейсу користувача (ІК) для цього застосунку були враховані кілька ключових вимог, що сприяють досягненню оптимального користувацького досвіду.

Перш за все, ІК має бути простим у використанні. Це означає, що навіть користувачі, які не мають досвіду роботи з мобільними пристроями або не знайомі з курсом з кібербезпеки, повинні мати можливість легко взаємодіяти з застосунком. Простота використання дозволяє зменшити кількість помилок, підвищити ефективність навчання і забезпечити позитивний досвід для всіх користувачів.

Крім того, ІК повинен бути інтуїтивно зрозумілим. Це передбачає наявність чіткої навігації і логічного розташування елементів, щоб користувачі могли швидко знаходити потрібну інформацію і виконувати завдання без необхідності читати детальні інструкції. Інтуїтивність інтерфейсу значно підвищує зручність користування і сприяє швидшому засвоєнню матеріалу курсу.

Привабливість ІК є ще одним важливим аспектом. Візуально привабливий та цікавий дизайн мотивує користувачів продовжувати навчання і підтримує їх інтерес до курсу. Використання яскравих кольорів, привабливої графіки і анімацій

може значно покращити загальне враження від застосунку і зробити процес навчання більш захоплюючим.

Мобільний застосунок для вивчення курсу з кібербезпеки, складається з кількох основних елементів, кожен з яких має значне значення у забезпеченні зручності та ефективності навчання.

Головний екран мобільного застосунку слугує стартовою точкою для користувачів. Він відображає список доступних курсів з кібербезпеки, які можна переглядати та вибирати.

Після вибору курсу користувачі перейдуть на екран курсу. Цей екран відображає детальну інформацію про курс, включаючи його зміст, який структуровано за модулями та уроками. На екрані курсу також доступні вікторини та інші навчальні матеріали, які допоможуть користувачам закріпити отримані знання. Завдяки цьому користувачі матимуть чітке уявлення про структуру курсу та зможуть легко переходити між різними його частинами.

Екран уроку призначений для детального ознайомлення з конкретними темами. Тут відображено зміст уроку, який може включати текстові матеріали, зображення та інші навчальні ресурси. Така багатоформатність дозволить задовольнити різні стилі навчання та зробить процес вивчення матеріалу більш цікавим та ефективним.

Для оцінки знань користувачів після вивчення матеріалу створено екран вікторини. На цьому екрані користувачі можуть відповідати на питання з множинним вибором або виконувати інші типи завдань. Вікторини допоможуть користувачам закріпити знання та оцінити свій рівень засвоєння матеріалу, надаючи негайний зворотній зв'язок.

Останнім ключовим елементом є екран профілю користувача. На цьому екрані користувачі можуть переглядати свій прогрес у навчанні, відстежувати досягнення, змінювати налаштування облікового запису та отримувати доступ до інших персоналізованих функцій. Це допоможе користувачам залишатися мотивованими та організованими під час проходження курсу.

3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Проектування бази даних

Для створення мобільного додатку для вивчення курсу з кібербезпеки була застосована система керування базами даних (СКБД) SQLite. SQLite – це легка, вбудована СКБД, яка не потребує окремого сервера, що робить її ідеальним вибором для мобільних додатків, де важлива портативність і продуктивність. SQLite також проста у використанні та має широкий набір функцій, які підходять для розробки навчальних додатків.

Важливою частиною цього проекту є розробка бази даних, яка містить кілька основних сутностей для забезпечення функціональності застосунку. Першою сутністю є користувачі (users), яка зберігає інформацію про всіх користувачів платформи. Це включає в себе такі дані, як ім'я, адресу електронної пошти, пароль, а також інші релевантні деталі, що допомагають ідентифікувати та керувати обліковими записами користувачів.

Другою важливою сутністю є курси (courses). Ця сутність відповідає за зберігання інформації про доступні на платформі курси. До цієї інформації входять назва курсу, його опис, ім'я викладача та інші деталі, які допомагають користувачам обирати відповідні курси для навчання.

Модулі (modules) є третьою сутністю і служать для організації матеріалів курсу в більш дрібні, керовані частини. Кожен модуль містить назву, опис та порядок відображення, що дозволяє структурувати навчальний матеріал у логічній послідовності.

Четверта сутність, уроки (lessons), зберігає детальну інформацію про конкретні уроки, які входять до складу модулів. Вона включає назву уроку, його опис, текстові матеріали, а також мультимедійні файли, що використовуються для навчання.

Наступною сутністю є вікторини (quizzes), які надають можливість перевірки знань, отриманих під час проходження уроків. Ця сутність містить назву вікторини, її опис, питання, варіанти відповіді та правильні відповіді.

Остання сутність, прогрес (progress), є ключовою для відстеження навчальних досягнень користувачів. Вона зберігає дані про пройдені модулі, уроки, виконані вікторини та бали, отримані на вікторинах. Це дозволяє користувачам бачити свій прогрес і мотивує їх до подальшого навчання.

Зв'язки між основними сутностями застосунку можна представити за допомогою наступної схеми. На вершині ієрархії знаходяться користувачі (users), які можуть бути зареєстровані у багатьох курсах (courses). Кожен курс, у свою чергу, може містити багато модулів (modules). Модулі складаються з уроків (lessons), які забезпечують основний навчальний контент. Уроки можуть містити одну або кілька вікторин (quizzes), які дозволяють користувачам перевіряти свої знання та закріплювати матеріал.

Користувачі мають можливість відстежувати свій прогрес у курсі, що включає прогрес на рівні модулів, уроків та вікторин. Це дозволяє створити індивідуальний підхід до навчання, де кожен користувач може бачити свої досягнення та розуміти, на яких етапах він потребує додаткової уваги. Відстеження прогресу (progress) включає реєстрацію завершених курсів, пройдених модулів, переглянутих уроків та результатів вікторин, що сприяє мотивуванню користувачів до подальшого навчання.

Таблиця 3.1

Користувачі (users)

Поле	Тип даних	Опис
id	INTEGER	Первинний ключ, унікальний ідентифікатор користувача
name	TEXT	Ім'я користувача
email	TEXT	Адреса електронної пошти користувача
password	TEXT	Пароль користувача
created_at	DATETIME	Дата і час створення запису
updated_at	DATETIME	Дата і час останнього оновлення запису

Таблиця 3.2

Курси (courses)

Поле	Тип даних	Опис
id	INTEGER	Первинний ключ, унікальний ідентифікатор курсу
name	TEXT	Назва курсу
description	TEXT	Опис курсу
instructor	TEXT	Ім'я викладача курсу
created_at	DATETIME	Дата і час створення запису
updated_at	DATETIME	Дата і час останнього оновлення запису

Таблиця 3.3

Модулі (modules)

Поле	Тип даних	Опис
course_id	INTEGER	Іноземний ключ, що посилається на id курсу, до якого належить модуль
name	TEXT	Назва модуля
description	TEXT	Опис модуля
order	INTEGER	Порядок відображення модуля в курсі
created_at	DATETIME	Дата і час створення запису
updated_at	DATETIME	Дата і час останнього оновлення запису

Таблиця 3.4

Уроки (lessons)

Поле	Тип даних	Опис
id	INTEGER	Первинний ключ, унікальний ідентифікатор уроку
module_id	INTEGER	Іноземний ключ, що посилається на id модуля, до якого належить урок
name	TEXT	Назва уроку
description	TEXT	Опис уроку
text	TEXT	Текст уроку
media	TEXT	Шлях до мультимедійних файлів, пов'язаних з уроком
created_at	DATETIME	Дата і час створення запису
updated_at	DATETIME	Дата і час останнього оновлення запису

Таблиця 3.5

Вікторини (quizzes)

Поле	Тип даних	Опис
id	INTEGER	Первинний ключ, унікальний ідентифікатор вікторини
lesson_id	INTEGER	Іноземний ключ, що посилається на id уроку, до якого належить вікторина
name	TEXT	Назва вікторини
description	TEXT	Опис вікторини
questions	TEXT	JSON-масив, що містить питання вікторини
answers	TEXT	JSON-масив, що містить варіанти відповідей на питання вікторини
correct_answers	TEXT	JSON-масив, що містить правильні відповіді на питання вікторини
created_at	DATETIME	Дата і час створення запису
updated_at	DATETIME	Дата і час останнього оновлення запису

Таблиця 3.6

Прогрес (progress)

Поле	Тип даних	Опис
id	INTEGER	Первинний ключ, унікальний ідентифікатор запису про прогрес
user_id	INTEGER	Іноземний ключ, що посилається на id користувача
course_id	INTEGER	Іноземний ключ, що посилається на id курсу
module_id	INTEGER	Іноземний ключ, що посилається на id модуля
lesson_id	INTEGER	Іноземний ключ, що посилається на id уроку
quiz_id	INTEGER	Іноземний ключ, що посилається на id вікторини
is_completed	BOOLEAN	Флаг, що вказує, чи пройдено відповідний модуль, урок або вікторину
score	INTEGER	Бал, отриманий на вікторині
created_at	DATETIME	Дата і час створення запису
updated_at	DATETIME	Дата і час останнього оновлення запису

Запити на створення таблиць.

Таблиця users:

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP
);
```

Таблиця courses:

```
CREATE TABLE courses (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT,
    instructor TEXT,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP
);
```

Таблиця modules:

```
CREATE TABLE modules (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    course_id INTEGER NOT NULL,
    name TEXT NOT NULL,
    description TEXT,
    order INTEGER NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

FOREIGN KEY (course_id) REFERENCES courses(id)

);

Таблиця lessons:

CREATE TABLE lessons (

id INTEGER PRIMARY KEY AUTOINCREMENT,

module_id INTEGER NOT NULL,

name TEXT NOT NULL,

description TEXT,

text TEXT,

media TEXT,

created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

FOREIGN KEY (module_id) REFERENCES modules(id)

);

Таблиця quizzes:

CREATE TABLE quizzes (

id INTEGER PRIMARY KEY AUTOINCREMENT,

lesson_id INTEGER NOT NULL,

name TEXT NOT NULL,

description TEXT,

questions JSON NOT NULL,

answers JSON NOT NULL,

correct_answers JSON NOT NULL,

created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

FOREIGN KEY (lesson_id) REFERENCES lessons(id)

);

Таблиця progress:

CREATE TABLE progress (

id INTEGER PRIMARY KEY AUTOINCREMENT,

user_id INTEGER NOT NULL,

course_id INTEGER NOT NULL,

module_id INTEGER,

lesson_id INTEGER,

quiz_id INTEGER,

is_completed BOOLEAN NOT NULL,

score INTEGER,

created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

FOREIGN KEY (user_id) REFERENCES users(id),

FOREIGN KEY (course_id) REFERENCES courses(id)

);

Для створення зазначеної бази даних у Django застосовуємо фреймворк Object-Relational Mapper (ORM). ORM полегшує роботу з базами даних, дозволяючи описувати моделі даних у вигляді об'єктів Python.

Першим етапом є розробка моделей для кожного об'єкта в базі даних. Створюємо файл models.py у каталозі Django-додатку.

from django.db import models

class User(models.Model):

name = models.CharField(max_length=255)

email = models.EmailField(unique=True)

password = models.CharField(max_length=128)

created_at = models.DateTimeField(auto_now_add=True)

updated_at = models.DateTimeField(auto_now=True)

class Course(models.Model):

```
    name = models.CharField(max_length=255)
    description = models.TextField()
    instructor = models.CharField(max_length=255)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
class Module(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    description = models.TextField()
    order = models.IntegerField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
class Lesson(models.Model):
    module = models.ForeignKey(Module, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    description = models.TextField()
    text = models.TextField()
    media = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
class Quiz(models.Model):
    lesson = models.ForeignKey(Lesson, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    description = models.TextField()
    questions = models.JSONField()
    answers = models.JSONField()
    correct_answers = models.JSONField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
class Progress(models.Model):
```

user = models.ForeignKey(User, on_delete=models.CASCADE)

course = models.ForeignKey(Course, on_delete=models.CASCADE)

module = models.ForeignKey(Module, on_delete=models.CASCADE, null=True)

lesson = models.ForeignKey(Lesson, on_delete=models.CASCADE, null=True)

quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE, null=True)

is_completed = models.BooleanField()

score = models.IntegerField(null=True)

created_at = models.DateTimeField(auto_now_add=True)

updated_at = models.DateTimeField(auto_now=True)

Після створення моделей потрібно створити міграції та виконати їх. Для цього використовуємо наступні команди:

python manage.py makemigrations

python manage.py migrate

Після створення міграцій ми використовуємо моделі для взаємодії з базою даних.

3.2 Розробка API та бэкенду

Одним із ключових аспектів є реалізація REST API, яке забезпечує доступ до функціональності та даних бекенду. Це API, побудоване на основі формату JSON, забезпечує взаємодію з мобільним додатком через HTTP-запити.

Нижче наведений опис маршрутів API, методів HTTP та очікуваних даних запиту та відповіді:

Користувачі (users):

- GET /users/: Отримати список усіх користувачів.
- POST /users/: Створити нового користувача.
- GET /users/<id>/: Отримати інформацію про користувача за його ID.
- PUT /users/<id>/: Оновити інформацію про користувача.
- DELETE /users/<id>/: Видалити користувача.

Курси (courses):

- GET /courses/: Отримати список усіх курсів.
- POST /courses/: Створити новий курс.
- GET /courses/<id>/: Отримати інформацію про курс за його ID.
- PUT /courses/<id>/: Оновити інформацію про курс.
- DELETE /courses/<id>/: Видалити курс.

Модулі (modules):

- GET /courses/<course_id>/modules/: Отримати список усіх модулів курсу.
- POST /courses/<course_id>/modules/: Створити новий модуль курсу.
- GET /courses/<course_id>/modules/<id>/: Отримати інформацію про модуль курсу за його ID.
- PUT /courses/<course_id>/modules/<id>/: Оновити інформацію про модуль курсу.
- DELETE /courses/<course_id>/modules/<id>/: Видалити модуль курсу.

Уроки (Lessons)

- GET /courses/<course_id>/modules/<module_id>/lessons/: Отримати список усіх уроків модуля.
- POST /courses/<course_id>/modules/<module_id>/lessons/: Створити новий урок модуля.
- GET /courses/<course_id>/modules/<module_id>/lessons/<id>/: Отримати інформацію про урок за його ID.
- PUT /courses/<course_id>/modules/<module_id>/lessons/<id>/: Оновити інформацію про урок.
- DELETE /courses/<course_id>/modules/<module_id>/lessons/<id>/: Видалити урок.

Вікторини (quizzes):

- GET /courses/<course_id>/modules/<module_id>/lessons/<lesson_id>/quizzes/: Отримати вікторину для уроку.

- POST /courses/<course_id>/modules/<module_id>/lessons/<lesson_id>/quizzes/: Створити нову вікторину для уроку.
- GET /courses/<course_id>/modules/<module_id>/lessons/<lesson_id>/quizzes/<id>/: Отримати інформацію про вікторину за її ID.
- PUT /courses/<course_id>/modules/<module_id>/lessons/<lesson_id>/quizzes/<id>/: Оновити інформацію про вікторину.
- DELETE /courses/<course_id>/modules/<module_id>/lessons/<lesson_id>/quizzes/<id>/: Видалити вікторину.

Прогрес (progress):

- GET /users/<user_id>/progress/: Отримати прогрес користувача у всіх курсах.
- GET /users/<user_id>/progress/<course_id>/: Отримати прогрес користувача у певному курсі.
- GET /users/<user_id>/progress/<course_id>/modules/<module_id>/: Отримати прогрес користувача у певному модулі курсу.
- GET /users/<user_id>/progress/<course_id>/modules/<module_id>/lessons/<lesson_id>/: Отримати прогрес користувача у певному уроці курсу.
- GET /users/<user_id>/progress/<course_id>/modules/<module_id>/lessons/<lesson_id>/quizzes/<quiz_id>/: Отримати прогрес користувача у вікторині.
- POST /users/<user_id>/progress/<course_id>/modules/<module_id>/lessons/<lesson_id>/quizzes/<quiz_id>/submit/: Оновити прогрес користувача у вікторині.

Реалізація API.

1. Встановлюємо Django REST framework:

pip install djangorestframework

2. Додаємо 'rest_framework' до списку встановлених додатків у вашому файлі settings.py:

```
INSTALLED_APPS = [
    ...,
    'rest_framework',
    ...,
]
```

3. Визначаємо схеми серіалізації для наших моделей у файлі serializers.py:

```
from rest_framework import serializers
from .models import User, Course, Module, Lesson, Quiz
class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('id', 'name', 'email', 'password')
class CourseSerializer(serializers.ModelSerializer):
    class Meta:
        model = Course
        fields = ('id', 'name', 'description', 'instructor')
class ModuleSerializer(serializers.ModelSerializer):
    class Meta:
        model = Module
        fields = ('id', 'course', 'name', 'description', 'order')
class LessonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Lesson
        fields = ('id', 'module', 'name', 'description', 'text', 'media')
class QuizSerializer(serializers.ModelSerializer):
```

```
    class Meta:
        model = Quiz
        fields = ('id', 'lesson', 'name', 'description', 'questions', 'answers', 'correct_answers')
```

4. Створюємо представлення API у файлі views.py:

```
from rest_framework import generics
from .models import User, Course, Module, Lesson, Quiz
from .serializers import UserSerializer, CourseSerializer, ModuleSerializer, LessonSerializer, QuizSerializer
class UserList(generics.ListAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
class UserDetail(generics.RetrieveAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
class CourseList(generics.ListAPIView):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer
class CourseDetail(generics.RetrieveAPIView):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer
class ModuleList(generics.ListAPIView):
    queryset = Module.objects.all()
    serializer_class = ModuleSerializer
class ModuleDetail(generics.RetrieveAPIView):
    queryset = Module.objects.all()
    serializer_class = ModuleSerializer
class LessonList(generics.ListAPIView):
    queryset = Lesson.objects.all()
    serializer_class = LessonSerializer
```

```
class LessonDetail(generics.RetrieveAPIView):
    queryset = Lesson.objects.all()
    serializer_class = LessonSerializer
class QuizList(generics.ListAPIView):
    queryset = Quiz.objects.all()
    serializer_class = QuizSerializer
class QuizDetail(generics.RetrieveAPIView):
    queryset = Quiz.objects.all()
    serializer_class = QuizSerializer
```

5. Створюємо URL-адрес для API у файлі urls.py:

```
from django.urls import path, include
from . import views
urlpatterns = [
    path('users/', views.UserList.as_view(), name='user-list'),
    path('users/<int:pk>/', views.UserDetail.as_view(), name='user-detail'),
    path('courses/', views.CourseList.as_view(), name='course-list'),
    path('courses/<int:pk>/', views.CourseDetail.as_view(), name='course-detail'),
    path('courses/<int:course_pk>/modules/', views.ModuleList.as_view(),
name='module-list'),
    path('courses/<int:course_pk>/modules/<int:pk>/',
views.ModuleDetail.as_view(), name='module-detail'),
    path('courses/<int:course_pk>/modules/<int:module_pk>/lessons/',
views.LessonList.as_view(), name='lesson-list'),
    path('courses/<int:course_pk>/modules/<int:module_pk>/lessons/<int:pk>/',
views.LessonDetail.as_view(), name='lesson-detail'),
    path('courses/<int:course_pk>/modules/<int:module_pk>/lessons/<int:lesson_p
k>/quizzes/', views.QuizList.as_view(), name='quiz-list'),
    path('courses/<int:course_pk>/modules/<int:module_pk>/lessons/<int:lesson_p
k>/quizzes/<int:pk>/', views.QuizDetail.as_view(), name='quiz-detail'),
```

path('users/<int:user_pk>/progress/', views.UserProgressList.as_view(), name='user-progress-list'),

path('users/<int:user_pk>/progress/<int:course_pk>/', views.CourseProgressDetail.as_view(), name='course-progress-detail'),

path('users/<int:user_pk>/progress/<int:course_pk>/modules/<int:module_pk>/', views.ModuleProgressDetail.as_view(), name='module-progress-detail'),

path('users/<int:user_pk>/progress/<int:course_pk>/modules/<int:module_pk>/lessons/<int:lesson_pk>/', views.LessonProgressDetail.as_view(), name='lesson-progress-detail'),

path('users/<int:user_pk>/progress/<int:course_pk>/modules/<int:module_pk>/lessons/<int:lesson_pk>/quizzes/<int:quiz_pk>/submit/', views.SubmitQuiz.as_view(), name='submit-quiz'),

]

Розробка бекенду за допомогою Django.

Реєстрації користувачів:

```
from rest_framework import generics, permissions
from .models import User
from .serializers import UserSerializer
class UserRegistration(generics.CreateAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    permission_classes = [permissions.AllowAny]
```

Аутентифікації користувачів:

```
from rest_framework import generics, permissions
from django.contrib.auth import authenticate, login
from .models import User
from .serializers import UserSerializer
class UserLogin(generics.RetrieveAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
```

```
    permission_classes = [permissions.AllowAny]
    def post(self, request, *args, **kwargs):
        email = request.data['email']
        password = request.data['password']
        user = authenticate(request, email=email, password=password)
        if user is not None:
            login(request, user)
            serializer = self.get_serializer(user)
            return Response(serializer.data)
        else:
            return Response({'error': 'Невірні облікові дані'})
```

Оновлення інформації про користувача:

```
from rest_framework import generics, permissions
from .models import User
from .serializers import UserSerializer
class UserUpdate(generics.UpdateAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    permission_classes = [permissions.IsAuthenticated]
    def update(self, request, *args, **kwargs):
        user = self.get_object()
        if user.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права оновлювати цього користувача'})
        serializer = self.get_serializer(user, data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data)
```

Видалення користувача:

```
from rest_framework import generics, permissions
```

```
from .models import User
from .serializers import UserSerializer
class UserDelete(generics.DestroyAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    permission_classes = [permissions.IsAuthenticated]
    def destroy(self, request, *args, **kwargs):
        user = self.get_object()
        if user.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права видаляти цього користувача'})
        user.delete()
        return Response({'message': 'User deleted'})
```

Створення курсів:

```
from rest_framework import generics, permissions
from .models import Course
from .serializers import CourseSerializer
class CourseCreate(generics.CreateAPIView):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer
    permission_classes = [permissions.IsAuthenticated]
```

Логіка оновлення інформації про курс:

```
from rest_framework import generics, permissions
from .models import Course
from .serializers import CourseSerializer
class CourseUpdate(generics.UpdateAPIView):
    queryset = Course.objects.all()
    serializer_class = CourseSerializer
    permission_classes = [permissions.IsAuthenticated]
    def update(self, request, *args, **kwargs):
```

```
        course = self.get_object()
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права оновлювати цей курс'})
        serializer = self.get_serializer(course, data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data)
```

Створення модулів:

```
from rest_framework import generics, permissions
from .models import Course, Module
from .serializers import ModuleSerializer
class ModuleCreate(generics.CreateAPIView):
    queryset = Module.objects.all()
    serializer_class = ModuleSerializer
    permission_classes = [permissions.IsAuthenticated]
    def create(self, request, *args, **kwargs):
        course_pk = request.data['course']
        course = Course.objects.get(pk=course_pk)
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права створювати модуль для цього курсу'})
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save(course=course)
        return Response(serializer.data)
```

Оновлення інформації про модуль:

```
from rest_framework import generics, permissions
from .models import Course, Module
from .serializers import ModuleSerializer
class ModuleUpdate(generics.UpdateAPIView):
```

```
    queryset = Module.objects.all()
    serializer_class = ModuleSerializer
    permission_classes = [permissions.IsAuthenticated]
    def update(self, request, *args, **kwargs):
        module = self.get_object()
        course = module.course
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права на оновлення цього модуля'})
        serializer = self.get_serializer(module, data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data)
```

Видалення модуля:

```
from rest_framework import generics, permissions
from .models import Course, Module
from .serializers import ModuleSerializer
class ModuleDelete(generics.DestroyAPIView):
    queryset = Module.objects.all()
    serializer_class = ModuleSerializer
    permission_classes = [permissions.IsAuthenticated]
    def destroy(self, request, *args, **kwargs):
        module = self.get_object()
        course = module.course
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права видаляти цей модуль'})
        if module.lessons.all().exists():
            return Response({'error': 'Модуль має уроки, не можливо видалити'})
        module.delete()
        return Response({'message': 'Module deleted'})
```

Створення уроків:

```
from rest_framework import generics, permissions
from .models import Course, Module, Lesson
from .serializers import LessonSerializer
class LessonCreate(generics.CreateAPIView):
    queryset = Lesson.objects.all()
    serializer_class = LessonSerializer
    permission_classes = [permissions.IsAuthenticated]
    def create(self, request, *args, **kwargs):
        module_pk = request.data['module']
        module = Module.objects.get(pk=module_pk)
        course = module.course
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права створювати урок для цього модуля'})
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save(module=module)
        return Response(serializer.data)
```

Оновлення інформації про урок:

```
from rest_framework import generics, permissions
from .models import Course, Module, Lesson
from .serializers import LessonSerializer
class LessonUpdate(generics.UpdateAPIView):
    queryset = Lesson.objects.all()
    serializer_class = LessonSerializer
    permission_classes = [permissions.IsAuthenticated]
    def update(self, request, *args, **kwargs):
        lesson = self.get_object()
        module = lesson.module
```

```
        course = module.course
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права оновлювати цей урок'})
        serializer = self.get_serializer(lesson, data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data)
```

Видалення уроку:

```
from rest_framework import generics, permissions
from .models import Course, Module, Lesson
from .serializers import LessonSerializer
class LessonDelete(generics.DestroyAPIView):
    queryset = Lesson.objects.all()
    serializer_class = LessonSerializer
    permission_classes = [permissions.IsAuthenticated]
    def destroy(self, request, *args, **kwargs):
        lesson = self.get_object()
        module = lesson.module
        course = module.course
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права видаляти цей урок'})
        if lesson.quizzes.all().exists():
            return Response({'error': 'Урок має тести, не можливо видалити'})
        lesson.delete()
        return Response({'message': 'Урок видалено'})
```

Створення вікторин:

```
class QuizCreate(generics.CreateAPIView):
    queryset = Quiz.objects.all()
    serializer_class = QuizSerializer
    permission_classes = [permissions.IsAuthenticated]
```

```
    def create(self, request, *args, **kwargs):
        lesson_pk = request.data['lesson']
        lesson = Lesson.objects.get(pk=lesson_pk)
        module = lesson.module
        course = module.course
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права створювати тест для цього уроку'})
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save(lesson=lesson)
        return Response(serializer.data)
```

Оновлення інформації про вікторину:

```
from rest_framework import generics, permissions
from .models import Course, Module, Lesson, Quiz
from .serializers import QuizSerializer
class QuizUpdate(generics.UpdateAPIView):
    queryset = Quiz.objects.all()
    serializer_class = QuizSerializer
    permission_classes = [permissions.IsAuthenticated]
    def update(self, request, *args, **kwargs):
        quiz = self.get_object()
        lesson = quiz.lesson
        module = lesson.module
        course = module.course
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права оновлювати цей тест'})
        serializer = self.get_serializer(quiz, data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
```

```
        return Response(serializer.data)
```

Видалення вікторини:

```
from rest_framework import generics, permissions
from .models import Course, Module, Lesson, Quiz
from .serializers import QuizSerializer
class QuizDelete(generics.DestroyAPIView):
    queryset = Quiz.objects.all()
    serializer_class = QuizSerializer
    permission_classes = [permissions.IsAuthenticated]
    def destroy(self, request, *args, **kwargs):
        quiz = self.get_object()
        lesson = quiz.lesson
        module = lesson.module
        course = module.course
        if course.instructor.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права видаляти цей тест'})
        quiz.delete()
        return Response({'message': 'Quiz deleted'})
```

Оновлення прогресу користувача у вікторині:

```
from rest_framework import generics, permissions
from .models import User, Course, Module, Lesson, Quiz, QuizSubmission
class QuizSubmissionUpdate(generics.UpdateAPIView):
    queryset = QuizSubmission.objects.all()
    serializer_class = QuizSubmissionSerializer
    permission_classes = [permissions.IsAuthenticated]
    def update(self, request, *args, **kwargs):
        submission = self.get_object()
        user = submission.user
        quiz = submission.quiz
        lesson = quiz.lesson
```

```
        module = lesson.module
        course = module.course
        # Перевіряємо, чи є користувач, який робить запит, власником
        # запису про проходження вікторини.
        if user.pk != request.user.pk:
            return Response({'error': 'Ви не маєте права оновлювати цей файл})
        # Оновлюємо дані про проходження вікторини.
        serializer = self.get_serializer(submission, data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        # Розраховуємо та оновлюємо загальний бал користувача за вікторину.
        submission.calculate_score()
        submission.save()
        return Response(serializer.data)
```

Отримання прогресу користувача у вікторині:

```
from rest_framework import generics, permissions
from .models import User, Course, Module, Lesson, Quiz, QuizSubmission
class QuizSubmissionDetail(generics.RetrieveAPIView):
    queryset = QuizSubmission.objects.all()
    serializer_class = QuizSubmissionSerializer
    permission_classes = [permissions.IsAuthenticated]
    def get_object(self):
        user = self.request.user
        quiz_pk = self.kwargs['quiz_pk']
        # Отримуємо запис про проходження вікторини для користувача та вікторини.
        submission = QuizSubmission.objects.get(user=user, quiz=quiz_pk)
        return submission
```

3.3 Створення мобільного інтерфейсу

Для розробки мобільного інтерфейсу нашого застосунку були використані наступні технології. Kivy - це безкоштовний та відкритий фреймворк Python, який призначений для створення кросплатформних мобільних додатків. Однією з основних переваг Kivy є те, що він використовує OpenGL для рендерингу графіки, що робить його високопродуктивним та гнучким. Це дозволяє розробникам створювати інтерактивні та візуально привабливі інтерфейси користувача.

Однією з особливостей Kivy є можливість створення інтерфейсів користувача за допомогою декларативної мови, яка подібна до HTML. Це спрощує процес розробки, оскільки програмісти можуть зручно описувати компоненти інтерфейсу та їхні властивості. Крім того, Kivy підтримує роботу з подіями та динамічну поведінку, що дозволяє створювати складні та інтерактивні застосунки без необхідності писати велику кількість коду.

Додатково до основного фреймворку, було використано Kivy UIX - набір бібліотек Kivy, які містять готові компоненти інтерфейсу користувача. Серед цих компонентів є кнопки, мітки, текстові поля, списки та інші елементи, які широко використовуються в мобільних застосунках. Використання Kivy UIX значно спрощує та прискорює процес розробки інтерфейсу, оскільки розробники можуть використовувати готові рішення замість того, щоб створювати все з нуля.

KivyMD — це зовнішній пакет для Kivy, який забезпечує користувацький інтерфейс стилем матеріального дизайну від Google. Це робить інтерфейс інтуїтивно зрозумілим і візуально привабливим для користувачів.

Таким чином, завдяки Kivy, Kivy UIX та KivyMD створено ефективний та привабливий інтерфейс для нашого мобільного застосунку. Ці технології дозволяють нам зосередитися на функціональності та зручності використання, що є важливими аспектами при розробці освітнього продукту, такого як курс з кібербезпеки.

Створення основного інтерфейсу:

```
import kivy
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.textinput import TextInput
from kivy.uix.gridlayout import GridLayout
from kivy.uix.scrollview import ScrollView
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.image import Image
from kivy.uix.popup import Popup
from kivy.properties import ObjectProperty
from kivy.network.urlrequest import URLError
from kivy.clock import Clock
from kivy.core.window import Window
from kivy.uix.behaviors import FocusableBehavior
from kivymd.app import MDApp
from kivymd.uix.button import MDFlatButton
from kivymd.uix.textfield import MDTextField
from kivymd.uix.card import MDCard
from kivymd.uix.label import MDLabel
from kivymd.uix.dialog import MDDialog
from kivymd.uix.list import MDList, OneLineListItem
# Створення класу для головного екрану
class MainScreen(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        # Створення кнопки "Вхід"
        self.login_btn = MDFlatButton(text="Вхід", on_press=self.login_dialog)
```

```
            # Створення кнопки "Реєстрація"
            self.register_btn = MDFlatButton(text="Реєстрація",
on_press=self.register_dialog)
            # Створення контейнера для кнопок
            button_layout = BoxLayout(orientation='horizontal', padding=20)
            button_layout.add_widget(self.login_btn)
            button_layout.add_widget(self.register_btn)
            # Додавання вмісту на екран
            self.add_widget(button_layout)
        # Метод для відкриття діалогового вікна входу
        def login_dialog(self):
            self.login_popup = LoginPopup()
            self.login_popup.open()
        # Метод для відкриття діалогового вікна реєстрації
        def register_dialog(self):
            self.register_popup = RegisterPopup()
            self.register_popup.open()
    # Створення класу для діалогового вікна входу
    class LoginPopup(MDDialog):
        def __init__(self, **kwargs):
            super().__init__(title="Вхід", **kwargs)
             # Створення поля для введення email
            self.email_field = MDTextField(label="Email", hint_text="Введіть email",
mode="email")
            # Створення поля для введення пароля
            self.password_field = MDTextField(label="Пароль", hint_text="Введіть
пароль", password=True)
            # Створення кнопки "Вхід"
            self.login_btn = MDFlatButton(text="Вхід", on_press=self.login)
            # Створення контейнера для вмісту
```

```
        content = BoxLayout(orientation='vertical', padding=20)
        content.add_widget(self.email_field)
        content.add_widget(self.password_field)
        content.add_widget(self.login_btn)
        # Додавання вмісту до діалогового вікна
        self.add_content(content)
    # Метод для обробки натискання кнопки "Вхід"
    def login(self, instance):
        email = self.email_field.text
        password = self.password_field.text
        # Спробувати увійти за допомогою API Django REST
        # (замість placeholder коду)
        # ...
        # Якщо вхід успішний, закрити діалогове вікно та перейти до
        # екрану зі списком курсів
        if success:
            self.dismiss()
                # Перехід до екрану зі списком курсів
        self.parent_screen.manager.transition.direction = 'left'
        self.parent_screen.manager.current = 'courses_screen'
    # Метод для скидання полів вводу
    def reset_fields(self):
      self.email_field.text = ""
        self.password_field.text = ""
# Створення класу для діалогового вікна реєстрації
class RegisterPopup(MDDialog):
    def __init__(self, **kwargs):
        super().__init__(title="Реєстрація", **kwargs)
        # Створення поля для введення email
```

```
            self.email_field = MDTextField(label="Email", hint_text="Введіть email", mode="email")
            # Створення поля для введення імені
            self.name_field = MDTextField(label="Ім'я", hint_text="Введіть ім'я")
            # Створення поля для введення прізвища
            self.surname_field = MDTextField(label="Прізвище", hint_text="Введіть прізвище")
            # Створення поля для введення пароля
            self.password_field = MDTextField(label="Пароль", hint_text="Введіть пароль", password=True)
            # Створення кнопки "Зареєструватися"
            self.register_btn = MDFlatButton(text="Зареєструватися", on_press=self.register)
            # Створення контейнера для вмісту
            content = BoxLayout(orientation='vertical', padding=20)
            content.add_widget(self.email_field)
            content.add_widget(self.name_field)
            content.add_widget(self.surname_field)
            content.add_widget(self.password_field)
            content.add_widget(self.register_btn)
            # Додавання вмісту до діалогового вікна
            self.add_content(content)
        # Метод для обробки натискання кнопки "Зареєструватися"
        def register(self, instance):
            email = self.email_field.text
          name = self.name_field.text
            surname = self.surname_field.text
            password = self.password_field.text
            # Спробувати зареєструватися за допомогою API Django REST
            # (замість placeholder коду)
```

```
            # ...
            # Якщо реєстрація успішна, закрити діалогове вікно та
            # відобразити повідомлення про успіх
            if success:
                self.dismiss()
                message_popup = MDDialog(title="Успішна реєстрація", text="Ви успішно зареєструвалися!")
                message_popup.open()
        # Метод для скидання полів вводу
        def reset_fields(self):
            self.email_field.text = ""
            self.name_field.text = ""
            self.surname_field.text = ""
            self.password_field.text = ""
    # Створення класу для екрану зі списком курсів
    class CoursesScreen(Screen):
        def __init__(self, **kwargs):
            super().__init__(**kwargs)
            # Створення заголовка екрану
            self.screen_title = MDLabel(text="Доступні курси", font_size=24, halign="center")
            self.add_widget(self.screen_title)
            # Створення списку курсів
            self.course_list = MDList()
            # Завантаження даних про курси з API Django REST
            # (замість placeholder коду)
            # ...
            # Додавання курсів до списку
            for course in courses_data:
                course_item = OneLineListItem(text=course['name'])
```

```
            course_item.bind(on_press=self.open_course)
            self.course_list.add_widget(course_item)
        # Створення скролбару для списку
        self.course_scroll = ScrollView(effect_used="scrollx")
        self.course_scroll.add_widget(self.course_list)
        self.add_widget(self.course_scroll)
    # Метод для відкриття екрану з описом курсу
        def open_course(self, instance):
        course_name = instance.text
        # Завантаження даних про курс з API Django REST
        # (замість placeholder коду)
        # ...
        # Створення екрану з описом курсу
        course_screen = CourseScreen(course_data=course_info)
        # Перехід до екрану з описом курсу
        self.parent_screen.manager.transition.direction = 'left'
        self.parent_screen.manager.current = course_screen
# Створення класу для екрану з описом курсу
class CourseScreen(Screen):
    def __init__(self, course_data, **kwargs):
        super().__init__(**kwargs)
     # Створення заголовка екрану з назвою курсу
        self.screen_title = MDLabel(text=course_data['name'], font_size=24,
halign="center")
        self.add_widget(self.screen_title)
        # Створення опису курсу
        self.course_description = MDLabel(text=course_data['description'],
halign="left", valign="top")
        self.course_description.text_size = self.course_description.width, None
        # Створення контейнера для модулів
```

```
            self.modules_container = MDCard(padding=20)
            # Завантаження даних про модулі курсу з API Django REST
            # (замість placeholder коду)
            # ...
            # Додавання модулів до контейнера
            for module in modules_data:
                module_item = MDLabel(text=module['name'], font_size=18, halign="left")
                self.modules_container.add_widget(module_item)
            # Створення скролбару для контейнера з модулями
            self.modules_scroll = ScrollView(effect_used="scrollx")
            self.modules_scroll.add_widget(self.modules_container)
            self.add_widget(self.modules_scroll)
            # Додавання опису курсу та контейнера з модулями до екрану
            self.add_widget(self.course_description)
            self.add_widget(self.modules_container)
        # Створення головного класу програми
        class MyApp(MDApp):
            def __init__(self, **kwargs):
                super().__init__(**kwargs)
                # Створення менеджера переходів між екранами
                self.screen_manager = ScreenManager()
                # Додавання екранів до менеджера
                self.screen_manager.add_widget(MainScreen(name="main_screen"))
                self.screen_manager.add_widget(CoursesScreen(name="courses_screen"))
                # Створення початкового екрану
                self.screen_manager.current = "main_screen"
            def build(self):
                return self.screen_manager
        # Запуск програми
```

```
if __name__ == "__main__":
    MyApp().run()
```

Main.py:

- Цей файл містить код для головного екрану програми, екранів входу та реєстрації, а також екрану зі списком курсів.
- Використовуються бібліотеки Kivy та KivyMD для створення інтерфейсу користувача.
- Класи MainScreen, LoginPopup, RegisterPopup та CoursesScreen описують логіку та вміст відповідних екранів.
- Заглушки # ... замінюються на код для взаємодії з API Django REST для отримання даних курсів, користувачів та аутентифікації.

MyApp:

- Цей клас є головним класом програми, що успадковується від MDApp KivyMD.
- Він створює менеджер переходів між екранами та додає екрани до менеджера.
- Метод build() повертає менеджер переходів, який використовується як головний виджет програми.

ВИСНОВКИ

У процесі виконання дипломного проєкту було створено мобільний застосунок для вивчення курсу з кібербезпеки мовою Python з використанням фреймворку Django. Застосунок надає зручний інтерфейс для користувачів, що дозволяє їм ефективно взаємодіяти з навчальним контентом, проходити тести та відстежувати свій прогрес.

У першому розділі було проведено аналіз предметної області, що включав дослідження поточного стану розробки мобільних застосунків для освітніх цілей, особливо в галузі кібербезпеки. Було розглянуто існуючі рішення та виявлено основні проблеми та виклики, з якими стикаються розробники та користувачі. Це дозволило визначити ключові вимоги до функціональності та користувацького досвіду майбутнього застосунку.

У другому розділі було розглянуто теоретичні основи розробки мобільного застосунку. Було проаналізовано архітектурні підходи, які забезпечують ефективну роботу та масштабованість системи. Обрано використання фреймворку Django для серверної частини, що забезпечило надійність та безпеку. Крім того, було розглянуто технології, які використовуються для створення мобільного інтерфейсу, включаючи фреймворки для кросплатформної розробки.

У третьому розділі описано процес розробки мобільного застосунку. Детально розглянуто всі етапи створення програмного коду, від проектування бази даних до реалізації функціональних модулів та інтеграції з навчальним контентом. Особливу увагу було приділено реалізації системи тестування знань користувачів та механізмам відстеження прогресу.

В результаті виконаної роботи було створено функціональний мобільний застосунок, який дозволяє користувачам вивчати основи кібербезпеки через інтерактивний контент та проходження тестів. Розроблений застосунок сприятиме підвищенню рівня обізнаності з питань кібербезпеки. Усі поставлені завдання було успішно виконано, що підтверджується функціональністю та зручністю користування створеним продуктом.

Апробація результатів дослідження:

1. Кеда Д.О., Аверічев І.М. Розробка мобільного застосунку для вивчення курсу з кібербезпеки мовою Python з використанням фреймворку Django. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 38-40
2. Кеда Д.О., Аверічев І.М. Переваги використання фреймворку Django для розробки веб-застосунків. IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» - Збірник тез 15.05.2024, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 122-123.

ПЕРЕЛІК ПОСИЛАНЬ

1. Python 3.12.3 documentation URL: https://docs.python.org/uk/3/ (дата звернення 10.03.2024).
2. Welcome to Kivy URL: https://kivy.org/doc/stable/ (дата звернення 11.03.2024).
3. Django documentation URL: https://docs.djangoproject.com/en/5.0/ (дата звернення 12.03.2024).
4. Django REST Framework URL: https://www.django-rest-framework.org/ (дата звернення 12.03.2024).
5. Блинова Н. М., Кирилова О. В., Долженко М. В. Дидактичний потенціал мобільних застосунків для вивчення англійської мови як іноземної. *Вісник університету імені Альфреда Нобеля. Серія: Педагогіка і психологія. Педагогічні науки*. 2023. №1 (25). С. 184– 193. https://pedpsy.duan.edu.ua/images/PDF/2023/1/21.pdf
6. Як захиститися від фішингових атак і повідомляти про них URL: https://support.google.com/websearch/answer/106318?hl=uk (дата звернення 15.03.2024).
7. Викрадення даних URL: https://www.eset.com/ua/support/information/entsiklopediya-ugroz/krazha-dannykh/ (дата звернення 15.03.2024).
8. Що таке DDoS-атака? URL: https://datalabsua.com/ua/what-is-ddos-attack/ (дата звернення 15.03.2024).
9. Створення та використання надійних паролів URL: http://surl.li/tytzc (дата звернення 15.03.2024).
10. Бойко І. Чому важливо оновлювати пристрої та програми на них. *Уніан «Інформаційне агентство»*. 2022. URL: https://www.unian.ua/techno/chomu-vazhlivo-onovlyuvati-pristroji-ta-programi-na-nih-12030279.html
11. Мова програмування Python та її застосування в різних галузях URL: https://foxminded.ua/de-vykorystovuietsia-python/ (дата звернення 20.03.2024).

12. Велика кількість бібліотек, сувора динамічна типізація та проста логіка. Розробники — про переваги та недоліки Python. *Редакція DOU*. 2022. URL: https://dou.ua/lenta/articles/pros-and-cons-of-python/
13. Розробка веб додатків з використанням Python і Django. URL: http://surl.li/trbij (дата звернення 25.03.2024).
14. Кемпбелл С. Підручник Django для початківців: особливості, Архітектура та історія. *Guru99*. 2024. URL: https://www.guru99.com/uk/django-tutorial.html#summary
15. Порівняльний аналіз основних фреймворків: Django, Ruby on Rails, Laravel та Express.js. *UAITLAB*. 2024. URL: http://surl.li/tyurw

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільного застосунку для вивчення курсу з кібербезпеки мовою Python з використанням фреймворку Django

Виконав студент 4 курсу
групи ПД-43
Кеда Денис Олегович
Керівник роботи
К.е.н, доцент кафедри ІПЗ Аверічев Ігор Миколайович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** — поліпшення обізнаності користувачів у галузі кібербезпеки шляхом створення інтерактивного мобільного застосунку.
- **Об’єкт дослідження** — процес навчання курсу з кібербезпеки.
- **Предмет дослідження** — мобільний застосунок для навчання курсу з кібербезпеки.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати існуючі мобільні застосунки для вивчення курсу з кібербезпеки.
2. Розробити функціональні та нефункціональні вимоги до мобільного застосунку для вивчення курсу з кібербезпеки.
3. Спроектувати та реалізувати мобільний застосунок для вивчення курсу з кібербезпеки з використанням фреймворку Django.
4. Провести тестування мобільного застосунку.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	CyberSmart	SANS Institute	Чемпіони з кібербезпеки	CyberSavvy
Інтерактивний навчальний контент	-	+	+	+
Відстеження прогресу	-	-	-	+
Надання теоретичної інформації з кібербезпеки	+	+	-	+
Тести для перевірки знань	-	+	-	+
Підтримка української мови	+	-	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

- Можливість реєструватися та авторизуватися користувачам.
- Можливість переглядати навчальні матеріали курсу з кібербезпеки.
- Можливість проходити вікторини для перевірки знань.
- Можливість пройти підсумковий тест по завершенню курсу.
- Можливість переглядати статистику та прогрес користувача у вивченні курсу з кібербезпеки.

Нефункціональні вимоги:

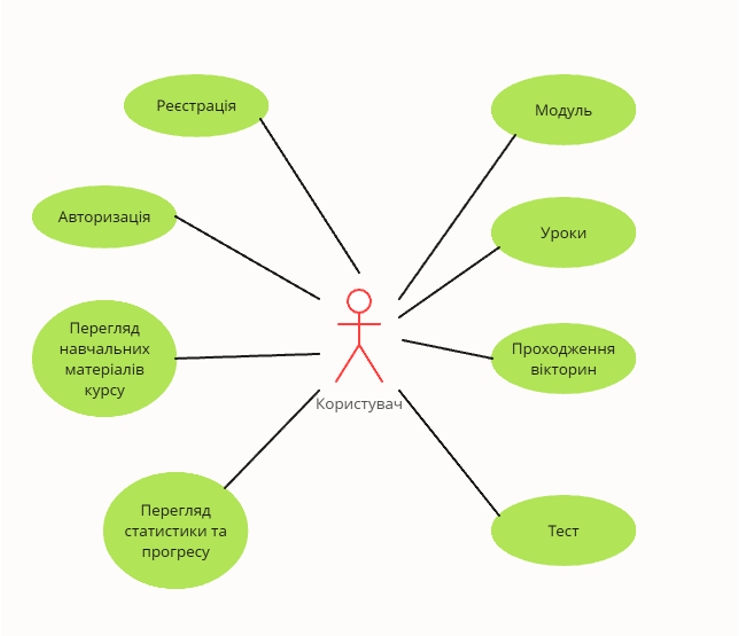
- Українська локалізація інтерфейсу.
- Завантаження теоретичної інформації в форматі TXT.
- Наявність широкого кола питань в тестах з множинним вибором.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

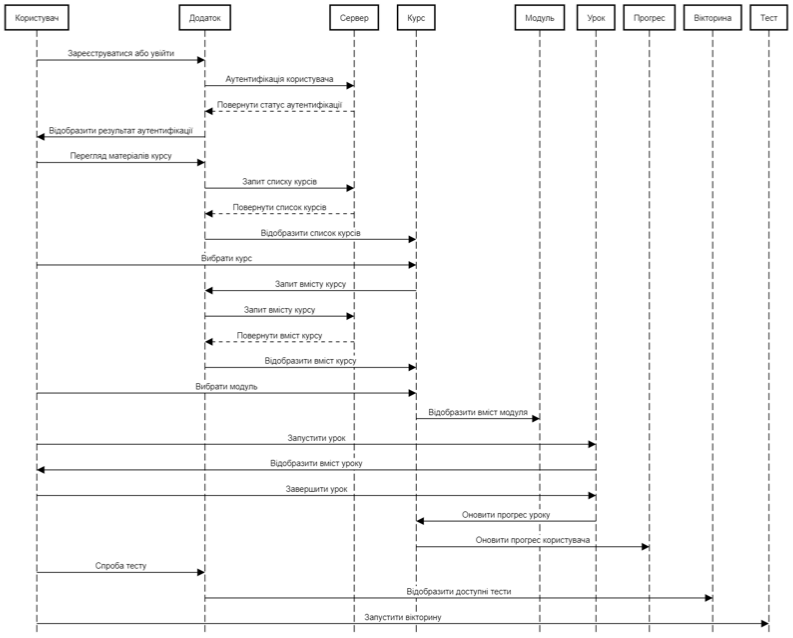
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



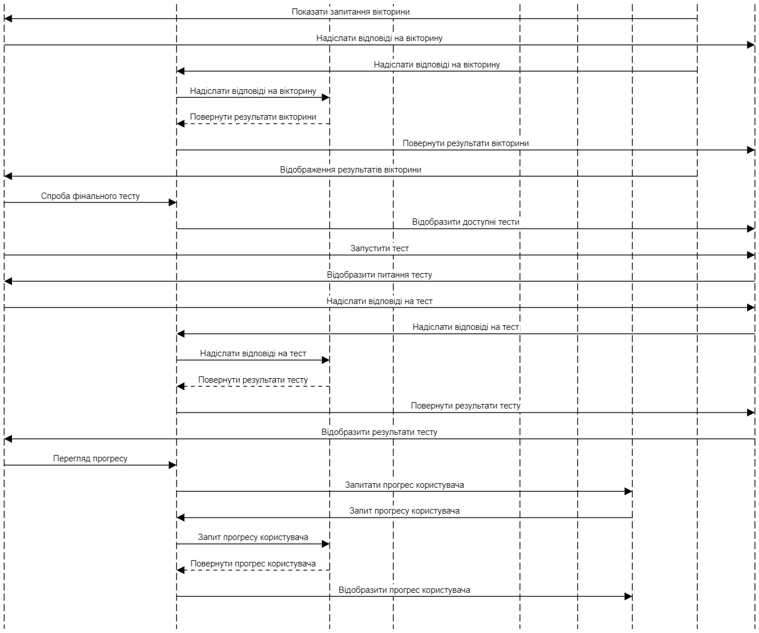
7

ДІАГРАМА ПОСЛІДОВНОСТІ



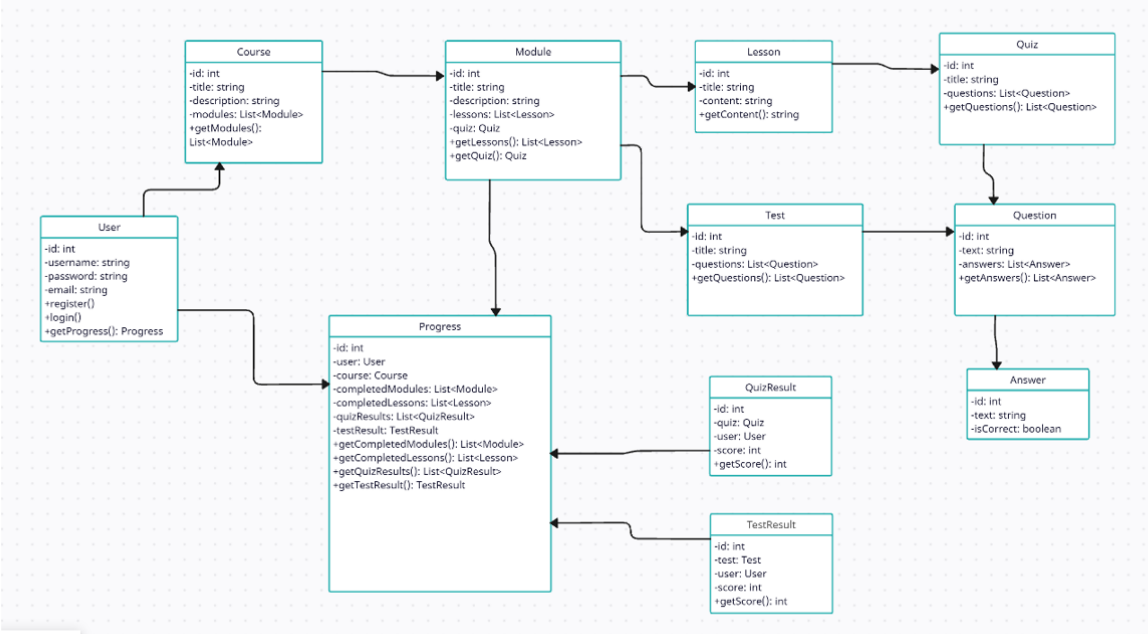
8

ДІАГРАМА ПОСЛІДОВНОСТІ



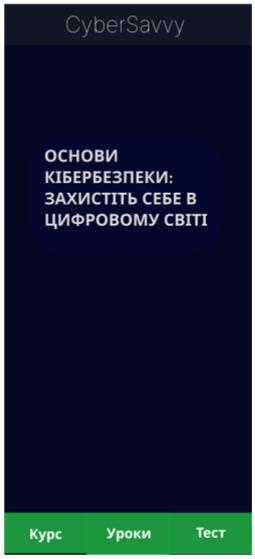
9

ДІАГРАМА КЛАСІВ

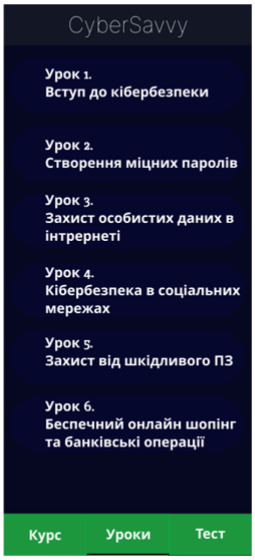


10

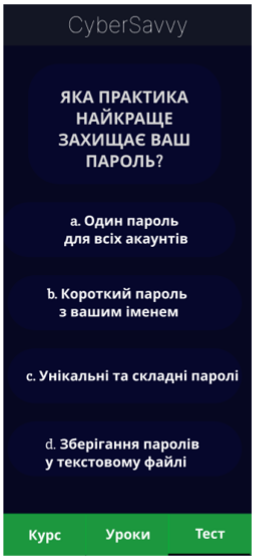
ЕКРАННІ ФОРМИ



Екран курсу



Вибір уроку



Проходження тесту

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Кеда Д.О., Аверічев І.М. Розробка мобільного застосунку для вивчення курсу з кібербезпеки мовою Python з використанням фреймворку Django. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 38-40
2. Кеда Д.О., Аверічев І.М. Переваги використання фреймворку Django для розробки веб-застосунків. IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» - Збірник тез 15.05.2024, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 122-123.

ВИСНОВКИ

1. Проведено аналіз мобільних застосунків для вивчення курсу з кібербезпеки, визначено їх основні функціональні можливості та недоліки.
2. Сформульовано функціональні вимоги до мобільного застосунку, описуючи його можливості та призначення, та визначено нефункціональні вимоги, які стосуються його продуктивності, надійності та інших характеристик.
3. Розроблено мобільний додаток для вивчення курсу з кібербезпеки. Додаток включає інтерактивний навчальний контент у форматі вікторин, підсумковий тест по завершенню курсу, а також функції для відстеження статистики та прогресу користувача. Додаток також містить навчальні модулі та матеріали для самостійного вивчення, що дозволяє адаптувати навчальний процес під індивідуальні потреби кожного користувача.
4. Проведено тестування мобільного застосунку для виявлення та виправлення помилок.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМИ

```
# models.py
from django.db import models

class User(models.Model):
    username = models.CharField(max_length=255)
    password = models.CharField(max_length=255)

class CourseMaterial(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()

class Quiz(models.Model):
    title = models.CharField(max_length=255)
    questions = models.ManyToManyField('Question')

class Question(models.Model):
    text = models.CharField(max_length=255)
    options = models.ManyToManyField('Option')

class Option(models.Model):
    text = models.CharField(max_length=255)
    is_correct = models.BooleanField(default=False)

class Statistic(models.Model):
    user = models.ForeignKey(User,
on_delete=models.CASCADE)
    course_material =
models.ForeignKey(CourseMaterial,
on_delete=models.CASCADE)
    progress = models.IntegerField(default=0)
# views.py
from rest_framework.response import Response
from rest_framework.views import APIView
from.models import User, CourseMaterial, Quiz,
Question, Option
from.serializers import UserSerializer,
CourseMaterialSerializer, QuizSerializer,
QuestionSerializer, OptionSerializer

class UserView(APIView):
    def post(self, request):
        serializer = UserSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'message': 'User created
successfully'})
        return Response(serializer.errors, status=400)

class CourseMaterialView(APIView):
    def get(self, request):
        course_materials = CourseMaterial.objects.all()
        serializer =
CourseMaterialSerializer(course_materials, many=True)
        return Response(serializer.data)

class QuizView(APIView):
    def get(self, request):
        quizzes = Quiz.objects.all()
        serializer = QuizSerializer(quizzes, many=True)
        return Response(serializer.data)

class QuestionView(APIView):
    def get(self, request):
        questions = Question.objects.all()
        serializer = QuestionSerializer(questions,
many=True)
        return Response(serializer.data)

class OptionView(APIView):
    def get(self, request):
        options = Option.objects.all()
        serializer = OptionSerializer(options, many=True)
        return Response(serializer.data)

class StatisticView(APIView):
    def get(self, request):
        statistics = Statistic.objects.all()
        serializer = StatisticSerializer(statistics,
many=True)
        return Response(serializer.data)
# serializers.py
from rest_framework import serializers
from.models import User, CourseMaterial, Quiz,
Question, Option, Statistic

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['username', 'password']

class
CourseMaterialSerializer(serializers.ModelSerializer):
    class Meta:
        model = CourseMaterial
        fields = ['title', 'content']

class QuizSerializer(serializers.ModelSerializer):
    class Meta:
        model = Quiz
        fields = ['title', 'questions']

class QuestionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Question
        fields = ['text', 'options']

class OptionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Option
        fields = ['text', 'is_correct']

class StatisticSerializer(serializers.ModelSerializer):
    class Meta:
        model = Statistic
        fields = ['user', 'course_material', 'progress']
# main.py
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
```

```
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button

class LoginScreen(GridLayout):
    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.cols = 2
        self.add_widget(Label(text='User Name'))
        self.username = TextInput(multiline=False)
        self.add_widget(self.username)
        self.add_widget(Label(text='password'))
        self.password = TextInput(password=True,
multiline=False)
        self.add_widget(self.password)
        self.login_button = Button(text='Login')
        self.add_widget(self.login_button)

class CourseMaterialScreen(GridLayout):
    def __init__(self, **kwargs):
        super(CourseMaterialScreen,
self).__init__(**kwargs)
        self.cols = 1
        self.course_material_label = Label(text='Course
Material')
        self.add_widget(self.course_material_label)
        self.course_material_text =
TextInput(multiline=True)
        self.add_widget(self.course_material_text)

class QuizScreen(GridLayout):
    def __init__(self, **kwargs):
        super(QuizScreen, self).__init__(**kwargs)
        self.cols = 1
        self.quiz_label = Label(text='Quiz')
        self.add_widget(self.quiz_label)
        self.questions = []
```