

Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛІАНСЬКА АКАДЕМІЯ»  
Кафедра інформатики факультет інформатики

## **АВТОМАТИЗАЦІЯ ОЦІНКИ НАВИЧОК ПІД ЧАС МОСК-ІНТЕРВ'Ю В РЕАЛЬНОМУ ЧАСІ**

**Текстова частина до курсової роботи  
за спеціальністю „Інженерія програмного забезпечення” 6.050103**

Керівник курсової роботи

Медвідь С. О.  
Магістр комп'ютерних наук,  
старший викладач

---

(підпис)

“ \_\_\_\_\_ ” \_\_\_\_\_ 2025 р.

Виконав студент 3-го курсу

Шлапак Д.В.

“ \_\_\_\_\_ ” \_\_\_\_\_ 2025 р.

Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультет інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,  
проф., д.ф.-м.н.

\_\_\_\_\_ А. М. Глибовець  
(підпис)

„\_\_\_\_\_” \_\_\_\_\_ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ  
на курсову роботу

студенту 3-го курсу, факультету інформатики  
Шлапаку Данилу Віталійовичу

ТЕМА Автоматизація оцінки навичок під час mock-інтерв'ю в реальному часі

Вихідні дані:

- Технічне завдання
- Документація мови програмування Python
- Документація OpenAI API
- Документація AWS Lambda

Зміст ТЧ до курсової роботи:

Зміст

Анотація

Вступ

1 Аналіз теоретичних засад автоматизації оцінки навичок

2 Методологія дослідження

3 Практична реалізація

4 Аналіз результатів

Висновки

Список літератури

Дата видачі „\_\_\_\_\_” \_\_\_\_\_ 2025 р.

Керівник \_\_\_\_\_  
(підпис)

Завдання отримав \_\_\_\_\_  
(підпис)

## Календарний план виконання курсової роботи

**Тема:** Автоматизація оцінки навичок під час mock-інтерв'ю в реальному часі

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Визначення теми та попередні дослідження	Жовтень 2024	
2.	Огляд літератури та аналіз вимог до системи	Листопад 2024	
3.	Проектування архітектури системи та вибір технологій для реалізації serverless архітектури	Грудень 2024	
4.	Розробка основних компонентів serverless системи. Реалізація основного функціоналу для реєстрації користувачів, управління профілем та проведення інтерв'ю. Інтеграція API Gateway з FrontEnd частиною застосунку	Січень 2025	
5.	Завершення розробки передових функцій системи та початок тестування. Реалізація миттєвого зворотнього зв'язку з GPT, генерація питань та планування інтерв'ю	Лютий 2025	
6.	Оцінка продуктивності системи та завершення написання курсової роботи	Березень 2025	
7.	Подання фінальної курсової роботи та підготовка презентації для захисту	Квітень 2025	
8.	Захист курсової роботи	Травень 2025	

Студент Шлапак Д.В.

Керівник Медвідь С.О.

“ \_\_\_\_\_ ” \_\_\_\_\_ 2025 р.

## ЗМІСТ

АНОТАЦІЯ.....	5
СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ.....	6
ВСТУП.....	7
РОЗДІЛ 1: ТЕОРЕТИЧНІ ЗАСАДИ АВТОМАТИЗАЦІЇ ОЦІНКИ НАВИЧОК ІЗ ЗАСТОСУВАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ .....	9
1.1 Розвиток великих мовних моделей (LLM).....	9
1.2 Застосування великих мовних моделей в освітніх цілях .....	12
1.2.1 Використання великих мовних моделей при підготовці до інтерв'ю .	13
1.2.2 Використання великих мовних моделей при підготовці до інтерв'ю .	14
1.3 Serverless архітектура .....	15
1.4 Дослідницькі прогалини .....	16
РОЗДІЛ 2: МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ ТА ПРОЕКТУВАННЯ СИСТЕМИ.....	18
2.1 Підхід до розробки та методологія дослідження.....	18
2.2 Визначення та аналіз вимог до системи.....	19
2.2.1 Функціональні вимоги.....	19
2.2.2 Нефункціональні вимоги.....	20
2.3 Архітектурне рішення .....	21
2.3.1 Хмарна інфраструктура та сервіси AWS.....	21
2.3.2 Мінімізація Vendor Lock-in .....	22
2.3.3 AWS Lambda: одна функція – один контроллер.....	23
2.3.4 Повторне використання коду за допомогою AWS Lambda Layers .....	25
2.3.5 Архітектура бази даних.....	26
2.4 Обґрунтування вибору технологічного стеку .....	28
2.4.1 Мова програмування Python .....	28
2.4.2 Використані бібліотеки для реалізації BE.....	29
2.4.3 TypeScript для побудови користувацької частини застосунку.....	30
2.4.4 ReactJS.....	31
2.4.5 Інші бібліотеки для реалізації користувацької частини .....	31
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО ПІДХОДУ ....	34
3.1 Реалізація основних компонентів запропонованої системи.....	34
3.1.1 Створення шаблону проекту з використанням AWS SAM .....	34

3.1.2 Реалізація основної логіки з використанням багаторівневої архітектури.....	36
3.2 Інтеграція LLM для генерації та аналізу відповідей .....	38
3.3 Тестування API застосунку.....	41
3.4 Особливості розгортання системи з використанням serverless архітектури .....	42
3.4.1 Локальне розгортання функцій.....	42
3.4.2 Розгортання функцій в середовищі AWS.....	42
РОЗДІЛ 4: АНАЛІЗ РЕЗУЛЬТАТІВ.....	44
4.1 Порівняння результатів із початковими гіпотезами.....	44
4.2 Аналіз переваг і недоліків serverless архітектури .....	45
4.3 Основні результати дослідження.....	46
4.4 Перспективи подальших досліджень .....	47
ВИСНОВКИ .....	49
ВИКОРИСТАНІ ДЖЕРЕЛА.....	50
ДОДАТКИ .....	53

## АНОТАЦІЯ

У курсовій роботі розглянуто проблему автоматизації оцінки технічних навичок кандидатів під час тренувальних співбесід (mock-інтерв'ю) з використанням великих мовних моделей (LLM). Запропоновано архітектурне рішення, яке поєднує можливості моделі GPT-4o для генерації запитань та аналізу відповідей з безсерверною інфраструктурою на базі AWS Lambda. Для реалізації системи використано декларативний підхід до розгортання інфраструктури через AWS SAM. У роботі також проаналізовано переваги й обмеження serverless-архітектури та окреслено перспективи подальших досліджень.

**Ключові слова:** великі мовні моделі, mock-інтерв'ю, оцінка навичок, AWS Lambda, serverless-архітектура, Whisper, OpenAI API.

## **СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ**

LLM – великі мовні моделі

RAG – генерація з доповненням через пошук

FaaS – function as a service

## ВСТУП

У сучасних умовах розвитку інформаційних технологій технічні співбесіди стали ключовим етапом працевлаштування в ІТ, що потребує від кандидатів не лише знань, а й уміння ефективно їх презентувати. Особливо складною підготовка є для новачків, які потребують практики у реалістичних умовах. Найефективнішим методом є тренувальні інтерв'ю (mock-інтерв'ю) з експертами, проте їх кількість є обмеженою.

Одним із рішень цієї проблеми є автоматизація оцінки технічних навичок за допомогою великих мовних моделей (LLM), які не лише генерують персоналізовані технічні запитання, але й виконують аналіз відповідей кандидатів у режимі реального часу.

Реалізація системи автоматизованої оцінки потребує архітектурного рішення, здатного обробляти велику кількість одночасних запитів без втрати продуктивності. У цьому контексті доцільним є впровадження безсерверної архітектури (serverless), яка дозволяє автоматично масштабувати ресурси відповідно до реального навантаження та забезпечує більш просте адміністрування.

Об'єктом дослідження у цій курсовій роботі є процес автоматизації оцінки технічних навичок за допомогою LLM, а предметом – розробка serverless-системи, що поєднує генерацію запитань, транскрипцію відповідей і їх подальший аналіз.

Мета роботи полягає у створенні функціональної системи для автоматизації mock-інтерв'ю з використанням LLM і serverless-архітектури, що дозволить масштабовано, стабільно й ефективно проводити оцінювання кандидатів.

Актуальність обраної тематики визначається зростаючою потребою ринку праці у якісній підготовці молодих спеціалістів до технічних співбесід і зростанням популярності LLM як інструмента для підтримки навчального процесу. Новизна роботи полягає в інтеграції великої мовної моделі з безсервєрною архітектурою для створення гнучкої та масштабованої системи оцінки навичок, що поєднує адаптивне генерування запитань, транскрипцію відповідей і їх детальний аналіз у реальному часі.

Результати роботи можуть бути корисними як для окремих кандидатів, що готуються до співбесід, так і для освітніх закладів або компаній, що шукають шляхи автоматизації процесу попереднього оцінювання технічних знань претендентів. Таким чином, реалізація цього проєкту сприяє підвищенню доступності та якості навчальних сервісів у галузі ІТ.

# **РОЗДІЛ 1: ТЕОРЕТИЧНІ ЗАСАДИ АВТОМАТИЗАЦІЇ ОЦІНКИ НАВИЧОК ІЗ ЗАСТОСУВАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ**

Цей розділ побудований за принципом послідовного викладу матеріалу: від загального огляду до конкретних аспектів дослідження. Такий підхід дозволяє сформулювати загальне уявлення про предмет дослідження та сприяє глибшому розумінню специфічних питань і деталей реалізації.

## **1.1 Розвиток великих мовних моделей (LLM)**

Розвиток великих мовних моделей (LLM) пройшов декілька етапів, кожен з яких суттєво вплинув на сучасний стан технологій обробки природної мови. Значним поштовхом до стрімкого розвитку галузі стала поява трансформерних архітектур у 2017 році, представлених у дослідженні «Attention is All You Need» групою дослідників з Google [1]. Автори цієї роботи представили першу модель перетворення послідовностей, що повністю базувалася на механізмі уваги (attention), замінивши рекурентні шари, які зазвичай використовувалися в архітектурах кодувальника-декодувальника (encoder-decoder). Ця архітектурна зміна дозволила значно прискорити навчання та поліпшити якість генерації тексту, що зробило трансформерні моделі домінуючим рішенням для широкого спектру задач, включаючи переклад, генерацію текстів, класифікацію, а також відповіді на питання [1].

У 2018 році компанія OpenAI представила модель GPT-1 (Generative Pre-trained Transformer), яка вперше використала трансформерну архітектуру для генерації тексту та заклала фундамент для розвитку підходу генеративного попереднього навчання (generative pre-training). Цей метод передбачає початкове тренування моделі на великій кількості текстових даних із подальшим точним налаштуванням (fine-tuning) для конкретних задач.

Важливу роль у ефективному використанні попередньо навчених моделей відіграє саме тонке налаштування (fine-tuning). Цей процес передбачає додаткове навчання моделі на спеціалізованих наборах даних, що дозволяє значно підвищити точність та актуальність генерованого тексту. Завдяки fine-tuning можна зменшити ймовірність виникнення «галюцинацій» (hallucinations)—явища, коли модель створює текст, що не відповідає вихідним даним або контексту. Для запобігання таким випадкам часто застосовується промпт-інжиніринг (prompt engineering)—методика формування запитів до моделі, яка дає змогу точніше контролювати її поведінку та генерувати більш змістовні й правдиві тексти [2] [3].

Того ж року компанія Google розробила модель BERT (Bidirectional Encoder Representations from Transformers), яка продемонструвала принципово новий підхід до попереднього навчання. Особливістю BERT стало використання двонаправленого контексту, що дозволило моделі одночасно враховувати інформацію як ліворуч, так і праворуч від кожного слова у реченні. Цей підхід значно покращив результати у задачах, пов'язаних із глибоким розумінням природної мови, таких як відповіді на питання (question answering), аналіз настрою (sentiment analysis) та класифікація текстів [4].

Наступним значним етапом розвитку великих мовних моделей стала поява моделі GPT-2 у 2019 році, яка суттєво розширила можливості генеративних моделей. GPT-2, що складалася з 1,5 мільярда параметрів, продемонструвала вражаючу здатність до прогнозування наступних елементів послідовностей, завдяки чому вона могла генерувати тексти високої когерентності та змістовності, зберігаючи логічний контекст. Саме ця властивість дозволила GPT-2 ефективно використовуватися в задачах створення контенту, автоматичного доповнення тексту та інтерактивних діалогових системах [5].

У 2020 році компанія OpenAI презентувала ще більш потужну модель— GPT-3, яка мала рекордні на той час 175 мільярдів параметрів. Важливими методологічними нововведеннями GPT-3 стали концепції «few-shot» і «zero-shot» навчання, які дозволяють моделі виконувати задачі з мінімальною кількістю або взагалі без будь-яких прикладів для тренування. Це зробило GPT-3 надзвичайно універсальною у застосуванні та значно зменшило витрати на підготовку спеціалізованих навчальних наборів даних [6]. Паралельно з GPT-3 з'явилися інші значущі моделі, такі як Megatron від Nvidia, Blender від Facebook, T5 та Meena від Google, кожна з яких розширила межі застосування великих мовних моделей у різних спеціалізованих сферах [7], [8].

У 2021 році активний розвиток LLM тривав, характеризуючись появою численних нових моделей, а також інноваційних технік навчання та адаптації. Однією з ключових інновацій стала методика LoRA (Low-Rank Adaptation), яка дала змогу значно знизити обчислювальні ресурси, необхідні для точного налаштування (fine-tuning) великих мовних моделей. Завдяки LoRA, розробники отримали ефективний інструмент для швидкого адаптування великих моделей під специфічні задачі, без потреби повністю перенавчати всі параметри моделі, що суттєво зменшило витрати та прискорило процес розробки [9], [10].

У 2023 році компанія OpenAI представила модель GPT-4— мультимодальну велику мовну модель, що налічує приблизно один трильйон параметрів і таким чином перевершує GPT-3 за обсягом у п'ять разів. GPT-4 здатна ефективно обробляти великі обсяги інформації, наприклад, аналізувати до 50 сторінок тексту за один раз, що відкриває нові перспективи для глибокого семантичного аналізу, інтерактивного навчання та обробки складних текстових і мультимодальних даних [11]. Саме мультимодальність, тобто здатність моделі одночасно сприймати й аналізувати різні типи інформації (текстову, графічну, аудіо та відео), стала однією з ключових особливостей нового покоління LLM, істотно розширивши сферу їхнього практичного застосування [12].

За останні кілька років розвиток великих мовних моделей (LLM) суттєво прискорився, зробивши їх ключовим інструментом для вирішення широкого спектру завдань, пов'язаних з обробкою природної мови, аналізом текстових даних та інтерактивним навчанням. Перші вагомі досягнення таких моделей, зокрема GPT-3 від OpenAI, показали високий потенціал для ефективної генерації зв'язних, змістовних текстів, успішного виконання задач машинного перекладу, анотації, автоматичного узагальнення інформації, а також відповідей на питання з мінімальною кількістю прикладів для навчання ("few-shot learning") [6], [13].

Подальші дослідження в цій сфері зосередилися на подоланні проблеми «галюцинацій», коли моделі створюють неточну або неправдиву інформацію. Одним із ефективних методів для зменшення таких помилок стало застосування підходу Retrieval-Augmented Generation (RAG), який передбачає інтеграцію зовнішніх джерел знань безпосередньо в процес генерації відповідей. Такий підхід забезпечив не лише покращення точності й відповідності відповідей реальним фактам, але й суттєво розширив можливості моделей щодо генерації інформації, релевантної до конкретного контексту [14].

Таким чином, сьогодні великі мовні моделі не лише визначають нові стандарти в галузі обробки природної мови, а й активно трансформують підходи до освіти, комунікації та вирішення широкого спектру практичних завдань.

## **1.2 Застосування великих мовних моделей в освітніх цілях**

Застосування LLM в освітніх цілях відкриває широкі перспективи для вдосконалення якості та ефективності навчального процесу. Завдяки потужним можливостям аналізу природної мови та обробки значних обсягів інформації, LLM здатні забезпечити персоналізацію навчального матеріалу, враховуючи індивідуальні потреби і рівень знань студента.

Однією з ключових переваг LLM є можливість автоматично створювати навчальні ресурси, такі як тексти та тестові завдання на закріплення матеріалу.

Це може значно зменшити навантаження на викладачів, дозволяючи їм приділяти більше уваги безпосередньому навчанню та підтримці студентів. Крім цього, LLM можуть використовуватись для підтримки процесів оцінювання — зокрема, аналізу письмових робіт, тестових відповідей або інших форм навчальної активності, — з метою формування конструктивного зворотного зв'язку та виявлення прогалин у знаннях і для допомоги викладачу.

Проте використання LLM у супроводжується й певними обмеженнями. Особливо відчутними вони є у точних науках, де моделі можуть неправильно виконувати розрахунки або не точно формулювати умови задач.

Незважаючи на всі виклики, ідея інтеграції LLM у навчальне середовище залишається досить перспективною, за рахунок того що LLM здатні створити умови для більш адаптивного, інтерактивного й ефективного навчального процесу.

### **1.2.1 Використання великих мовних моделей при підготовці до інтерв'ю**

Інтеграція LLM в освітню сферу відкриває нові можливості для підвищення ефективності навчального процесу. Однією з головних переваг таких систем є їх здатність автоматизувати рутинні завдання, що традиційно потребували значних часових витрат з боку викладача. LLM можуть генерувати навчальні матеріали, створювати варіанти тестів, перевіряти відповіді на стандартні запитання та формувати базовий зворотний зв'язок. Вони також здатні адаптувати зміст під потреби окремого студента, що є важливим для реалізації персоналізованого підходу до навчання [15].

Проте варто розуміти, що на сучасному етапі розвитку LLM залишаються обмеженими у своїх можливостях і не можуть повноцінно замінити викладача. Найбільш критичними є випадки, коли йдеться про точність формулювань, розв'язання складних задач або застосування міждисциплінарного мислення. Моделі можуть допускати помилки, узагальнювати інформацію без урахування

контексту або навіть генерувати переконливі, але неправдиві відповіді ("галюцинації") [2]. Крім того, LLM не здатні повною мірою враховувати емоційний стан студентів, складність соціальної взаємодії чи нюанси педагогічної етики. У цьому контексті викладач виступає як незамінний супервайзер, що не лише перевіряє точність інформації, а й адаптує навчальний процес до потреб аудиторії.

Таким чином, великі мовні моделі можна розглядати як потужний допоміжний інструмент, а не як повноцінну альтернативу викладачу. Їхнє ефективне застосування можливе у тісній взаємодії з людиною, яка здатна інтерпретувати, оцінювати та коригувати результати їх роботи. Найбільш перспективною є модель «людина + ШІ», де LLM бере на себе автоматизовану частину роботи, а педагог зберігає роль стратегічного керівника і гаранта якості освітнього процесу. Такий підхід дозволяє поєднати переваги технології з професійною інтуїцією та досвідом викладача, що є критично важливим для формування довіри до нових освітніх інструментів [16].

### **1.2.2 Використання великих мовних моделей при підготовці до інтерв'ю**

Особливим напрямом застосування LLM є їх використання у підготовці кандидатів до професійних співбесід. Моделі можуть генерувати реалістичні сценарії, максимально наближені до реальних умов професійних інтерв'ю. Завдяки цьому кандидати мають змогу отримати практичний досвід відповідей на типові питання, які можуть виникнути під час справжніх співбесід, що дозволяє їм бути краще підготовленими до реальних умов [17], [18].

Ключовою перевагою LLM є здатність адаптувати запитання до конкретних галузей знань і спеціалізацій кандидатів. Це забезпечує персоналізований підхід, який враховує індивідуальний досвід, професійні навички та рівень кваліфікації кожного користувача. Моделі можуть аналізувати

відповіді кандидатів у реальному часі, що дозволяє оперативно виявляти слабкі місця, які потребують подальшого вдосконалення [17].

LLM також створюють умови для надання миттєвого конструктивного зворотного зв'язку, який допомагає кандидатам не лише зрозуміти свої помилки, але й швидко їх виправити. Крім того, застосування LLM сприяє розвитку професійних комунікаційних навичок, допомагаючи кандидатам краще формулювати свої думки, ефективніше відповідати на складні запитання та покращувати свою загальну самопрезентацію [19].

### 1.3 Serverless архітектура

Serverless архітектура — це сучасна парадигма побудови програмних систем, яка дозволяє розробникам фокусуватися виключно на бізнес-логіці, не переймаючись управлінням серверною інфраструктурою. Основна ідея полягає в тому, що обчислювальні ресурси надаються хмарним провайдером динамічно, «on demand», а всі питання масштабування, доступності, оновлень і моніторингу автоматично делегуються. В основі цієї моделі лежить концепція функцій як сервісу (Function-as-a-Service, FaaS), де кожна функція виконується у відповідь на конкретну подію. Найпопулярнішими платформами є AWS Lambda, Google Cloud Functions і Azure Functions [20].

Вибір serverless-підходу особливо виправданий у сценаріях, де навантаження на систему є нерівномірним або складним для прогнозування. У таких випадках автоматичне масштабування стає критично важливим. Наприклад, у додатку для проведення mock-інтерв'ю навантаження може значно зрости в години пік. Завдяки serverless-архітектурі інфраструктура автоматично адаптується до змін обсягу запитів — система масштабується вгору при високому навантаженні й назад униз, коли навантаження спадає. Це дозволяє уникнути простою або перевитрати ресурсів і підтримувати стабільну роботу навіть за пікових умов [20].

Ще одним важливим фактором є модель оплати «pay-as-you-go» — оплата відбувається лише за час фактичного виконання функцій. Це не лише знижує операційні витрати на старті, а й робить архітектуру більш економічно обґрунтованою у довгостроковій перспективі. Також serverless дозволяє пришвидшити ітерації розробки: кожен функцію можна оновлювати незалежно, без перезапуску всієї системи. Це особливо зручно в умовах швидких змін вимог або під час експериментального впровадження нових функцій. Serverless добре інтегрується з мікросервісним підходом — кожна функція реалізує окрему одиницю логіки, що спрощує масштабування, тестування та підтримку коду [21].

Однак поряд з перевагами serverless має і певні обмеження. Найпоширеніша проблема — cold start: початкова затримка, яка виникає під час запуску функції після певного періоду простою. Це може бути критичним для застосунків, що вимагають миттєвої реакції в реальному часі. Крім того, використання специфічних інструментів хмарного провайдера може призвести до такого явища, як vendor lock-in — залежності від конкретної екосистеми, що ускладнює міграцію до іншого постачальника послуг або повторне використання коду [22].

#### **1.4 Дослідницькі прогалини**

Попри значний прогрес у розвитку великих мовних моделей (LLM) та serverless-архітектур, на перетині цих технологій залишаються недостатньо вивчені напрямки, які є ключовими для побудови освітніх рішень. В межах даної курсової роботи ідентифіковано такі дослідницькі прогалини:

##### **1. Динамічна адаптація запитань під час тоск-інтерв'ю**

Бракує рішень, що дозволяють LLM в реальному часі змінювати рівень складності запитань на основі відповідей кандидата.

##### **2. Формування персоналізованих профілів навичок**

Сучасні підходи не забезпечують комплексної оцінки soft- і hard-skills та не формують узагальнений портрет кандидата після інтерв'ю.

### 3. Функціонал для HR: пошук і ранжування кандидатів

Відсутні інструменти, які б дозволяли HR-фахівцям здійснювати ефективний відбір кандидатів на основі автоматизованого аналізу навичок.

### 4. Реалізація системи на базі serverless-архітектури

Недостатньо досліджень щодо застосування безсерверних технологій для обробки mock-інтерв'ю з мінімальними затримками, масштабуванням у години пікових навантажень та забезпеченням безперервного аналізу відповідей.

Таким чином, дане дослідження спрямоване на розв'язання зазначених проблем шляхом побудови гнучкої, адаптивної та масштабованої системи автоматизованої оцінки навичок, яка поєднує можливості LLM з перевагами serverless-архітектури.

## **РОЗДІЛ 2: МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ ТА ПРОЕКТУВАННЯ СИСТЕМИ**

У цьому розділі розглядається підхід до побудови системи автоматизованої оцінки навичок на основі великих мовних моделей і serverless-архітектури. Описується обрана методологія розробки, формулюються функціональні й нефункціональні вимоги до системи, архітектура системи, а також обґрунтовується вибір технологічного стеку.

Комплексне дослідження потреб користувачів, бізнес-вимог і технічних обмежень на ранніх етапах створює передумови для реалізації стабільної, масштабованої та ефективної системи [23].

### **2.1 Підхід до розробки та методологія дослідження**

У сучасній IT-індустрії широке поширення отримали гнучкі підходи до розробки програмного забезпечення, зокрема методології Scrum та Kanban. Їх головною перевагою є здатність адаптуватися до змін вимог і зовнішніх умов, що особливо важливо для продуктів зі змінною або недостатньо чіткою специфікацією [23]. Ітеративний характер Agile-підходів, регулярний зворотний зв'язок зі стейкхолдерами та гнучке планування сприяють поступовому вдосконаленню продукту й мінімізації ризиків на кожному з етапів життєвого циклу [24].

Водночас для реалізації системи в рамках даної курсової роботи використання повноцінної Agile-методології є недоцільним, оскільки вимоги до системи, її цілі та обсяг функціональності були визначені заздалегідь. У таких випадках ефективнішим підходом є використання каскадної моделі (Waterfall), яка передбачає послідовне проходження етапів: аналіз вимог, проектування, реалізація, тестування та впровадження [25]. Такий підхід забезпечує передбачуваність, чіткість у плануванні та мінімізує потребу у гнучких механізмах адаптації.

Крім того, модель Waterfall орієнтована на ретельне документування кожного етапу, що створює додаткову аналітичну базу для подальшої оцінки якості реалізованого рішення [25], [26].

## **2.2 Визначення та аналіз вимог до системи**

### **2.2.1 Функціональні вимоги**

Функціональні вимоги визначають основні операції та взаємодії, які система повинна підтримувати для забезпечення ефективного функціонування і задоволення потреб користувачів [26].

На основі аналізу сучасних платформ для проведення технічних інтерв'ю, а також з урахуванням особливостей процесу оцінки навичок під час mock-інтерв'ю, було виокремлено такі функціональні вимоги до нашої системи:

#### **Реєстрація та управління профілем**

Система повинна надавати користувачу можливість створити обліковий запис, увійти до системи та редагувати персональний профіль. Профіль має містити базову інформацію про користувача, а також дані щодо освіти, професійного досвіду та навичок.

#### **Управління навичками**

Користувач повинен мати змогу додавати, редагувати та видаляти технічні навички. Зібрана інформація використовується як основа для генерації персоналізованих запитань під час інтерв'ю.

#### **Проведення інтерв'ю в реальному часі**

Система повинна підтримувати можливість проведення mock-інтерв'ю з автоматичною генерацією запитань відповідно до профілю користувача. Відповіді можуть надаватись у текстовій або голосовій формі та оцінюються вручну або автоматизовано.

#### **Оновлення запитань та зворотній зв'язок у режимі реального часу**

На основі відповідей користувача система повинна автоматично адаптувати наступні запитання, забезпечуючи поступове ускладнення або спрощення діалогу. Миттєвий зворотний зв'язок допомагає користувачу відразу отримати інформацію про свої помилки чи сильні сторони.

### **2.2.2 Нефункціональні вимоги**

Нефункціональні вимоги визначають якісні характеристики системи, що забезпечують її надійність, зручність, стабільність і безпеку в роботі [26]. На відміну від функціональних вимог, вони не описують конкретні дії, а формують загальні очікування до поведінки програмного забезпечення в реальних умовах експлуатації.

#### **Безпека**

Система повинна гарантувати захист персональних даних користувачів шляхом застосування сучасних засобів шифрування, безпечної автентифікації та авторизації. Усі дані мають оброблятися й передаватися згідно зі стандартами інформаційної безпеки, що мінімізує ризик витоку або несанкціонованого доступу.

#### **Швидкість зворотного зв'язку**

Усі операції, пов'язані з генерацією запитань та наданням фідбеку, повинні виконуватись з мінімальними затримками. Це критично для забезпечення реалістичності тоск-інтерв'ю та формування позитивного користувацького досвіду.

#### **Продуктивність**

Система має демонструвати стабільну роботу навіть за умов високого навантаження. Ключові функції повинні виконуватись без збоїв і з допустимим часом відгуку незалежно від кількості одночасних користувачів.

## **Масштабованість**

Архітектура системи повинна підтримувати горизонтальне масштабування, що дозволяє автоматично адаптувати ресурси до зростаючого навантаження. Це забезпечить стабільність роботи під час пікової активності та дозволить ефективно обслуговувати більшу кількість користувачів без зниження продуктивності.

## **2.3 Архітектурне рішення**

У цьому підрозділі демонструється загальна логіка побудови системи, її взаємозв'язки між ними, а також технологічні засоби реалізації кожного з функціональних блоків.

### **2.3.1 Хмарна інфраструктура та сервіси AWS**

Одним із ключових рішень у проектуванні serverless-архітектури є вибір хмарного провайдера, здатного забезпечити стабільну, масштабовану й безпечну інфраструктуру для обробки подій у режимі реального часу. Серед основних критеріїв вибору слід виділити наявність керованих сервісів, конкурентну цінову модель, високу доступність, а також підтримку сучасних стандартів безпеки. З огляду на ці чинники, в межах реалізації даного проєкту було обрано платформу Amazon Web Services (AWS).

AWS є одним із лідерів ринку хмарних обчислень і стабільно займає провідні позиції у звітах аналітичних компаній [27]. Завдяки розгалуженій екосистемі сервісів, високій надійності та глобальній присутності, платформа широко використовується для розробки безсерверних рішень у середовищах зі змінним навантаженням. Додатковою перевагою є доступ до Free Tier — безкоштовного пробного періоду, який надає до 1 мільйона викликів AWS Lambda щомісяця, що дозволяє значно знизити витрати на етапі прототипування та початкової розробки.

Ключовим обчислювальним компонентом архітектури є AWS Lambda — сервіс FaaS, який автоматично масштабується залежно від кількості вхідних запитів. Його подієво-орієнтована модель дозволяє реалізувати ефективну реакцію на події без необхідності управління серверами. Lambda інтегрується з іншими сервісами AWS, зокрема з Amazon DynamoDB, що використовується як основне сховище для збереження даних користувачів, інтерв'ю та результатів оцінювання, та з API Gateway, який виконує функції маршрутизації HTTP-запитів.

Amazon DynamoDB обрано як базу даних завдяки її високій продуктивності, низькій затримці та здатності автоматично масштабуватися у відповідь на зміни навантаження. Вона добре підходить для роботи із запитами в реальному часі та ефективного зберігання напівструктурованих об'єктів, таких як профілі навичок або відповіді на запитання.

Взаємодія між клієнтською та серверною частинами реалізується за допомогою AWS API Gateway, що забезпечує створення, захист і централізоване управління RESTful-інтерфейсами. API Gateway підтримує автентифікацію, контроль навантаження, логування запитів, а також прозору інтеграцію з Lambda-функціями, що дозволяє побудувати масштабовану й безпечну серверну частину без зайвих витрат на інфраструктуру.

### **2.3.2 Мінімізація Vendor Lock-in**

Одним із найбільш критичних викликів при побудові систем на основі хмарних технологій є Vendor Lock-in — залежність від специфічної інфраструктури або сервісів одного постачальника. Така залежність обмежує гнучкість системи, ускладнює її масштабування, обслуговування та міграцію у разі зміни технологічного середовища [28].

Сучасні провайдери хмарних обчислень — Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP) — пропонують власні реалізації

безсерверних функцій: AWS Lambda, Azure Functions і Google Cloud Functions відповідно. Хоча ці сервіси виконують схожі функції, кожен із них вимагає використання фреймворків, SDK, API або конфігурацій, специфічних для конкретної платформи [29]. Це ускладнює міграцію між провайдерами: зміна середовища може потребувати значного рефакторингу, оновлення інфраструктурного коду та перегляду архітектурних рішень [29].

Щоб мінімізувати рівень залежності ще на етапі проєктування, доцільно застосовувати принципи ізоляції бізнес-логіки від платформено-залежних компонентів. Одним із найбільш ефективних інженерних підходів у цьому контексті є реалізація структурного патерну “Фасад” (Facade) [30]. Цей патерн передбачає створення абстрактного шару взаємодії, який інкапсулює всі інфраструктурні виклики за спільним інтерфейсом. У разі необхідності перенесення на іншу платформу достатньо реалізувати альтернативний фасад, не змінюючи основну логіку роботи системи.

### **2.3.3 AWS Lambda: одна функція – один контроллер**

Одним із ключових архітектурних рішень у проєкті стало впровадження підходу, за яким кожен REST-контролер реалізується у вигляді окремої Lambda-функції. Така модель дозволяє досягти високого рівня модульності, керованості та стабільності системи, що є критичним у контексті serverless-архітектури.

Насамперед, дана стратегія забезпечує чітке розділення відповідальностей між компонентами. Кожна функція відповідає за окрему частину бізнес-логіки, що суттєво спрощує процес розробки, тестування та відлагодження. У разі виникнення помилки її легко локалізувати на рівні конкретного контролера, не впливаючи на інші частини системи.

Крім того, ізольованість виконання Lambda-функцій підвищує загальну надійність і безпеку: збої в одній функції не поширюються на інші, а

налаштування середовища, ролі доступу та параметри масштабування можуть бути задані індивідуально для кожної одиниці. Це дозволяє адаптувати виконання під конкретні потреби кожного сервісу без ризику порушення загальної роботи системи.

Також перевагою є незалежність у розгортанні: кожну Lambda-функцію можна оновлювати окремо, що прискорює процес релізів, зменшує час простою та мінімізує ризики, пов'язані з глобальними змінами інфраструктури. Такий підхід ідеально поєднується з використанням AWS API Gateway, який виконує маршрутизацію HTTP-запитів до відповідних функцій. У результаті формується прозора, масштабована та легко підтримувана архітектура REST API.

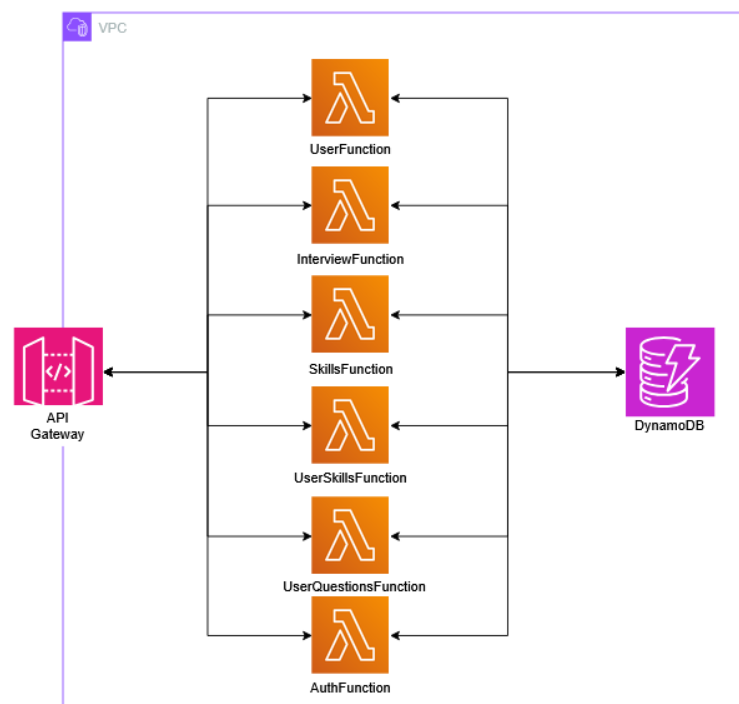


Рисунок 2.1 – Схема використання AWS Lambda

У порівнянні з альтернативними моделями цей підхід є найбільш збалансованим. Варіант, за якого кожен endpoint реалізується окремою функцією, призводить до фрагментації логіки, надлишкового дублювання коду й складнощів із повторним використанням спільних компонентів. З іншого боку,

реалізація всієї логіки в межах однієї функції суперечить принципу єдиної відповідальності (SRP), ускладнює масштабування та створює додаткові труднощі під час оновлення або моніторингу — навіть незначні зміни вимагають повторного тестування всієї функції.

### 2.3.4 Повторне використання коду за допомогою AWS Lambda Layers

Для забезпечення модульності, зменшення дублювання коду та спрощення супроводу системи в архітектурі реалізовано використання Lambda Layers — механізму, що дозволяє винести спільні компоненти в окремі шари, доступні для повторного використання у кількох Lambda-функціях [31].

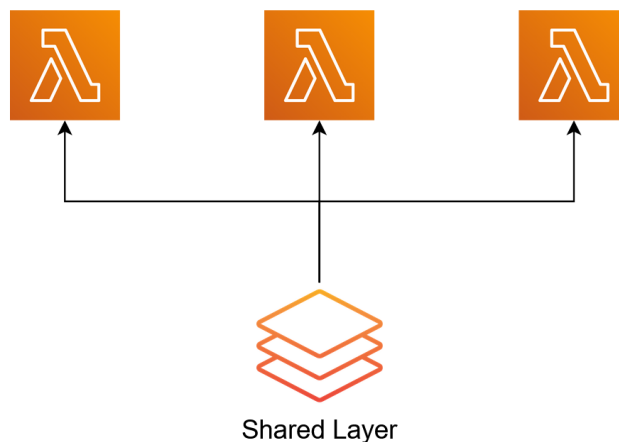


Рисунок 2.2 – Загальна схема роботи AWS Layer

Lambda Layers слугують середовищем для розміщення загального коду, який не повинен змінюватися в межах окремої бізнес-логіки. Зокрема, до таких шарів входять:

- утиліти для валідації даних
- обробники помилок
- спільні структури DTO (Data Transfer Objects)
- обгортки для викликів зовнішніх API

- базові сервіси доступу до сховищ або конфігурацій

Завдяки такій структурі значно зменшується дублювання коду між функціями, що, своєю чергою, покращує читабельність і спрощує підтримку. Оновлення загального шару виконується централізовано, без потреби вносити зміни до кожної функції окремо. Це знижує ризик помилок, пов'язаних із несинхронним оновленням однакових фрагментів логіки [31].

Крім того, повторне використання спільних компонентів оптимізує процес розгортання нових функцій: при створенні нових контролерів або сервісів можна одразу підключити існуючий Layer, не витрачаючи час на копіювання та перевірку типових блоків. Такий підхід покращує структуру проєкту, сприяє стандартизації логіки, а також дозволяє зосередитись на розробці бізнес-функціоналу, мінімізуючи технічний борг [31].

#### ▼ Function overview [Info](#)

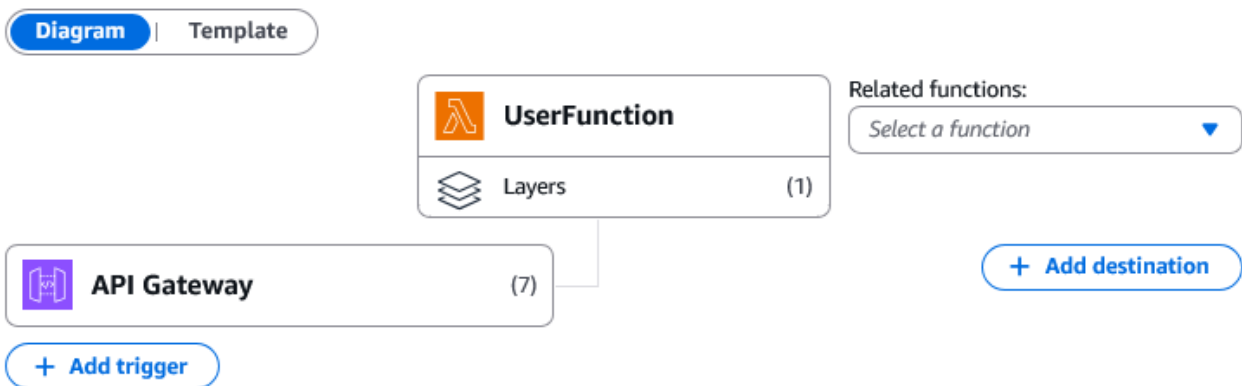


Рисунок 2.3 – Діаграма Lambda функції на AWS

### 2.3.5 Архітектура бази даних

Для збереження даних у системі було обрано NoSQL-рішення Amazon DynamoDB. Такий вибір зумовлений вимогами до високої масштабованості, мінімальних затримок при обробці запитів у реальному часі та можливістю

нативної інтеграції з іншими сервісами AWS в рамках побудови serverless-архітектури.

DynamoDB є однією з найбільш популярних баз даних для serverless-систем завдяки своїй архітектурі, яка забезпечує автоматичне масштабування, високу доступність і низьку затримку навіть за умов інтенсивного паралельного доступу [32]. У поєднанні з AWS Lambda, API Gateway та іншими хмарними сервісами, DynamoDB дозволяє будувати повністю безсерверні рішення без необхідності вручну керувати ресурсами.

В контексті даної системи DynamoDB використовується для зберігання основних сутностей:

- користувачів і їхніх профілів
- навичок, пов'язаних із користувачами
- сценаріїв інтерв'ю
- історії відповідей та оцінок

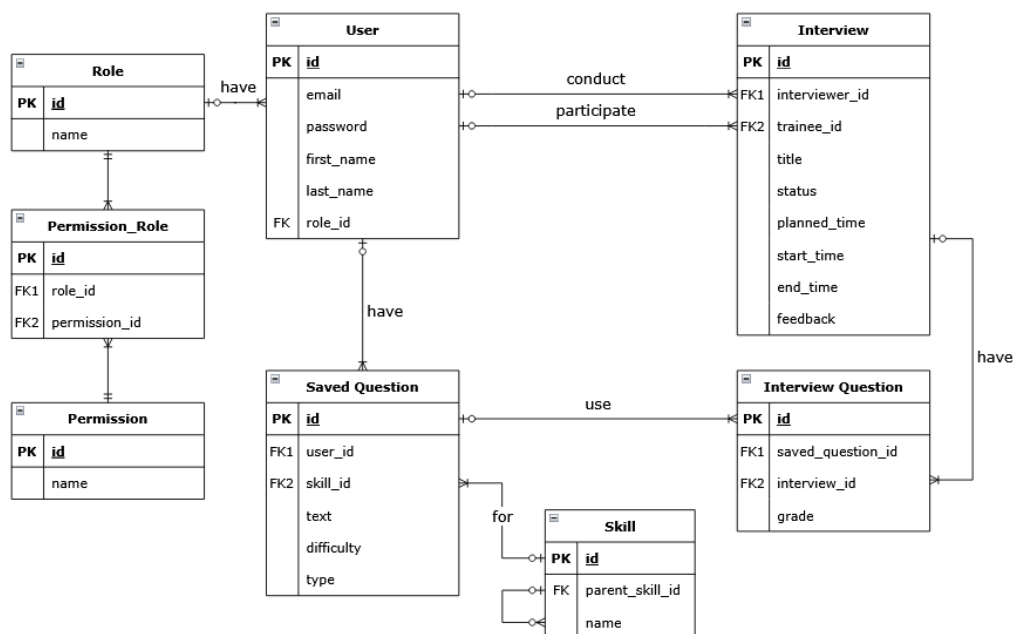


Рисунок 2.4 – Схеми бази даних

Хоча Amazon DynamoDB не є реляційною базою даних у класичному розумінні, її модель даних забезпечує ефективне логічне групування об'єктів завдяки використанню комбінованих ключів (partition key та sort key), а також підтримці глобальних і локальних secondary indexes.

## **2.4 Обґрунтування вибору технологічного стеку**

Підбір технологічного стеку є одним із ключових рішень на етапі проектування програмної системи, оскільки саме він визначає можливості реалізації функціональних і нефункціональних вимог. Сукупність обраних мов програмування, інструментів і хмарних сервісів повинна забезпечувати стабільність роботи, гнучкість масштабування, високу продуктивність і можливість швидкої адаптації системи до змін середовища у процесі її розвитку.

### **2.4.1 Мова програмування Python**

Для розробки безсерверних застосунків на платформі AWS Lambda підтримується декілька мов програмування, серед яких найпопулярнішими є JavaScript (Node.js), Python, Java, C#, Ruby та Go [33]. Вибір мови залежить від низки факторів, зокрема вимог до швидкості розробки, доступності бібліотек, ефективності інтеграції з іншими сервісами AWS, а також специфіки предметної області. [34] .

У межах реалізації даного проєкту основною мовою програмування було обрано Python, що є обґрунтованим рішенням з кількох причин. По-перше, Python має офіційно підтримуваний рантайм у AWS Lambda, який забезпечує стабільну роботу, регулярне оновлення та безпечне розгортання функцій без необхідності налаштовувати середовище вручну [33]. По-друге, Python де-факто є стандартом у науковому та дослідницькому середовищі, ключові бібліотеки для реалізації систем на базі штучного інтелекту, такі як HuggingFace

Transformers та LangChain, побудовані саме на Python. Важливо також врахувати популярність мови серед розробників.

Таким чином, Python повністю задовольняє вимоги до технологічного стеку системи автоматизації оцінки навичок у serverless-середовищі, забезпечуючи баланс між ефективністю розробки, багатством екосистеми та гнучкістю інтеграції.

#### **2.4.2 Використані бібліотеки для реалізації BE**

У межах реалізації серверної частини системи було обрано низку бібліотек Python, які забезпечують ефективну взаємодію з хмарними сервісами, побудову RESTful API, обробку HTTP-запитів, валідацію даних і реалізацію механізмів безпечної автентифікації користувачів. Вибір бібліотек здійснювався з урахуванням вимог до продуктивності, масштабованості та відповідності сучасним стандартам безпеки.

Для взаємодії із сервісами AWS використовується бібліотека boto3 — офіційна SDK для Python, що забезпечує прямий доступ до таких сервісів, як AWS Lambda, Amazon DynamoDB, Amazon S3 та інших [35]. Застосування boto3 дозволяє керувати ресурсами AWS безпосередньо з коду, реалізуючи операції зі створення, оновлення та зчитування даних. Бібліотека активно підтримується та оновлюється Amazon, що гарантує її сумісність із останніми версіями сервісів.

Організація обробки HTTP-запитів та побудова RESTful API реалізовані за допомогою фреймворку FastAPI. Цей фреймворк використовує можливості асинхронного програмування на основі бібліотеки asyncio, що дозволяє досягати високої продуктивності навіть при великій кількості одночасних запитів [36].

Оскільки FastAPI спочатку орієнтований на традиційні серверні середовища, для адаптації до serverless-архітектури AWS Lambda

використовується бібліотека Mangum. Вона виступає проміжним шаром, що трансформує події AWS API Gateway у стандартні HTTP-запити для FastAPI-додатка без необхідності змінювати основний код [37].

Для валідації вхідних даних застосовується бібліотека Pydantic, яка забезпечує перевірку типів даних на основі декларативного опису моделей. Бібліотека підтримує валідацію складних типів, включаючи спеціалізовані перевірки, наприклад, правильність форматування email-адрес за допомогою модуля pydantic[email] [38].

Реалізація механізмів безпеки автентифікації та обробки облікових даних здійснюється за допомогою бібліотек python-jose, passlib та bcrypt [39]. Бібліотека python-jose забезпечує генерацію і верифікацію JWT-токенів, а для захисту паролів використовується passlib у поєднанні з алгоритмом bcrypt.

### **2.4.3 TypeScript для побудови користувацької частини застосунку**

Ще одним важливим рішенням під час розробки користувацького інтерфейсу системи стало використання мови програмування TypeScript замість традиційного JavaScript. Попри те що JavaScript займав домінуючу роль у розробці веб-інтерфейсів протягом тривалого часу, останні роки спостерігається стійке зростання популярності TypeScript [40], що пояснюється потребою у створенні більш надійних і підтримуваних застосунків.

Вибір TypeScript у межах реалізації даної системи є обґрунтованим низкою технічних переваг. По-перше, TypeScript забезпечує статичну типізацію, що дозволяє виявляти помилки ще на етапі компіляції, а не під час виконання програми. Це істотно підвищує надійність коду та зменшує ризик виникнення критичних помилок під час виконання програми. По-друге, використання типів спрощує інтеграцію з бекендом. При зміні API або моделей даних система типів TypeScript негайно виявляє можливі несумісності, що дозволяє розробникам

оперативно адаптувати клієнтську частину без ризику втрати функціональності. Це забезпечує більш стабільне й контрольоване масштабування системи, особливо у процесі її подальшої еволюції. Крім того, TypeScript покращує читабельність і підтримку коду, що є важливим фактором для довготривалих проєктів із можливістю залучення нових розробників.

#### **2.4.4 ReactJS**

Для розробки користувацького інтерфейсу системи було обрано бібліотеку ReactJS, яка сьогодні є однією з найпоширеніших технологій для створення односторінкових веб-застосунків (SPA — Single Page Applications). React активно підтримується компанією Meta та має широку екосистему супровідних бібліотек і інструментів, що робить його стабільною та перспективною платформою для довгострокових проєктів [41].

Однією з головних переваг React є компонентно-орієнтована архітектура, яка забезпечує побудову інтерфейсу у вигляді набору незалежних, модульних і повторно використовуваних елементів. У межах даної системи це дозволяє ефективно структурувати різні частини користувацького інтерфейсу — зокрема модуль інтерв'ю, управління профілем користувача та навичками, що значно спрощує підтримку і розширення функціональності у майбутньому.

Крім того, React забезпечує гнучку інтеграцію з широким набором бібліотек для розширення базових можливостей, таких як керування маршрутизацією (React Router), управління формами (React Hook Form), аутентифікацією, управлінням глобальним станом (Redux) та стилізацією (TailwindCSS, Styled Components).

#### **2.4.5 Інші бібліотеки для реалізації користувацької частини**

Для реалізації користувацької частини системи було обрано низку бібліотек, спрямованих на забезпечення ефективної роботи інтерфейсу, взаємодії з сервером та підтримки високої якості коду.

Для організації навігації між сторінками в межах односторінкового застосунку використовується бібліотека `react-router-dom`, яка забезпечує побудову вкладених і динамічних маршрутів, а також налаштування доступу до окремих сторінок на основі ролі користувача. Оскільки система автентифікації реалізована через JWT-токени, маршрути захищаються за допомогою перевірки присутності та валідності токенів на клієнті.

Для роботи з формами застосовується бібліотека `react-hook-form`, яка забезпечує декларативне визначення форм, оптимізовану валідацію даних та мінімізацію ререндерингів компонентів.

Для виконання HTTP-запитів використовується `Axios`, який забезпечує гнучку конфігурацію запитів, обробку відповідей та інтеграцію з механізмами авторизації через заголовки авторизації, що містять JWT-токени.

Стилізація інтерфейсу реалізована за допомогою `Tailwind CSS` — утилітарного CSS-фреймворку, що дозволяє безпосередньо у шаблонах `JSX` визначати стилі елементів.

Для збірки застосунку використовується `Vite`, який забезпечує надзвичайно швидке оновлення змін під час розробки завдяки використанню нативних ES-модулів, а також оптимізовану збірку для продакшн-середовища.

Якість коду контролюється за допомогою `ESLint`, який перевіряє відповідність коду встановленим правилам стилю та допомагає виявляти потенційні помилки. Для перевірки правильного використання `React-хуків` додатково інтегровано плагін `eslint-plugin-react-hooks`.

Окрім основних бібліотек, у проєкті використовуються `dayjs` для роботи з датами, `react-tooltip` для реалізації контекстних підказок у інтерфейсі та `react-icons` для інтеграції графічних елементів без використання сторонніх ресурсів.

## РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО ПІДХОДУ

### 3.1 Реалізація основних компонентів запропонованої системи

#### 3.1.1 Створення шаблону проекту з використанням AWS SAM

Першим етапом практичної реалізації системи стала побудова базового шаблону інфраструктури за допомогою AWS SAM (Serverless Application Model). AWS SAM є розширенням стандарту AWS CloudFormation, що дозволяє описувати архітектуру serverless-застосунків у компактній декларативній формі за допомогою файлу `template.yaml` [42]. В шаблоні описується структура компонентів системи: функцій AWS Lambda, ресурсів API Gateway, IAM-ролей для доступу до ресурсів AWS, змінних середовища та інших конфігураційних параметрів (Додаток А) [42].

Ключовою частиною шаблону є секція `Resources`, де описуються архітектурні компоненти. Для кожної Lambda-функції визначаються такі основні параметри (Додаток Б):

- `Handler` – шлях до функції-обробника;
- `Runtime` – середовище виконання (у нашому випадку Python 3.11 або 3.12);
- `Events` – тригер подій, наприклад HTTP-запит через API Gateway;
- `Environment` – змінні середовища, що передаються у функцію під час виконання;
- `Layers` – підключення спільних бібліотек або коду через Lambda Layers.

У нашій системі кожна Lambda-функція відповідає окремому REST-контролеру, що чітко ізолює різні частини бізнес-логіки та дозволяє більш гнучко масштабувати й оновлювати систему. Така структура спрощує тестування та підтримку, оскільки кожна функція має єдиний обов'язок (Single Responsibility Principle).

Ще одним важливим аспектом є організація структури проєкту. Кожна Lambda-функція розміщена у власному підкаталозі, що містить обробник подій (handler), бізнес-логіку та специфічні залежності [31]. Для підвищення повторного використання коду спільні компоненти – такі як DTO, утиліти та обробники винятків – винесені в окремі шари (Lambda Layers) [31]. Це забезпечує централізоване управління загальним кодом і значно полегшує його супровід.

На рисунку 3.1 представлено структуру каталогів проєкту. Видно, що функції організовані у підкаталогах (наприклад, auth/, user/), що дозволяє легко орієнтуватися у проєкті та підтримувати модульність.

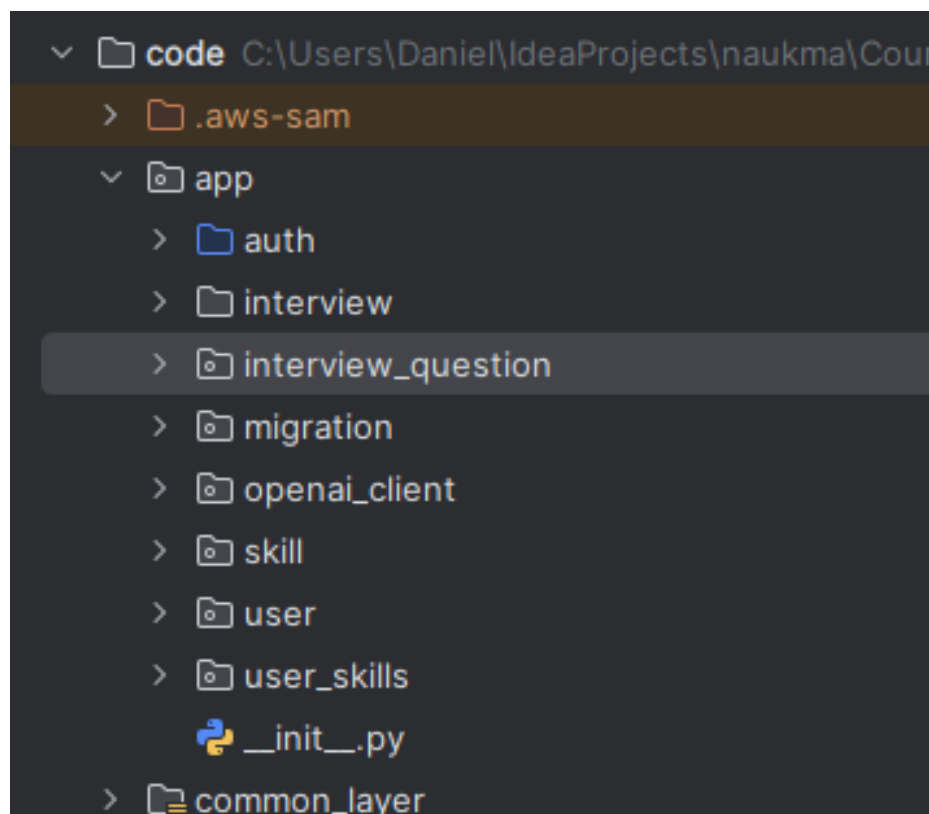


Рисунок 3.1 – Структура каталогів проєкту

Створення шаблону проєкту за допомогою AWS SAM дозволило швидко побудувати каркас архітектури системи, реалізувати базову інтеграцію з API Gateway та підготувати середовище для подальшої розробки.

### 3.1.2 Реалізація основної логіки з використанням багаторівневої архітектури

Для забезпечення підтримуваності, масштабованості та повторного використання коду у структурі кожної Lambda-функції реалізовано багаторівневу архітектуру, що включає три основні рівні: controller, service та repository. Такий підхід забезпечує чіткий розподіл відповідальностей між компонентами, підвищує якість розробки та спрощує подальше розширення системи.

На рівні controller здійснюється обробка вхідних HTTP-запитів та взаємодія з фреймворком FastAPI. Контролери відповідають за прийом даних від користувача, їх первинну обробку та делегування логіки обробки відповідним сервісам. Рівень service містить бізнес-логіку системи. Саме тут реалізуються основні процеси, пов'язані з проведенням mock-інтерв'ю, обробкою відповідей, аналізом результатів та генерацією рекомендацій для користувачів. Сервіси не взаємодіють безпосередньо з базою даних, а працюють через інтерфейси репозиторіїв, що підвищує гнучкість архітектури. Рівень repository відповідає за взаємодію з базою даних Amazon DynamoDB. Репозиторії реалізують операції читання, запису та оновлення даних.

На рисунку 3.2 представлено приклад коду контролера, що демонструє структуру обробки HTTP-запиту у FastAPI та передачу запиту до сервісного шару.

```

10 router = APIRouter(dependencies=[Depends(jwt_authentication)])
11 app = FastAPI()
12 user_service = UserService()
13
14
15 @router.post(path="/users", response_model=UserDto, status_code=201) 2 usages (2 dynamic) ▲ Danylo Shlapak
16 def create_user(user_dto: UserRegistrationDto):
17     return user_service.create_user(user_dto)
18
19
20 @router.put(path="/users/{user_id}", response_model=UserDto) 1 usage (1 dynamic) ▲ Danylo Shlapak
21 def update_user(user_id: str, user_update_dto: UserUpdateDto):
22     return user_service.update_user(user_id, user_update_dto)
23
24

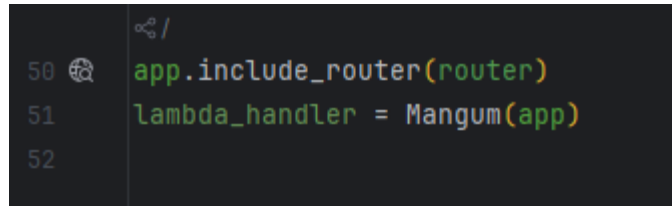
```

Рисунок 3.2 – Приклад коду обробника запитів

Всі спільні компоненти системи – такі як DTO-структури, обробники винятків, утиліти для обробки відповідей і базові сервіси – винесені до окремого AWS Lambda Layer. Це дозволяє підключати спільний код до різних функцій, що значно зменшує дублювання та спрощує супровід.

Важливою архітектурною практикою, що реалізована у проекті, є застосування підходу *dependency injection*. Компоненти системи отримують свої залежності через конструктори або спеціальні механізми реєстрації залежностей у FastAPI, що полегшує тестування, оскільки під час тестування можна легко підміняти реальні залежності їхніми тестовими імплементаціями.

Для інтеграції FastAPI із середовищем виконання AWS Lambda використовується бібліотека Mangum, яка адаптує стандартний інтерфейс FastAPI під специфікацію подій API Gateway. Це дозволяє повноцінно використовувати роутінг, валідацію, обробку запитів і генерацію документації OpenAPI без необхідності змінювати основну бізнес-логіку або структуру застосунку [37].



```
50 app.include_router(router)
51 lambda_handler = Mangum(app)
52
```

Рисунок 3.3 – Приклад використання бібліотеки-адаптера Mangum

Таким чином, впровадження багаторівневої архітектури та використання адаптерів для інтеграції із серверлес-середовищем дозволило створити модульну, розширювану і легко підтримувану систему, що відповідає сучасним стандартам веб-розробки.

### 3.2 Інтеграція LLM для генерації та аналізу відповідей

Одним із ключових компонентів системи автоматизованої оцінки навичок є інтеграція з великою мовною моделлю (LLM) для забезпечення динамічної генерації запитань, транскрипції відповідей у текстовий формат та аналізу відповідей кандидатів. В межах реалізації цієї функціональності використовується OpenAI SDK — офіційний клієнтський інструмент для взаємодії з моделями компанії OpenAI.

Для генерації технічних питань і проведення аналізу відповідей інтегровано модель GPT-4o, яка є однією з найпотужніших моделей для обробки та генерації тексту природною мовою [11]. Вона дозволяє створювати персоналізовані питання відповідно до профілю навичок користувача та адаптивно надавати зворотний зв'язок після отримання відповіді. Для транскрипції голосових відповідей у текстовий формат застосовується модель Whisper-1, що спеціалізується на розпізнаванні мовлення та підтримує багатомовність, що особливо важливо у контексті міжнародних користувачів.

Взаємодія з моделями OpenAI здійснюється через ChatCompletion API, де кожен запит формулюється у вигляді списку повідомлень (messages), що

включають роль відправника (system або user) та зміст повідомлення. Важливим аспектом інтеграції є правильне формування промптів, що забезпечує релевантність і точність відповідей моделі. У системі використовується двокомпонентна структура промпта: повідомлення системи встановлює контекст завдання, визначає критерії оцінки та правила генерації нових питань, тоді як повідомлення користувача передає конкретне питання та відповідь кандидата для аналізу [3].

Нижче наведено приклад промпта, що використовується для оцінки відповіді кандидата. Цей промпт налаштовує модель на виконання завдання технічного інтерв'ю, формулюючи чіткі критерії оцінювання.

```
def analyze_candidate_answer(self, question: str, answer: str) -> AnalysisResult: 1 usage
    prompt = (
        "You are an expert technical interviewer and a specialist in the subject area of the provided question. "
        "Your role is to carefully assess the candidate's answer from a professional technical perspective, "
        "focusing on correctness, clarity, depth of knowledge, and relevance. "
        "Consider nuances specific to the domain (e.g., programming languages, algorithms, "
        "system design, etc.) and ensure your evaluation is highly accurate and context-aware."
        " - grade: number from 1 to 10 indicating the answer quality"
        " - comments: detailed feedback on strengths and weaknesses"
        " - nextQuestion: a suggested next question based on the candidate's response"
        f"Question: \'{question}\'\n"
        f"Candidate's answer: \'{answer}\'"
    )
```

Рисунок 3.4 – Приклад промпта, що використовується для оцінки відповіді

Для інтеграції з OpenAI у межах проєкту реалізовано власний Python-клієнт під назвою OpenAIClient. Цей клієнт інкапсулює логіку взаємодії з API OpenAI, зокрема функції для генерації технічних питань, транскрипції голосових відповідей за допомогою Whisper-1 та аналізу результатів із використанням GPT-4o. Такий підхід дозволив централізувати всі виклики до зовнішнього API та зробив систему більш керованою і зручною для супроводу.

Нижче представлено ключовий фрагмент реалізації класу OpenAIClient, у якому показано послідовність кроків для аналізу голосової відповіді кандидата. На

першому етапі використовується модель Whisper-1 для транскрипції аудіофайлу у текстовий формат. Після отримання тексту відповіді система надсилає його разом із питанням до GPT-4o для виконання аналізу. В результаті формується структурована відповідь, яка містить оцінку за кожним із критеріїв та адаптивне наступне питання.

```
def process_audio_answer(self, audio_file: BinaryIO, question: str) -> AnalysisResult: 1 usage
    answer_text = self.transcribe_audio(audio_file)
    return self.analyze_candidate_answer(question, answer_text)

def transcribe_audio(self, audio_file: BinaryIO) -> str: 2 usages
    transcript = self.client.audio.transcriptions.create(
        model="whisper-1",
        file=audio_file
    )
    return transcript.text

def analyze_candidate_answer(self, question: str, answer: str) -> AnalysisResult: 1 usage
    prompt = (
        """
        """
    )

    response_schema = {...}

    response = self.client.chat.completions.create(
        model="gpt-4o",
        messages=[{"role": "system", "content": prompt}],
        temperature=0.2,
        max_tokens=1000,
        response_format=response_schema
    )

    content = response.choices[0].message.content
    return json.loads(content)
```

Рисунок 3.5 – Код для аналізу голосової відповіді кандидата

При налаштуванні виклику API увагу також приділено параметрам генерації, які впливають на якість відповіді моделі. Параметр `temperature` використовується для контролю рівня креативності відповідей і встановлений у межах 0.2–0.3 для досягнення більш детермінованих результатів. Параметр `max_tokens` обмежує довжину відповіді моделі, що дозволяє уникати надмірних відповідей. Додаткові параметри, такі як `top_p`, `frequency_penalty` та `presence_penalty`, можуть налаштовуватися залежно від конкретних завдань.

### 3.3 Тестування API застосунку

В рамках розробки нашої системи було прийнято рішення насамперед зосередитися на end-to-end (E2E) тестуванні, яке охоплює повний цикл взаємодії між ключовими компонентами. Такий підхід дозволив протестувати основні сценарії використання – від автентифікації користувача до генерації питань і аналізу відповідей – без необхідності занурення у внутрішню реалізацію кожного окремого модуля [43].

Це рішення є виправданим з огляду на архітектуру системи, що реалізована як набір серверлес-функцій, інтегрованих через API Gateway. Оскільки основний функціонал надається через зовнішній API, логічно тестувати саме ці точки взаємодії з позиції кінцевого користувача. Таким чином досягається впевненість у стабільності роботи всіх компонентів.

```
def test_login_success():
    make_request( method: "POST", endpoint: "register", TEST_USER)

    response = make_request( method: "POST", endpoint: "login", data: {
        "email": TEST_USER["email"],
        "password": "password"
    })

    assert response.status_code == 200
    data = response.json()
    assert "token" in data

def test_login_invalid_credentials():
    make_request( method: "POST", endpoint: "register", TEST_USER)

    response = make_request( method: "POST", endpoint: "login", data: {
        "email": TEST_USER["email"],
        "password": "wrong_password"
    })

    assert response.status_code == 401
    assert "invalid credentials" in response.json()["detail"].lower()
```

Рисунок 3.6 – Приклад коду E2E тестів

### **3.4 Особливості розгортання системи з використанням serverless архітектури**

Процес розгортання та керування ресурсами автоматизовано за допомогою AWS Serverless Application Model (AWS SAM), що дозволяє декларативно описувати архітектуру системи та мінімізувати втручання у процес оновлення і підтримки [42].

#### **3.4.1 Локальне розгортання функцій**

Для локальної розробки та тестування функцій застосовується механізм контейнеризації на основі Docker. Docker є популярним інструментом для створення ізольованих середовищ виконання, що дозволяє моделювати реальні умови роботи AWS Lambda. Завдяки цьому розробник може локально тестувати функції у середовищі, яке максимально наближене до реального [21].

Під час запуску команди `sam build`, AWS SAM автоматично створює для кожної функції окремий Docker Image. Ці образи будуються на основі офіційних базових образів AWS Lambda для відповідної мови програмування. Кожен образ включає в себе код функції, її залежності та середовище виконання.

Водночас слід зазначити, що процес побудови, запуску та налагодження Lambda-функцій локально є суттєво складнішим і повільнішим порівняно з традиційною розробкою монолітних або класичних серверних застосунків. Кожен цикл змін у коді передбачає необхідність повторної побудови контейнера, що додає затримки та ускладнює швидку ітерацію під час розробки.

#### **3.4.2 Розгортання функцій в середовищі AWS**

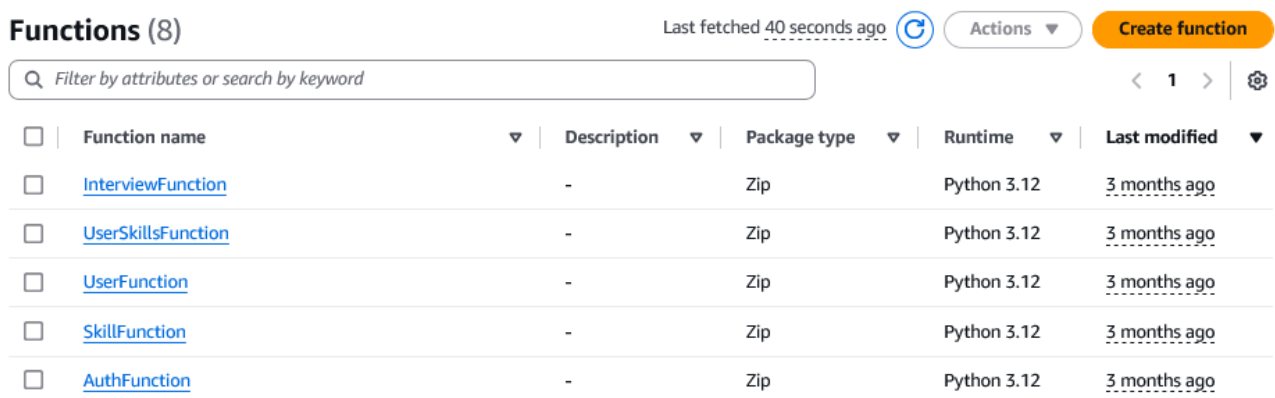
Після тестування функцій локально наступним кроком є розгортання всієї інфраструктури у середовищі AWS. Це здійснюється за допомогою команди: `sam deploy --guided`. Під час першого виконання цієї команди ініціюється

інтерактивний процес налаштування основних параметрів: імені стеку CloudFormation, регіону AWS, ролей доступу, змінних середовища для функцій, політик оновлення та дозволів на створення нових ресурсів. Цей процес не є повністю автоматизованим і вимагає уважного заповнення полів та розуміння принципів роботи AWS-середовища [42].

Після завершення первинного guided-деплою всі налаштування зберігаються у файлі `samconfig.toml`, що дозволяє виконувати наступні оновлення функцій і ресурсів значно швидше (Додаток В). Для подальших розгортань достатньо виконати просту команду: `sam deploy` [42].

Під час кожного деплою AWS SAM трансформує декларативний шаблон `template.yaml` у стандартний шаблон AWS CloudFormation, який автоматично керує створенням, оновленням або видаленням ресурсів відповідно до змін у кодї чи конфігурації. Це дозволяє не лише розгорнути нові функції, а й вносити оновлення до вже існуючих без ризику ручних помилок або несинхронізованих змін [42].

Особливістю цього процесу є можливість інкрементального оновлення: якщо зміни стосуються лише окремої функції чи компонента, AWS SAM оновлює лише відповідні ресурси, що суттєво зменшує час деплою і мінімізує ризику порушення стабільності роботи інших частин системи.



<input type="checkbox"/>	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	<a href="#">InterviewFunction</a>	-	Zip	Python 3.12	3 months ago
<input type="checkbox"/>	<a href="#">UserSkillsFunction</a>	-	Zip	Python 3.12	3 months ago
<input type="checkbox"/>	<a href="#">UserFunction</a>	-	Zip	Python 3.12	3 months ago
<input type="checkbox"/>	<a href="#">SkillFunction</a>	-	Zip	Python 3.12	3 months ago
<input type="checkbox"/>	<a href="#">AuthFunction</a>	-	Zip	Python 3.12	3 months ago

Рисунок 3.7 – Список розгорнутих Lambda-функцій в середовищі AWS

## РОЗДІЛ 4: АНАЛІЗ РЕЗУЛЬТАТІВ

### 4.1 Порівняння результатів із початковими гіпотезами

На початковому етапі дослідження було сформульовано низку гіпотез щодо функціональних можливостей та стабільності роботи запропонованої системи. Зокрема, передбачалося, що інтеграція великої мовної моделі (LLM) дозволить забезпечити адаптивну генерацію питань у режимі реального часу, а також автоматизований аналіз відповідей кандидатів із формуванням якісного зворотного зв'язку. Крім того, очікувалося, що використання serverless-архітектури забезпечить гнучкість у масштабуванні та мінімізує потребу в складному управлінні інфраструктурою [21].

Результати реалізації підтвердили більшість початкових припущень. Функціонал генерації питань успішно працює у режимі реального часу: система формує питання, що адаптуються до обраного переліку навичок користувача, і демонструє стабільність навіть при підвищеному навантаженні. Аналіз відповідей за допомогою GPT-4o дозволяє отримати детальний зворотний зв'язок з оцінкою ключових критеріїв, що підтверджує практичну ефективність інтеграції LLM у навчальні системи.

Serverless-архітектура також підтвердила свою доцільність: автоматичне масштабування та використання декларативних шаблонів (AWS SAM) значно спростили розгортання та оновлення системи. Практичні результати показали, що інфраструктура залишається стабільною при додаванні нових функцій без порушення роботи вже існуючих компонентів.

Водночас досвід розробки виявив деякі обмеження, які частково уточнюють первинні гіпотези. Зокрема, при інтеграції з OpenAI API були зафіксовані затримки у відповідях, що впливає на загальний досвід користувача під час проходження інтерв'ю. Окрім цього, важливою стала необхідність

ретельного налаштування промптів для отримання стабільних і релевантних результатів, що потребує окремої уваги на етапі розробки.

## 4.2 Аналіз переваг і недоліків serverless архітектури

У процесі реалізації запропонованої системи було застосовано serverless-архітектуру на базі сервісів AWS, що дозволило на практиці оцінити як її сильні сторони, так і існуючі обмеження.

Однією з ключових переваг serverless-архітектури є високий рівень автоматизації процесів розгортання. Завдяки використанню декларативних шаблонів AWS SAM (зокрема, файлу `template.yaml`) розробник отримує можливість стандартизувати процес створення, оновлення та масштабування компонентів системи. Це суттєво знижує ймовірність помилок, пов'язаних із ручним налаштуванням інфраструктури, і дозволяє зосередитися на розробці бізнес-логіки замість витрат часу на обслуговування серверів. Додатково архітектура забезпечує автоматичне масштабування обчислювальних ресурсів відповідно до реального навантаження, що дозволяє системі ефективно адаптуватися до змін кількості користувачів або активності без потреби завчасного резервування ресурсів.

Разом із тим, під час розробки системи були виявлені й певні виклики. Serverless-підхід вимагає від розробника глибшого розуміння хмарних сервісів і принципів подієво-орієнтованої обробки запитів. Це ускладнює процес розробки у порівнянні з традиційними архітектурами, де управління сервером здійснюється безпосередньо. Окрім того, для локального тестування Lambda-функцій необхідним є використання контейнеризації, що призводить до збільшення часу на побудову, запуск і налагодження функцій. Це створює певні обмеження у швидкості розробницького циклу, особливо на етапах частих змін і оновлень коду.

Додатковим аспектом, що потребує врахування, є затримки при так званому cold start функцій, які виникають під час першого запуску після періоду неактивності. Ці затримки можуть впливати на час відповіді системи і формувати відчуття меншої швидкодії у користувачів при першому зверненні.

Підсумовуючи, можна зазначити, що serverless-архітектура продемонструвала високу ефективність у реалізації масштабованої системи та підтвердила свою доцільність для проєктів, що вимагають автоматизації розгортання і адаптивності до навантажень. Проте ця архітектура також висуває додаткові вимоги до кваліфікації розробників і створює виклики, пов'язані з процесами локальної розробки та налагодження.

### **4.3 Основні результати дослідження**

Проведене дослідження дозволило розробити та впровадити функціональну систему автоматизованої оцінки навичок для технічних інтерв'ю, що поєднує у собі інтеграцію з великою мовною моделлю (LLM) та архітектуру serverless на базі сервісів AWS. У процесі реалізації було досягнуто низки важливих результатів, що підтверджують доцільність обраних рішень.

Перш за все, підтверджено можливість динамічної генерації запитань у реальному часі, що забезпечує адаптивність інтерв'ю залежно від обраних навичок кандидата. Це дозволяє кожному користувачеві проходити унікальні сценарії підготовки, що підвищує ефективність процесу навчання та тестування. Інтеграція моделі GPT-4o забезпечила не лише генерацію питань, а й автоматизований аналіз відповідей з формуванням деталізованого зворотного зв'язку. Такий підхід дозволяє кандидатам отримувати об'єктивну оцінку своїх знань та бачити конкретні рекомендації для покращення.

Система продемонструвала гнучкість і масштабованість завдяки використанню serverless-архітектури. Це дозволило забезпечити стабільну

роботу всіх функціональних модулів під час оновлень і зміни конфігурацій без негативного впливу на інші компоненти. Декларативний підхід до опису інфраструктури через AWS SAM значно спростив процес розгортання та оновлення системи, що було критично важливо на етапах швидкої розробки та тестування.

Особливе значення під час розробки мало створення ефективних промптів для взаємодії з LLM. Досвід показав, що якість отриманих результатів безпосередньо залежить від чіткості і правильності сформульованих інструкцій для моделі, що підтверджує важливість розвитку напряму `prompt engineering` у подібних системах.

#### **4.4 Перспективи подальших досліджень**

Одним з потенційних напрямків розвитку є впровадження більш адаптивних механізмів генерації питань, зокрема інтеграція з RAG, що дозволить підвищити релевантність та точність сформованого контенту завдяки використанню зовнішніх джерел знань, що особливо актуально для покриття спеціалізованих тем і складних сценаріїв.

Подальший розвиток механізмів динамічного підбору рівня складності запитань залежно від результатів попередніх відповідей кандидата відкриває можливість створення більш персоналізованих і гнучких сценаріїв інтерв'ю. Це значно підвищить ефективність підготовки користувачів і дозволить краще адаптувати процес до індивідуальних потреб.

Окремим важливим напрямом удосконалення є оптимізація взаємодії з LLM, зокрема вдосконалення стратегій формування промптів для досягнення стабільніших і передбачуваніших результатів генерації.

З архітектурної точки зору доцільним є дослідження підходів до зменшення затримок cold start у Lambda-функціях і вивчення альтернативних варіантів розгортання для оптимізації швидкодії.

## ВИСНОВКИ

У ході виконання курсової роботи було досліджено можливості LLM для автоматизації оцінки технічних навичок під час mock-інтерв'ю. Розроблено систему, що інтегрує LLM для генерації запитань і аналізу відповідей із serverless-архітектурою на базі AWS Lambda.

Використання serverless-архітектури на базі AWS Lambda із декларативним описом ресурсів через AWS SAM продемонструвало значні переваги з точки зору автоматизації процесів розгортання та масштабування. Завдяки автоматичному керуванню інфраструктурою та інкрементальному оновленню стеку CloudFormation вдалося мінімізувати операційні витрати та оперативно впроваджувати нові функціональні можливості без ризику порушити роботу існуючих компонентів .

Водночас практичний досвід показав, що застосування безсерверного підходу накладає певні обмеження на локальну розробку. Зокрема, цикл ітеративного тестування сповільнюється через необхідність побудови та запуску Docker-образів для кожної Lambda-функції, а явище cold-start призводить до затримок при першому виклику функцій після періоду простою. Ці фактори варто враховувати під час планування робочих процесів і налаштування середовища розробки.

Перспективним напрямом подальших досліджень є впровадження Retrieval-Augmented Generation для підвищення релевантності генерованих запитань за рахунок використання зовнішніх джерел знань, а також розвиток методів prompt engineering для досягнення більш стабільних та передбачуваних результатів взаємодії з LLM.

## ВИКОРИСТАНІ ДЖЕРЕЛА

- [1] A. Vaswani *et al.*, “Attention Is All You Need,” Aug. 02, 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.
- [2] Z. Ji *et al.*, “Survey of Hallucination in Natural Language Generation,” *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–38, Dec. 2023, doi: 10.1145/3571730.
- [3] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, “Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing,” Jul. 28, 2021, *arXiv*: arXiv:2107.13586. doi: 10.48550/arXiv.2107.13586.
- [4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” May 24, 2019, *arXiv*: arXiv:1810.04805. doi: 10.48550/arXiv.1810.04805.
- [5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners”.
- [6] “[2102.09690] Calibrate Before Use: Improving Few-Shot Performance of Language Models.” Accessed: Apr. 19, 2025. [Online]. Available: <https://arxiv.org/abs/2102.09690>
- [7] M. Shoyebi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, “Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism,” Mar. 13, 2020, *arXiv*: arXiv:1909.08053. doi: 10.48550/arXiv.1909.08053.
- [8] S. Roller *et al.*, “Recipes for building an open-domain chatbot,” Apr. 30, 2020, *arXiv*: arXiv:2004.13637. doi: 10.48550/arXiv.2004.13637.
- [9] E. J. Hu *et al.*, “LoRA: Low-Rank Adaptation of Large Language Models,” Oct. 16, 2021, *arXiv*: arXiv:2106.09685. doi: 10.48550/arXiv.2106.09685.
- [10] N. Ding *et al.*, “Parameter-efficient fine-tuning of large-scale pre-trained language models,” *Nat. Mach. Intell.*, vol. 5, no. 3, pp. 220–235, Mar. 2023, doi: 10.1038/s42256-023-00626-4.
- [11] OpenAI *et al.*, “GPT-4 Technical Report,” Mar. 04, 2024, *arXiv*: arXiv:2303.08774. doi: 10.48550/arXiv.2303.08774.
- [12] Y. Dang *et al.*, “Explainable and Interpretable Multimodal Large Language Models: A Comprehensive Survey,” Dec. 03, 2024, *arXiv*: arXiv:2412.02104. doi: 10.48550/arXiv.2412.02104.
- [13] T. B. Brown *et al.*, “Language Models are Few-Shot Learners,” Jul. 22, 2020, *arXiv*: arXiv:2005.14165. doi: 10.48550/arXiv.2005.14165.
- [14] Z. Levonian *et al.*, “Retrieval-augmented Generation to Improve Math Question-Answering: Trade-offs Between Groundedness and Human Preference,” Nov. 11, 2023, *arXiv*: arXiv:2310.03184. doi: 10.48550/arXiv.2310.03184.
- [15] E. Kasneci *et al.*, “ChatGPT for good? On opportunities and challenges of large language models for education,” *Learn. Individ. Differ.*, vol. 103, p. 102274, Apr. 2023, doi: 10.1016/j.lindif.2023.102274.

- [16] R. AlAli and Y. Wardat, "Opportunities and Challenges of Integrating Generative Artificial Intelligence in Education," *Int. J. Relig.*, vol. 5, no. 7, pp. 784–793, May 2024, doi: 10.61707/8y29gv34.
- [17] E. Hashimoto, M. Nakano, T. Sakurai, S. Shiramatsu, T. Komazaki, and S. Tsuchiya, "A Career Interview Dialogue System using Large Language Model-based Dynamic Slot Generation," Dec. 22, 2024, *arXiv*: arXiv:2412.16943. doi: 10.48550/arXiv.2412.16943.
- [18] J. Barambones, C. Moral, A. de Antonio, R. Imbert, L. Martínez-Normand, and E. Villalba-Mora, "ChatGPT for Learning HCI Techniques: A Case Study on Interviews for Personas," *IEEE Trans. Learn. Technol.*, vol. 17, pp. 1460–1475, 2024, doi: 10.1109/TLT.2024.3386095.
- [19] "https://annals-csis.org/Volume\_39/drp/pdf/2063.pdf." Accessed: Apr. 20, 2025. [Online]. Available: [https://annals-csis.org/Volume\\_39/drp/pdf/2063.pdf](https://annals-csis.org/Volume_39/drp/pdf/2063.pdf)
- [20] "Serverless Architectures," *martinfowler.com*. Accessed: Apr. 20, 2025. [Online]. Available: <https://martinfowler.com/articles/serverless.html>
- [21] I. Baldini *et al.*, "Serverless Computing: Current Trends and Open Problems," Jun. 10, 2017, *arXiv*: arXiv:1706.03178. doi: 10.48550/arXiv.1706.03178.
- [22] M. Shahrhad *et al.*, "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," Jun. 06, 2020, *arXiv*: arXiv:2003.03423. doi: 10.48550/arXiv.2003.03423.
- [23] W. Zayat and O. Senvar, "Framework Study for Agile Software Development Via Scrum and Kanban," *Int. J. Innov. Technol. Manag.*, Jul. 2020, doi: 10.1142/S0219877020300025.
- [24] A. Rasnaxis and S. Berzisa, "Method for Adaptation and Implementation of Agile Project Management Methodology," *Procedia Comput. Sci.*, Mar. 2017, doi: 10.1016/J.PROCS.2017.01.055.
- [25] T. Thesing, C. Feldmann, and M. Burchardt, "Agile versus Waterfall Project Management: Decision Model for Selecting the Appropriate Approach to a Project," *Procedia Comput. Sci.*, Jan. 2021, doi: 10.1016/J.PROCS.2021.01.227.
- [26] K. T. Ulrich and S. D. Eppinger, "PRODUCT DESIGN AND DEVELOPMENT," Mar. 2018, doi: 10.4337/9781784718152.00017.
- [27] P. Borra, "COMPARISON AND ANALYSIS OF LEADING CLOUD SERVICE PROVIDERS (AWS, AZURE AND GCP)," *Int. J. Adv. Res. Eng. Technol.*, vol. 15, pp. 266–278, Jun. 2024, doi: 10.17605/OSF.IO/T2DHW.
- [28] M. Villamizar *et al.*, *Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud*. 2015. doi: 10.1109/ColumbianCC.2015.7333476.
- [29] T. Fred, "Serverless Frameworks: A Comparative Study of AWS Lambda, Google Cloud Functions, and Azure Functions," Sep. 2019.
- [30] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Jan. 1994, [Online]. Available: <http://www.ulb.tu-darmstadt.de/tocs/59840579.pdf>

- [31] “Managing Lambda dependencies with layers - AWS Lambda.” Accessed: Apr. 20, 2025. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/chapter-layers.html>
- [32] “What is Amazon DynamoDB? - Amazon DynamoDB.” Accessed: Apr. 26, 2025. [Online]. Available: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>
- [33] “Lambda runtimes - AWS Lambda.” Accessed: Apr. 26, 2025. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html>
- [34] S. Shrestha, “Comparing Programming Languages used in AWS Lambda for Serverless Architecture,” Jan. 2019, [Online]. Available: <https://www.theseus.fi/handle/10024/227117>
- [35] “Boto3 1.38.3 documentation.” Accessed: Apr. 26, 2025. [Online]. Available: <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>
- [36] “FastAPI.” Accessed: Apr. 26, 2025. [Online]. Available: <https://fastapi.tiangolo.com/>
- [37] “Mangum.” Accessed: Apr. 26, 2025. [Online]. Available: <https://mangum.fastapiexpert.com/>
- [38] “Welcome to Pydantic - Pydantic.” Accessed: Apr. 26, 2025. [Online]. Available: <https://docs.pydantic.dev/latest/>
- [39] “python-jose — python-jose 0.2.0 documentation.” Accessed: Apr. 26, 2025. [Online]. Available: <https://python-jose.readthedocs.io/en/latest/>
- [40] “2024 Stack Overflow Developer Survey.” Accessed: Apr. 26, 2025. [Online]. Available: <https://survey.stackoverflow.co/2024/>
- [41] “React.” Accessed: Apr. 26, 2025. [Online]. Available: <https://react.dev/>
- [42] “What is the AWS Serverless Application Model (AWS SAM)? - AWS Serverless Application Model.” Accessed: Apr. 26, 2025. [Online]. Available: <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html>
- [43] “(PDF) Best Practices for Testing in Micro services: Ensuring Quality Across Distributed Systems.” Accessed: May 01, 2025. [Online]. Available: [https://www.researchgate.net/publication/388419808\\_Best\\_Practices\\_for\\_Testing\\_in\\_Micro\\_services\\_Ensuring\\_Quality\\_Across\\_Distributed\\_Systems](https://www.researchgate.net/publication/388419808_Best_Practices_for_Testing_in_Micro_services_Ensuring_Quality_Across_Distributed_Systems)

## ДОДАТКИ

### Додаток А (обов'язковий)

Фрагмент файлу template.yaml

```
1  AWSTemplateFormatVersion: '2010-09-09'
2  Transform: AWS::Serverless-2016-10-31
3
4  Globals:
5    Function:
6      Timeout: 50
7      MemorySize: 256
8    Api:
9      Cors:
10       AllowOrigin: "'http://localhost:5173'"
11       AllowHeaders: "'Authorization, Cache-Control, Content-Type, AcceptLanguage'"
12       AllowMethods: "'OPTIONS,GET,POST,PUT,DELETE,PATCH'"
13
14  Resources:
15    CommonLayer:
16      Type: AWS::Serverless::LayerVersion
17      Properties:
18        LayerName: common-layer
19        ContentUri: common_layer/
20        CompatibleRuntimes:
21          - python3.12
22      Metadata:
23        BuildMethod: python3.12
```

Додаток Б  
(обов'язковий)  
Фрагмент шаблону SAM для опису Lambda-функції

```
127 AuthFunction:
128   Type: AWS::Serverless::Function
129   Properties:
130     FunctionName: AuthFunction
131     CodeUri: app/auth/
132     Handler: controller.auth_controller.lambda_handler
133     Runtime: python3.12
134     Architectures:
135       - x86_64
136     Layers:
137       - !Ref CommonLayer
138     Events:
139       GetAllSkillTrees:
140         Type: Api
141         Properties:
142           Path: /auth/register
143           Method: post
144       GetSkillTreeByRootId:
145         Type: Api
146         Properties:
147           Path: /auth/login
148           Method: post
149     Policies:
150       - AWSLambdaBasicExecutionRole
151       - AmazonDynamoDBFullAccess
```

Додаток В  
(обов'язковий)  
Фрагмент налаштувань з samconfig.toml

```
1 # More information about the configuration file can be found here:
2 # https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-config.html
3 version = 0.1
4
5 [default.global.parameters]
6 stack_name = "app"
7
8 [default.build.parameters]
9 cached = true
10 parallel = true
11
12 [default.validate.parameters]
13 lint = true
14
15 [default.deploy.parameters]
16 capabilities = "CAPABILITY_IAM"
17 confirm_changeset = true
18 resolve_s3 = true
19 stack_name = "coursework-app"
20 s3_prefix = "coursework-app"
21 region = "eu-central-1"
22 image_repositories = []
23
24 [default.package.parameters]
25 resolve_s3 = true
26
27 [default.sync.parameters]
28 watch = true
29
30 [default.local_start_api.parameters]
31 warm_containers = "EAGER"
32
33 [default.local_start_lambda.parameters]
34 warm_containers = "EAGER"
```