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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Адаптивні алгоритми перевірки знань у вивченні мови програмування Java»

на здобуття освітнього ступеня магістра
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

_____ Юрій ШЕВЧУК
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Юрій ШЕВЧУК

Керівник: Ірина ЗАМРІЙ
д.т.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

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

Шевчуку Юрію Олександровичу

1. Тема кваліфікаційної роботи: «Адаптивні алгоритми перевірки знань у вивченні мови програмування Java»

керівник кваліфікаційної роботи Ірина ЗАМРІЙ д.т.н., доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, параметри адаптивного оцінювання, методи адаптації складності завдань, вимоги до точності оцінювання знань студентів з програмування.

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

1. Дослідження методів адаптивного оцінювання знань.
2. Аналіз алгоритмів перевірки знань у навчальних платформах.
3. Розробка адаптивного алгоритму для перевірки знань з програмування.
4. Проектування системи адаптивного оцінювання знань для платформи Java.

5. Тестування та оцінка ефективності розробленого алгоритму.

5. Перелік ілюстративного матеріалу: *презентація*

1. Мета, об’єкт та предмет дослідження
2. Недоліки існуючих методів перевірки знань.
3. Математична модель.
4. Математична модель.
5. Алгоритм адаптивного тестування.
6. Екранні форми.
7. Таблиця компонентів розробленої системи.
8. Архітектура системи оцінювання
9. Технології розробки
10. Порівняльний аналіз
11. Порівняльний аналіз
12. Порівняння ключових показників ефективності
13. Висновки
14. Публікації та апробація роботи

6. Дата видачі завдання«16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.10-19.10.2024	
2	Вивчення матеріалів для аналізу сучасних підходів до адаптивного оцінювання знань	20.10-25.10.2024	
3	Дослідження алгоритмів адаптивного тестування	24.10-31.10.2024	
4	Аналіз впливу адаптивних алгоритмів на якість навчання програмуванню	01.11-10.11.2024	
5	Розробка адаптивного алгоритму для перевірки знань з програмування	11.11-19.11.2024	
6	Тестування алгоритму та оцінка його ефективності	20.11-22.11.2024	
7	Оформлення роботи: вступ, висновки, реферат	23.11-25.11.2024	
8	Розробка демонстраційних матеріалів	26.11-28.11.2024	
9	Попередній захист роботи	29.11-02.12.2024	

Здобувач вищої освіти _____ Юрій ШЕВЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ЗАМРІЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 79 стор., 6 табл., 9 рис., 25 джерел.

Мета роботи – підвищення ефективності перевірки знань шляхом впровадження адаптивних алгоритмів, що враховують індивідуальні особливості студентів та їхній рівень підготовки, для вивчення мови програмування Java.

Об'єкт дослідження – процес перевірки знань студентів у навчанні мови програмування Java.

Предмет дослідження - адаптивні алгоритми перевірки знань, які дозволяють автоматично налаштовувати рівень складності завдань відповідно до знань і навичок студентів, а також їхньої прогресії у вивченні Java.

У роботі проведено аналіз проблематики існуючих методів перевірки знань з програмування та обґрунтовано необхідність розробки адаптивної системи оцінювання. Досліджено сучасні алгоритми та підходи до адаптивного оцінювання знань, визначено їх переваги та недоліки. Спроектовано архітектуру та основні компоненти адаптивної системи оцінювання знань з Java-програмування.

Розроблено алгоритми адаптивного тестування та оцінювання практичних навичок програмування, які враховують індивідуальні особливості студентів та забезпечують персоналізований підхід до навчання. Реалізовано систему з використанням сучасних технологій та фреймворків мови Java. Проведено тестування та оцінку ефективності розробленої системи в реальному навчальному процесі.

Запропоновано рекомендації щодо вдосконалення методики оцінювання знань з програмування на Java та визначено перспективи подальшого розвитку дослідження, зокрема можливості інтеграції з іншими освітніми платформами та використання технологій штучного інтелекту для персоналізації оцінювання.

Результати дослідження мають практичну цінність для закладів вищої освіти та курсів з підготовки ІТ-фахівців. Впровадження розробленої системи дозволить

підвищити ефективність оцінювання знань та навичок студентів, забезпечити адаптивність навчального процесу та сприятиме підготовці висококваліфікованих спеціалістів з програмування на Java.

Подальші дослідження можуть бути спрямовані на вдосконалення алгоритмів адаптивного оцінювання, розширення функціональних можливостей системи та проведення довгострокових досліджень її ефективності в підготовці IT-фахівців.

Результати дослідження можуть бути використані для модернізації існуючих систем оцінювання знань з програмування, розробки нових освітніх платформ та вдосконалення навчальних програм з підготовки спеціалістів у галузі інформаційних технологій.

КЛЮЧОВІ СЛОВА: JAVA, АДАПТИВНЕ ОЦІНЮВАННЯ, ТЕСТУВАННЯ, АЛГОРИТМИ, ПРОГРАМУВАННЯ, ОНЛАЙН-ПЛАТФОРМА, ПРАКТИЧНІ НАВИЧКИ, ПЕРСОНАЛІЗАЦІЯ

ABSTRACT

Text part of the master's qualification work: 79 pages, 6 table, 9 pictures, 25 sources.

The purpose of the work is to increase the efficiency of knowledge testing by implementing adaptive algorithms that take into account the individual characteristics of students and their level of preparation for studying the Java programming language.

The object of the study is the process of testing students' knowledge in learning the Java programming language.

The subject of the study is adaptive algorithms for testing knowledge, which allow automatically adjusting the level of complexity of tasks in accordance with the knowledge and skills of students, as well as their progression in learning Java.

The work analyzes the problems of existing methods for testing programming knowledge and justifies the need to develop an adaptive assessment system. Modern algorithms and approaches to adaptive knowledge assessment are studied, their advantages and disadvantages are identified. The architecture and main components of an adaptive Java programming knowledge assessment system are designed.

Algorithms for adaptive testing and assessment of practical programming skills are developed, which take into account the individual characteristics of students and provide a personalized approach to learning. A system was implemented using modern technologies and Java language frameworks. Testing and evaluation of the effectiveness of the developed system in the real educational process were carried out.

Recommendations were offered for improving the methodology for assessing knowledge in Java programming and prospects for further development of the study were identified, in particular, the possibility of integration with other educational platforms and the use of artificial intelligence technologies for personalizing assessment.

The results of the study have practical value for higher education institutions and courses for training IT specialists. The implementation of the developed system will increase the effectiveness of assessing students' knowledge and skills, ensure the

adaptability of the educational process and contribute to the training of highly qualified specialists in Java programming.Further research can be aimed at improving adaptive assessment algorithms, expanding the functional capabilities of the system and conducting long-term studies of its effectiveness in training IT specialists.

The results of the study can be used to modernize existing systems for assessing programming knowledge, develop new educational platforms, and improve curricula for training specialists in the field of information technology.

KEYWORDS: JAVA, ADAPTIVE ASSESSMENT, TESTING, ALGORITHMS, PROGRAMMING, ONLINE PLATFORM, PRACTICAL SKILLS, PERSONALIZATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АСО – адаптивна система оцінювання

ЗВО – заклад вищої освіти

ІТ – інформаційні технології

ООП – об'єктно-орієнтоване програмування

ПЗ – програмне забезпечення

API – Application Programming Interface (прикладний програмний інтерфейс)

IDE – Integrated Development Environment (інтегроване середовище розробки)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

JVM – Java Virtual Machine (віртуальна машина Java)

MVC – Model-View-Controller (архітектурний шаблон проектування)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

REST – Representational State Transfer (архітектурний стиль для розподілених систем)

SQL – Structured Query Language (мова структурованих запитів)

UML – Unified Modeling Language (уніфікована мова моделювання)

XML – eXtensible Markup Language (розширювана мова розмітки)

ВСТУП

Актуальність теми дослідження зумовлена стрімким розвитком інформаційних технологій та зростаючим попитом на висококваліфікованих фахівців у галузі програмування. Java є однією з найпопулярніших мов програмування, яка широко використовується для розробки різноманітних програмних систем, від мобільних додатків до корпоративних рішень. Ефективна підготовка спеціалістів з програмування на Java вимагає наявності якісних методів оцінювання знань та навичок студентів. Традиційні підходи до перевірки знань часто не встигають за швидкими змінами в технологіях та не завжди відповідають практичним потребам індустрії. Тому розробка сучасної системи оцінювання знань з програмування на Java, яка враховує актуальні тенденції та вимоги ринку, є актуальною науковою та практичною проблемою.

Метою магістерської кваліфікаційної роботи є розробка та впровадження адаптивних алгоритмів перевірки знань, які сприятимуть ефективному вивченню мови програмування Java, враховуючи індивідуальні особливості студентів та їхній рівень підготовки.

Для досягнення поставленої мети були визначені наступні завдання дослідження:

1. Провести аналіз існуючих методів і технологій перевірки знань у навчанні програмуванню.

2. Розробити адаптивний алгоритм перевірки знань, що базується на принципах індивідуалізації навчання.

3. Створити прототип програмного забезпечення для реалізації адаптивних алгоритмів перевірки знань у контексті навчання Java.

4. Провести експериментальне дослідження ефективності розроблених алгоритмів на групі студентів.

5. Оцінити результати експерименту та визначити вплив адаптивних алгоритмів на успішність студентів у вивченні мови програмування Java.

Об'єктом дослідження є процес навчання мови програмування Java, зокрема, методи та технології, що використовуються для перевірки знань студентів у цій галузі.

Предметом дослідження є адаптивні алгоритми перевірки знань, які дозволяють автоматично налаштовувати рівень складності завдань відповідно до знань і навичок студентів, а також їхньої прогресії у вивченні Java.

У дослідженні використано наступні методи: аналіз літературних джерел та існуючих рішень, моделювання архітектури системи, розробка алгоритмів адаптивного тестування, програмна реалізація з використанням мови Java та супутніх технологій, тестування та оцінка ефективності системи.

Наукова новизна одержаних результатів полягає у:

1. Вперше розроблено адаптивний алгоритм оцінювання знань з програмування, який, на відміну від існуючих, динамічно коригує складність завдань на основі комплексного аналізу теоретичних знань та практичних навичок студента.
2. Удосконалено метод оцінки якості програмного коду шляхом інтеграції статичного та динамічного аналізу з урахуванням патернів проектування та стандартів розробки на Java.
3. Дістав подальшого розвитку метод виявлення плагіату в програмному коді через впровадження алгоритму порівняльного аналізу структурних елементів та семантичних зв'язків.

Практичне значення одержаних результатів полягає в можливості впровадження розробленої системи оцінювання знань з програмування на Java в навчальний процес закладів вищої освіти та курсів з підготовки IT-фахівців, що дозволить підвищити якість підготовки спеціалістів та забезпечити їх конкурентоспроможність на ринку праці.

Апробація результатів магістерської роботи. Основні положення та результати дослідження доповідалися та обговорювалися на науково-практичних конференціях та семінарах з питань інформаційних технологій в освіті.

1 ТЕОРЕТИЧНІ ОСНОВИ ОЦІНЮВАННЯ ЗНАНЬ

1.1 Аналіз проблематики методів перевірки знань

Методи перевірки знань при навчанні програмуванню потребують особливих підходів для забезпечення якісної підготовки фахівців. Система оцінювання знань з Java включає перевірку теоретичної бази та практичних навичок написання коду. Розробники навчальних курсів приділяють увагу створенню комплексних методик оцінювання. Сучасні методи перевірки базуються на поєднанні автоматизованого та ручного тестування виконаних завдань. Викладачі застосовують різні інструменти для контролю засвоєння матеріалу студентами. Методики оцінювання охоплюють всі етапи навчального процесу. Підходи до перевірки знань постійно вдосконалюються з розвитком технологій.

Традиційні методи контролю знань часто не відповідають сучасним вимогам до підготовки програмістів. Існуючі системи тестування зосереджуються на перевірці теоретичних знань без врахування практичних навичок. Роботодавці потребують фахівців з досвідом написання реальних програм. Студенти стикаються з проблемою недостатньої практики під час навчання. Методики оцінювання не завжди дозволяють виявити реальний рівень підготовки. Системи контролю знань вимагають доопрацювання для відповідності запитам ринку праці. Модернізація методів перевірки стає необхідною умовою якісної підготовки програмістів.

Автоматизовані системи тестування коду набувають все більшого поширення в навчальному процесі. Розробники освітніх платформ впроваджують нові інструменти для перевірки практичних завдань. Системи автоматичного тестування дозволяють швидко оцінити правильність роботи програм. Студенти отримують миттєвий зворотний зв'язок щодо виконаних завдань [1]. Викладачі використовують результати автоматизованої перевірки для оцінювання.

Автоматизація процесу тестування підвищує ефективність навчання. Системи перевірки коду стають невід'ємною частиною освітнього процесу.

Методи оцінювання знань враховують особливості вивчення мови Java як об'єктно-орієнтованої мови програмування. Перевірка розуміння принципів ООП потребує спеціальних підходів до тестування. Системи контролю оцінюють здатність студентів працювати з класами та об'єктами. Викладачі розробляють завдання для перевірки навичок наслідування та поліморфізму. Методики тестування охоплюють роботу з інтерфейсами та абстрактними класами. Студенти демонструють вміння застосовувати принципи інкапсуляції даних. Контроль знань включає перевірку розуміння модифікаторів доступу та методів класів. Системи перевірки практичних навичок оцінюють здатність студентів працювати з колекціями даних у Java. Завдання включають використання різних типів колекцій для вирішення задач. Методи контролю перевіряють розуміння ієрархії колекцій та їх особливостей. Тестування охоплює навички роботи зі списками, множинами та мапами. Студенти демонструють вміння вибирати оптимальні структури даних. Викладачі оцінюють ефективність реалізованих алгоритмів обробки колекцій. Системи тестування враховують правильність використання ітераторів та обробки елементів.

Методики оцінювання знань включають перевірку навичок обробки виключних ситуацій. Системи контролю тестують правильність обробки помилок у програмах. Завдання потребують реалізації механізмів відловлювання та генерації виключень. Студенти демонструють розуміння ієрархії виключень у Java. Викладачі оцінюють коректність використання блоків try-catch-finally. Методи перевірки охоплюють роботу з перевіряємими та неперевіряємими виключеннями. Тестування виявляє здатність створювати власні класи виключень. Контроль знань з багатопоточного програмування потребує комплексного підходу до оцінювання. Системи перевірки тестують навички створення та управління потоками. Завдання включають синхронізацію доступу до спільних ресурсів. Методики оцінювання перевіряють розуміння проблем багатопоточності [2, с. 5]. Викладачі контролюють правильність використання моніторів та блокувань. Студенти демонструють

вміння уникати взаємних блокувань потоків. Тестування охоплює роботу з пулами потоків та асинхронними обчисленнями.

Системи оцінювання включають перевірку навичок роботи з файлами та потоками введення-виведення. Методи контролю тестують вміння читати та записувати дані у файли. Завдання потребують реалізації різних режимів доступу до файлів. Викладачі оцінюють коректність обробки помилок введення-виведення. Студенти демонструють навички серіалізації об'єктів. Тестування охоплює роботу з бінарними та текстовими файлами. Перевірка включає використання буферизованих потоків даних. Методики перевірки знань оцінюють навички роботи з базами даних через JDBC. Системи контролю тестують вміння створювати підключення до баз даних. Завдання включають виконання SQL-запитів та обробку результатів. Викладачі перевіряють правильність використання prepared statements. Студенти демонструють розуміння транзакцій та їх властивостей. Тестування охоплює навички роботи з метаданими баз даних. Перевірка включає обробку помилок при роботі з СУБД.

Методи контролю знань включають оцінювання розуміння основ функціонального програмування в Java. Системи перевірки тестують навички роботи з лямбда-виразами та функціональними інтерфейсами. Завдання потребують застосування Stream API для обробки колекцій даних. Викладачі оцінюють правильність використання методів map, filter та reduce. Студенти демонструють вміння працювати з Optional для обробки null-значень. Тестування охоплює роботу з посиланнями на методи та конструктори. Перевірка включає використання функцій вищого порядку. Системи оцінювання знань тестують розуміння принципів модульності та роботи з пакетами. Методи контролю перевіряють правильність організації коду в пакети. Завдання включають використання модифікаторів доступу для забезпечення інкапсуляції. Викладачі оцінюють коректність імпорту класів та пакетів. Студенти демонструють навички створення та використання jar-файлів. Тестування охоплює роботу з системою модулів Java. Перевірка включає розуміння залежностей між модулями.

Методики перевірки знань оцінюють навички використання узагальнених типів даних. Системи контролю тестують розуміння принципів generic-програмування. Завдання потребують створення узагальнених класів та методів. Викладачі перевіряють правильність обмеження параметрів типу. Студенти демонструють вміння працювати з wildcards у generics. Тестування охоплює коваріантність та контраваріантність типів. Перевірка включає розуміння стирання типів при компіляції.

Методи оцінювання включають перевірку навичок роботи з анотаціями та рефлексією. Системи контролю тестують створення власних анотацій та їх обробників. Завдання потребують використання рефлексії для аналізу класів під час виконання. Викладачі оцінюють правильність отримання метаданих про класи та методи. Студенти демонструють вміння динамічно створювати об'єкти та викликати методи. Тестування охоплює роботу з параметрами анотацій. Перевірка включає використання рефлексії для модифікації поведінки програми. Системи перевірки знань оцінюють навички роботи з регулярними виразами. Методи контролю тестують вміння створювати шаблони для пошуку та заміни тексту. Завдання включають використання класів Pattern та Matcher. Викладачі перевіряють правильність застосування квантифікаторів та груп. Студенти демонструють розуміння прапорців регулярних виразів. Тестування охоплює роботу з межами слів та рядків. Перевірка включає оптимізацію регулярних виразів. Методики оцінювання знань тестують навички написання unit-тестів. Системи контролю перевіряють вміння використовувати фреймворк JUnit. Завдання потребують створення тестових класів та методів. Викладачі оцінюють покриття коду тестами. Студенти демонструють розуміння життєвого циклу тестів. Тестування охоплює роботу з анотаціями JUnit. Перевірка включає використання mock-об'єктів для модульного тестування. Методи контролю знань передбачають оцінювання навичок роботи з сучасними фреймворками Java [3]. Системи перевірки тестують розуміння принципів впровадження залежностей та інверсії управління. Завдання включають використання Spring Framework для розробки додатків. Викладачі оцінюють правильність конфігурації контексту Spring та

створення компонентів. Студенти демонструють вміння працювати з анотаціями для визначення бінів. Тестування охоплює використання аспектно-орієнтованого програмування. Методики перевірки включають роботу з профілями та властивостями Spring.

Системи оцінювання знань тестують навички розробки веб-застосунків з використанням Spring MVC. Методи контролю перевіряють розуміння патерну MVC та його реалізації. Завдання потребують створення контролерів та обробки HTTP-запитів. Викладачі оцінюють правильність маршрутизації та валідації даних форм. Студенти демонструють вміння працювати з представленнями та шаблонізаторами. Тестування охоплює обробку сесій та файлів. Перевірка включає реалізацію REST API. Методики перевірки знань оцінюють навички роботи з системами збірки проектів. Системи контролю тестують вміння використовувати Maven та Gradle. Завдання включають налаштування залежностей та плагінів. Викладачі перевіряють правильність створення життєвого циклу збірки. Студенти демонструють розуміння структури проекту та управління ресурсами. Тестування охоплює роботу з профілями збірки. Перевірка включає створення власних плагінів.

Методи оцінювання включають перевірку навичок роботи з системами контролю версій. Системи контролю тестують вміння використовувати Git для управління кодом. Завдання потребують створення та об'єднання гілок розробки. Викладачі оцінюють правильність вирішення конфліктів злиття. Студенти демонструють навички роботи з віддаленими репозиторіями. Тестування охоплює використання Git Flow для організації розробки. Перевірка включає роботу з тегами та релізами. Системи перевірки знань оцінюють навички розгортання Java-додатків. Методи контролю тестують вміння налаштовувати середовище виконання. Завдання включають розгортання додатків на серверах додатків. Викладачі перевіряють правильність конфігурації JVM та системних ресурсів. Студенти демонструють розуміння процесу створення контейнерів Docker [4, с. 61-68]. Тестування охоплює роботу з оркестрацією контейнерів. Перевірка включає налаштування безперервної інтеграції. Методики оцінювання знань тестують

навички оптимізації продуктивності додатків. Системи контролю перевіряють вміння профілювати та аналізувати код. Завдання потребують виявлення та усунення витоків пам'яті. Викладачі оцінюють правильність налаштування збирача сміття. Студенти демонструють навички оптимізації SQL-запитів. Тестування охоплює використання кешування даних. Перевірка включає розподілення навантаження між компонентами системи.

1.2 Причини необхідності вдосконалення існуючих підходів

Стрімкий розвиток галузі програмування створює нові вимоги до методів навчання Java. Традиційні підходи до перевірки знань не встигають адаптуватися під сучасні технології розробки програмного забезпечення. Методики оцінювання потребують постійного оновлення для відповідності потребам ринку праці. Роботодавці щороку підвищують вимоги до технічних навичок випускників навчальних закладів. Студенти прагнуть отримувати практичний досвід розробки під час навчання. Існуючі системи контролю знань часто зосереджені на теоретичній підготовці. Методи перевірки практичних навичок потребують суттєвого вдосконалення.

Обсяг навчального матеріалу з Java постійно збільшується через появу нових версій мови та фреймворків. Викладачі змушені регулярно оновлювати програми курсів для охоплення актуальних технологій. Системи оцінювання мають враховувати знання нових можливостей мови програмування. Методи контролю повинні перевіряти навички роботи з сучасними інструментами розробки. Студентам необхідно демонструвати розуміння останніх тенденцій у програмуванні. Існуючі підходи до тестування не завжди охоплюють нові технології. Методики перевірки знань потребують розширення для покриття всіх тем. Роботодавці очікують від випускників навичок командної розробки програмного забезпечення. Традиційні методи оцінювання зазвичай перевіряють індивідуальні знання студентів. Системи контролю мають включати завдання для роботи в команді [5, с. 61-68]. Студенти повинні демонструвати навички використання систем контролю версій. Методики перевірки мають оцінювати

вміння працювати над спільним кодом. Існуючі підходи рідко враховують командну взаємодію при розробці. Методи тестування потребують доповнення груповими проектами.

Розробка програмного забезпечення вимагає навичок роботи з великими проектами. Традиційні методи перевірки базуються на невеликих навчальних прикладах. Системи оцінювання мають включати роботу над масштабними додатками. Студенти повинні розуміти архітектуру складних програмних систем. Методики контролю мають перевіряти навички проектування великих додатків. Існуючі підходи зосереджені на окремих компонентах програм. Методи тестування потребують розширення для оцінки системного мислення.

Сучасна розробка програмного забезпечення вимагає знання допоміжних технологій та інструментів. Традиційні методи перевірки концентруються лише на мові Java. Системи оцінювання мають охоплювати весь технологічний стек розробки. Студенти повинні демонструвати навички роботи з базами даних та фреймворками. Методики контролю мають перевіряти знання систем збірки та розгортання. Існуючі підходи не враховують міждисциплінарний характер розробки. Методи тестування потребують розширення для перевірки суміжних технологій. Якісна розробка програмного забезпечення потребує глибокого розуміння принципів безпеки. Традиційні методи оцінювання приділяють недостатньо уваги питанням захисту даних. Системи контролю мають перевіряти навички створення безпечного коду. Студентам необхідно демонструвати знання криптографічних методів та протоколів. Методики тестування повинні охоплювати різні сценарії порушення безпеки [6]. Викладачі мають оцінювати вміння виявляти та усувати вразливості. Існуючі підходи до перевірки знань потребують посилення безпекової складової.

Сучасні вимоги до програмного забезпечення включають здатність обробляти великі обсяги даних. Традиційні методи перевірки не враховують особливості роботи з Big Data. Системи оцінювання мають тестувати навички масштабування додатків. Студенти повинні розуміти принципи розподілених обчислень. Методики контролю мають перевіряти знання технологій обробки

великих даних. Викладачі оцінюють вміння оптимізувати продуктивність систем. Існуючі підходи потребують розширення для охоплення сучасних методів обробки даних.

Розробка мобільних додатків стає невід'ємною частиною програмування на Java. Традиційні методи оцінювання зосереджені на десктопних та серверних додатках. Системи контролю мають включати перевірку навичок мобільної розробки. Студенти повинні демонструвати знання Android SDK та життєвого циклу додатків. Методики тестування мають охоплювати особливості мобільних платформ. Викладачі оцінюють вміння створювати адаптивні інтерфейси. Існуючі підходи потребують доповнення мобільними технологіями.

Мікросервісна архітектура вимагає нових підходів до розробки програмного забезпечення. Традиційні методи перевірки орієнтовані на монолітні додатки. Системи оцінювання мають тестувати навички створення та взаємодії мікросервісів. Студенти повинні розуміти принципи проектування розподілених систем. Методики контролю мають перевіряти знання контейнеризації та оркестрації. Викладачі оцінюють вміння забезпечувати відмовостійкість сервісів. Існуючі підходи не відповідають вимогам мікросервісної архітектури. Хмарні технології змінюють способи розгортання та масштабування додатків. Традиційні методи оцінювання не враховують особливості хмарної інфраструктури. Системи контролю мають перевіряти навички роботи з хмарними платформами. Студенти повинні демонструвати знання моделей розгортання та обслуговування. Методики тестування мають охоплювати управління ресурсами в хмарі. Викладачі оцінюють вміння забезпечувати високу доступність сервісів. Існуючі підходи потребують адаптації під хмарні технології.

Автоматизоване тестування програмного забезпечення потребує глибокого розуміння принципів якості коду. Традиційні методи оцінювання не приділяють достатньо уваги практиці написання тестів. Системи контролю мають перевіряти навички створення модульних та інтеграційних тестів. Студенти повинні демонструвати знання методологій тестування та фреймворків. Методики перевірки мають охоплювати різні види тестування програмного забезпечення.

Викладачі оцінюють вміння досягати високого покриття коду тестами. Існуючі підходи потребують посилення тестової складової навчання.

Розробка програмного забезпечення вимагає розуміння принципів управління проектами. Традиційні методи перевірки знань не враховують організаційні навички студентів. Системи оцінювання мають тестувати здатність планувати та відстежувати виконання завдань. Студенти повинні демонструвати знання методологій розробки програмного забезпечення. Методики контролю мають перевіряти навички роботи з системами управління проектами. Викладачі оцінюють вміння оцінювати та розподіляти ресурси. Існуючі підходи не охоплюють проектний менеджмент. Сучасна розробка вимагає навичок аналізу та візуалізації даних. Традиційні методи оцінювання не включають роботу з інструментами аналітики. Системи контролю мають перевіряти вміння обробляти та представляти дані. Студенти повинні демонструвати знання бібліотек для створення звітів та графіків. Методики тестування мають охоплювати різні формати представлення інформації. Викладачі оцінюють здатність робити висновки на основі даних. Існуючі підходи потребують розширення аналітичної складової.

Машинне навчання стає невід'ємною частиною сучасної розробки. Традиційні методи перевірки не враховують особливості роботи з моделями машинного навчання. Системи оцінювання мають тестувати навички інтеграції алгоритмів штучного інтелекту. Студенти повинні розуміти принципи навчання моделей та обробки даних. Методики контролю мають перевіряти знання бібліотек машинного навчання. Викладачі оцінюють вміння обирати та налаштовувати моделі. Існуючі підходи не відповідають вимогам штучного інтелекту. Розробка реактивних систем потребує нових підходів до програмування. Традиційні методи оцінювання орієнтовані на синхронні обчислення [7]. Системи контролю мають перевіряти навички асинхронного програмування. Студенти повинні демонструвати знання реактивних патернів та фреймворків. Методики тестування мають охоплювати обробку потоків даних. Викладачі оцінюють вміння

забезпечувати реактивність додатків. Існуючі підходи потребують адаптації під реактивне програмування.

Функціональне програмування набуває все більшого значення в розробці на Java. Традиційні методи оцінювання зосереджені на об'єктно-орієнтованому підході. Системи контролю мають перевіряти навички застосування функціональних концепцій. Студенти повинні демонструвати розуміння чистих функцій та незмінних структур даних. Методики тестування мають охоплювати роботу з функціональними інтерфейсами та потоками. Викладачі оцінюють вміння застосовувати функціональні патерни проектування. Існуючі підходи потребують розширення функціональної парадигми. Розробка програмного забезпечення вимагає знання методів оптимізації продуктивності. Традиційні методи перевірки не приділяють достатньо уваги питанням ефективності коду. Системи оцінювання мають тестувати навички профілювання та налагодження програм. Студенти повинні розуміти принципи управління пам'яттю та багатопоточністю. Методики контролю мають перевіряти вміння виявляти та усувати проблеми продуктивності. Викладачі оцінюють здатність оптимізувати критичні ділянки коду. Існуючі підходи потребують посилення уваги до оптимізації.

Сучасна розробка потребує розуміння методів забезпечення якості коду. Традиційні методи оцінювання не враховують метрики якості програмного забезпечення. Системи контролю мають перевіряти навички написання чистого та підтримуваного коду. Студенти повинні демонструвати знання принципів рефакторингу та проектних патернів. Методики тестування мають охоплювати статичний аналіз коду. Викладачі оцінюють вміння дотримуватися стандартів кодування. Існуючі підходи потребують впровадження метрик якості. Розробка веб-застосунків вимагає знання сучасних фронтенд-технологій. Традиційні методи перевірки зосереджені на серверній частині. Системи оцінювання мають тестувати навички створення користувацьких інтерфейсів. Студенти повинні демонструвати розуміння принципів веб-розробки та фреймворків. Методики контролю мають перевіряти знання JavaScript та веб-компонентів. Викладачі оцінюють вміння

інтегрувати фронтенд із бекендом. Існуючі підходи потребують розширення фронтенд-складової.

Інтеграція різних систем потребує розуміння принципів взаємодії сервісів. Традиційні методи оцінювання не охоплюють особливості інтеграційних рішень. Системи контролю мають перевіряти навички розробки API та обміну даними. Студенти повинні демонструвати знання протоколів та форматів передачі інформації. Методики тестування мають охоплювати різні сценарії інтеграції. Викладачі оцінюють вміння забезпечувати надійність взаємодії. Існуючі підходи потребують посилення інтеграційної складової.

Впровадження принципів DevOps змінює підходи до розробки та розгортання. Традиційні методи оцінювання не враховують автоматизацію процесів розробки. Системи контролю мають перевіряти навички налаштування конвеєрів безперервної інтеграції. Студенти повинні демонструвати знання інструментів автоматизації та моніторингу. Методики тестування мають охоплювати процеси розгортання та масштабування. Викладачі оцінюють вміння забезпечувати стабільність середовища розробки. Існуючі підходи потребують впровадження DevOps практик. Розробка додатків потребує розуміння принципів документування коду. Традиційні методи перевірки не приділяють достатньо уваги якості документації. Системи оцінювання мають тестувати навички створення технічної документації. Студенти повинні демонструвати вміння писати зрозумілі коментарі та опис API [8, с. 82-92]. Методики контролю мають перевіряти повноту та актуальність документації. Викладачі оцінюють здатність створювати корисні інструкції для користувачів. Існуючі підходи потребують посилення документаційної складової.

Сучасна розробка вимагає розуміння принципів управління конфігураціями. Традиційні методи оцінювання не охоплюють налаштування середовища розробки. Системи контролю мають перевіряти навички управління залежностями та версіями. Студенти повинні демонструвати знання систем контейнеризації та віртуалізації. Методики тестування мають охоплювати різні середовища виконання. Викладачі оцінюють вміння забезпечувати відтворюваність

середовища. Існуючі підходи потребують впровадження управління конфігураціями.

Робота з успадкованими системами вимагає особливих навичок підтримки коду. Традиційні методи перевірки зосереджені на розробці нових систем. Системи оцінювання мають тестувати вміння працювати з застарілим кодом. Студенти повинні демонструвати навички зворотного проектування та рефакторингу. Методики контролю мають перевіряти здатність модернізувати існуючі системи. Викладачі оцінюють вміння інтегрувати нові технології в старі системи. Існуючі підходи потребують уваги до підтримки успадкованого коду. Розвиток Інтернету речей створює нові вимоги до розробки програмного забезпечення. Традиційні методи оцінювання не враховують особливості вбудованих систем. Системи контролю мають перевіряти навички розробки для IoT пристроїв. Студенти повинні демонструвати розуміння протоколів обміну даними між пристроями. Методики тестування мають охоплювати роботу з сенсорами та актуаторами. Викладачі оцінюють вміння забезпечувати взаємодію різних компонентів IoT систем. Існуючі підходи потребують адаптації під Інтернет речей.

1.3 Огляд існуючих алгоритмів перевірки знань

Сучасні системи автоматизованої перевірки знань з Java відіграють ключову роль у процесі навчання програмуванню. Розробники освітніх платформ створюють спеціалізовані алгоритми для оцінки правильності написаних програм. Системи перевірки використовують різні підходи до аналізу структури та функціональності коду. Алгоритми тестування забезпечують швидку оцінку виконання практичних завдань. Методи автоматичної перевірки дозволяють обробляти велику кількість робіт одночасно та надавати студентам миттєвий зворотний зв'язок.

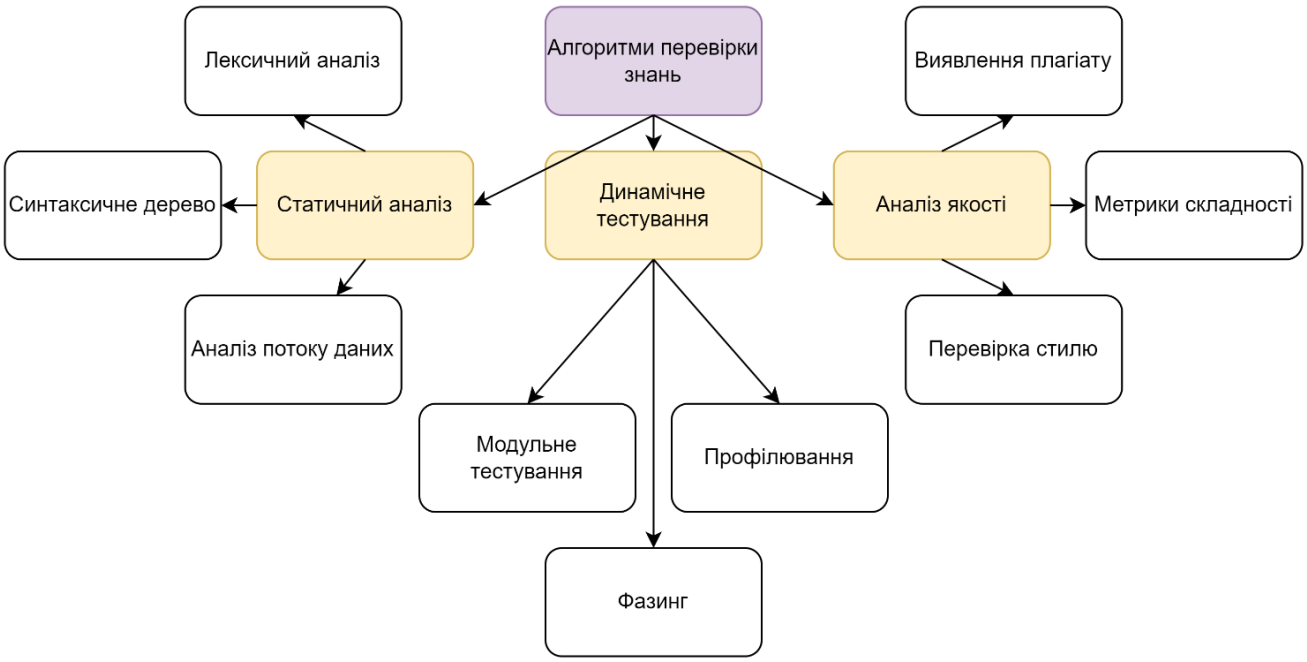


Рис. 1.1 Класифікація алгоритмів перевірки знань

Статичний аналіз коду становить фундаментальну частину процесу перевірки знань. Алгоритм лексичного аналізу здійснює розбиття вихідного коду на токени та виявляє синтаксичні помилки з лінійною складністю. Побудова абстрактного синтаксичного дерева створює структуроване представлення програми та дозволяє аналізувати семантичні зв'язки. Алгоритм аналізу потоку даних відстежує ініціалізацію та використання змінних. Статичні методи перевірки допомагають виявити проблеми в коді ще до його виконання.

Виявлення плагіату є вагомим аспектом оцінювання самостійності виконання завдань. Алгоритм відбитків обчислює хеш-значення для фрагментів коду та порівнює їх між різними роботами. Метод Winnowing застосовує ковзне вікно для вибору репрезентативних відбитків та зменшення помилкових спрацьовувань. Системи антиплагіату враховують можливість перейменування змінних та реструктуризації коду. Оцінка схожості базується на співвідношенні кількості співпадаючих та загальних відбитків.

Динамічне тестування програм забезпечує перевірку функціональності коду під час виконання. Алгоритми модульного тестування оцінюють роботу окремих компонентів через набори тестових випадків. Метод фазингу генерує випадкові

вхідні дані для виявлення помилок та вразливостей. Системи тестування вимірюють покриття коду та ефективність обробки граничних випадків. Важливим аспектом є перевірка обробки виключних ситуацій [9, с. 46-50].

Аналіз продуктивності програм здійснюється через комплексні алгоритми профілювання. Вимірювання часу виконання окремих частин програми дозволяє виявити вузькі місця. Алгоритми відстеження пам'яті контролюють процеси її виділення та звільнення. Оцінка ефективності базується на порівнянні базових та оптимізованих показників роботи програми. Системи профілювання надають детальну статистику використання ресурсів.

Якість написання коду оцінюється через систему метрик складності. Цикломатична складність за McCabe враховує структуру графу потоку керування програми. Метрики Холстеда аналізують лексичну складність коду через кількість операторів та операндів. Алгоритми перевірки стилю контролюють відповідність стандартам кодування. Оцінка якості коду допомагає формувати правильні практики програмування.

Безпека програмного коду перевіряється спеціалізованими алгоритмами аналізу. Методи статичного аналізу виявляють потенційні вразливості у структурі програми. Динамічне тестування перевіряє стійкість до атак під час виконання. Алгоритми оцінюють коректність реалізації механізмів автентифікації та авторизації. Системи безпеки контролюють роботу з конфіденційними даними.

Алгоритми оцінки архітектури програм аналізують загальну структуру додатків. Методи перевірки оцінюють взаємозв'язки між компонентами та їх зв'язність. Системи аналізу контролюють дотримання принципів проектування та патернів. Алгоритми виявляють проблеми масштабованості та підтримуваності коду. Оцінка архітектури сприяє розвитку навичок системного проектування.

Тестування взаємодії з базами даних здійснюється через спеціальні алгоритми перевірки. Методи аналізу оцінюють коректність SQL-запитів та структури бази даних. Системи тестування перевіряють обробку транзакцій та цілісність даних. Алгоритми контролюють механізми кешування та оптимізації доступу. Важливим аспектом є перевірка обробки конкурентних запитів.

Алгоритми тестування користувацьких інтерфейсів оцінюють зручність програм. Методи перевірки аналізують відповідність інтерфейсів стандартам проектування. Системи тестування оцінюють реакцію програм на дії користувачів. Алгоритми контролюють обробку помилок та інформативність повідомлень. Оцінка UI/UX допомагає розвивати навички проектування інтерфейсів.

Перевірка багатопоточності програм вимагає спеціалізованих алгоритмів аналізу. Методи тестування виявляють проблеми синхронізації та гонки даних. Системи перевірки оцінюють ефективність розподілу задач між потоками. Алгоритми контролюють роботу пулів потоків та асинхронних операцій. Тестування багатопоточності забезпечує надійність паралельних обчислень [10].

Алгоритми перевірки роботи з великими даними оцінюють масштабованість рішень. Методи тестування контролюють ефективність розподіленої обробки інформації. Системи аналізу перевіряють механізми агрегації та аналітики даних. Алгоритми оцінюють надійність зберігання великих обсягів інформації. Тестування Big Data формує навички роботи з масштабними системами.

Оцінка інтеграційних аспектів здійснюється через комплексні алгоритми тестування. Методи перевірки контролюють взаємодію між різними компонентами системи. Алгоритми аналізують коректність обміну даними та сумісність інтерфейсів. Системи тестування оцінюють стабільність інтеграційних процесів. Перевірка інтеграції забезпечує цілісність складних систем.

Алгоритми машинного навчання впроваджуються для вдосконалення процесу перевірки знань. Нейронні мережі застосовуються для прогнозування потенційних помилок у коді. Методи глибокого навчання допомагають аналізувати семантичну подібність програм. Системи штучного інтелекту забезпечують адаптивний підбір завдань. Використання ML підвищує точність та об'єктивність оцінювання.

Перспективи розвитку алгоритмів перевірки знань пов'язані з впровадженням нових технологій. Методи аналізу стають більш інтелектуальними та адаптивними. Системи тестування розвиваються в напрямку автоматизації та персоналізації навчання. Алгоритми оцінювання охоплюють все ширший спектр технологій програмування. Постійне вдосконалення методів контролю сприяє підвищенню якості освіти.

2 АНАЛІЗ МЕТОДІВ ТА АЛГОРИТМІВ ОЦІНЮВАННЯ ЗНАНЬ В ОНЛАЙН-ПЛАТФОРМІ JAVA

2.1 Аналіз існуючих математичних методів, моделей та алгоритмів для адаптивної оцінки знань

Математичні методи оцінювання знань становлять основу сучасних освітніх платформ для вивчення програмування. Розробка адаптивних алгоритмів перевірки потребує глибокого розуміння теорії тестування та психометрії. Методи класичної теорії тестів забезпечують базовий рівень оцінювання через розрахунок первинних балів. Сучасні підходи використовують складні математичні моделі для врахування різних параметрів тестових завдань. Системи адаптивного тестування застосовують методи теорії відповідей на завдання (IRT). Моделі IRT дозволяють оцінити латентні параметри знань студентів. Математичний апарат забезпечує об'єктивність та надійність результатів оцінювання.

Базові методи класичної теорії тестів оперують такими параметрами як складність завдань та розподільна здатність. Індекс складності розраховується як відношення кількості правильних відповідей до загальної кількості відповідей. Коефіцієнт розподільної здатності показує здатність завдання розрізняти сильних та слабких студентів. Методи розрахунку надійності тесту базуються на визначенні внутрішньої узгодженості завдань. Статистичні методи дозволяють виявити неякісні завдання для подальшого покращення тесту. Основні показники якості тестових завдань наведені в таблиці 2.1.

Таблиця 2.1

Показники якості тестових завдань

Показник	**Формула розрахунку**	**Діапазон значень**	**Рекомендовані значення**
Індекс складності	$P = \frac{R}{N}$	0-1	0.3-0.8

Продовження таблиці 2.1

Показники якості тестових завдань

Розподільна здатність	$D = \frac{Rh - Rl}{0.5N}$	-1-1	>0.3
Надійність	$\alpha = \left(\frac{k}{k-1}\right)\left(1 - \frac{\sum \sigma i^2}{\sigma x^2}\right)$	0-1	>0.7
Валідність	$V = \sqrt{\alpha}$	0-1	>0.8

Індекс складності (P) розраховується як відношення кількості правильних відповідей (R) до загальної кількості відповідей (N). Його значення може варіюватися від 0 до 1, а рекомендований діапазон - від 0.3 до 0.8. Цей показник відображає, наскільки складним є завдання для тестованих.

Розподільна здатність (D) визначається за формулою $\frac{Rh-Rl}{0.5N}$, де Rh - кількість правильних відповідей у групі з високими балами, а Rl - у групі з низькими балами. Вона може набувати значень від -1 до 1, а бажане значення - більше 0.3. Цей показник характеризує здатність завдання розрізняти студентів з різним рівнем підготовки.

Надійність (α) розраховується за формулою, що враховує кількість завдань у тесті (k), суму дисперсій балів за кожне завдання ($\sum \sigma i^2$) та дисперсію індивідуальних балів за тест (σx^2). Надійність може варіюватися від 0 до 1, а рекомендоване значення - вище 0.7. Вона відображає внутрішню узгодженість та стабільність результатів тесту.

Валідність (V) визначається як квадратний корінь з надійності ($\sqrt{\alpha}$) і може набувати значень від 0 до 1. Бажане значення - більше 0.8. Валідність показує, наскільки тест дійсно вимірює те, що має на меті.

Теорія відповідей на завдання використовує більш складні математичні моделі для оцінювання. Однопараметрична модель Раша враховує лише складність завдань та рівень підготовки студентів. Двопараметрична модель додатково включає параметр розподільної здатності завдань. Трипараметрична модель

розширює попередні моделі параметром вгадування правильної відповіді. Методи IRT дозволяють побудувати характеристичні криві завдань та тесту. Математичний апарат IRT забезпечує інваріантність оцінок відносно набору завдань. Основні моделі IRT представлені на рисунку 2.1.

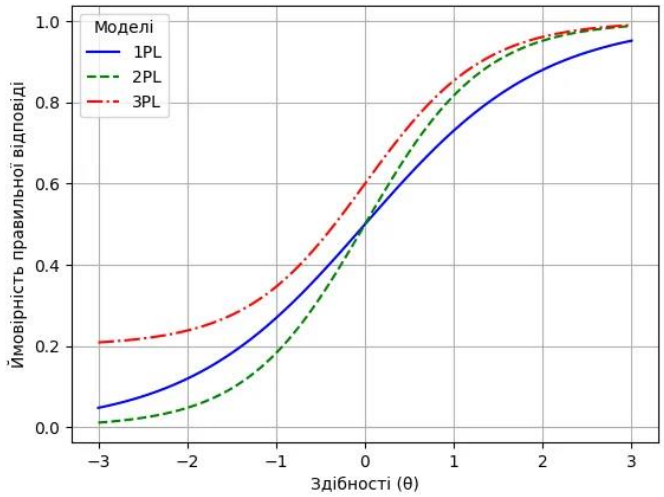


Рис. 2.1 Характеристичні криві моделей IRT

Адаптивне тестування базується на методах послідовного вибору оптимальних завдань. Алгоритми вибору використовують інформаційні функції для максимізації точності оцінювання. Оцінювання методом Баєса дозволяє уточнювати рівень підготовки після кожної відповіді. Системи адаптивного тестування застосовують різні критерії зупинки процедури. Математичні моделі забезпечують мінімізацію кількості завдань при заданій точності. Основні характеристики методів адаптивного тестування наведені в таблиці 2.2.

Таблиця 2.2

Характеристики методів адаптивного тестування

Метод Баєса	**Критерій вибору завдань**	**Оцінювання рівня підготовки**	**Критерій зупинки**
Максимум інформації	Максимум інформаційної функції	Метод максимальної правдоподібності	Досягнення заданої точності

Продовження таблиці 2.2

Характеристики методів адаптивного тестування

Метод Баєса	**Критерій вибору завдань**	**Оцінювання рівня підготовки**	**Критерій зупинки**
метод Баєса	Мінімум очікуваної помилки	Оцінювання методом Баєса	Стабілізація оцінки
Стратифікаційний	Відповідність рівню підготовки	Покроковий перерахунок	Фіксована довжина

Нейронні мережі забезпечують можливість врахування додаткових факторів при оцінюванні. Архітектури глибокого навчання дозволяють виявляти приховані закономірності у відповідях студентів. Методи машинного навчання застосовуються для прогнозування успішності виконання завдань. Алгоритми кластеризації допомагають виявити групи студентів зі схожими патернами відповідей. Нейромережеві моделі здатні адаптуватися до індивідуальних особливостей навчання. Системи на основі штучного інтелекту забезпечують персоналізацію процесу оцінювання. Методи глибокого навчання дозволяють автоматизувати аналіз складних завдань з програмування

Оцінювання практичних навичок програмування потребує спеціальних математичних підходів. Методи статичного аналізу коду використовують метрики складності та якості програм. Алгоритми перевірки включають розрахунок цикломатичної складності та зв'язності компонентів. Системи оцінювання враховують різні характеристики написаного коду [11, с. 186]. Метрики об'єктно-орієнтованого проектування дозволяють оцінити якість архітектури програм. Методи аналізу залежностей виявляють проблеми в структурі коду. Математичні моделі забезпечують комплексну оцінку практичних завдань.

Методи динамічного аналізу базуються на дослідженні поведінки програм під час виконання. Алгоритми тестування використовують різні набори вхідних даних для перевірки функціональності. Системи оцінювання збирають метрики

продуктивності та використання ресурсів. Математичні моделі дозволяють прогнозувати поведінку програм на великих наборах даних. Методи профілювання виявляють проблемні місця в коді. Алгоритми аналізу покриття коду оцінюють повноту тестування. Динамічні характеристики доповнюють статичний аналіз програм.

Оцінювання алгоритмічної складності рішень базується на асимптотичному аналізі. Методи розрахунку часової складності враховують кількість базових операцій. Алгоритми оцінки просторової складності аналізують використання пам'яті. Системи тестування перевіряють ефективність реалізованих алгоритмів. Математичні моделі дозволяють порівнювати різні підходи до вирішення задач. Методи аналізу складності допомагають виявити неоптимальні рішення. Асимптотичні оцінки забезпечують об'єктивне порівняння алгоритмів.

Методи оцінки стилю написання коду використовують формальні метрики якості. Алгоритми аналізу перевіряють відповідність загальноприйнятим стандартам оформлення. Системи оцінювання враховують читабельність та структурованість програм. Математичні моделі дозволяють кількісно оцінити якість написання коду. Методи аналізу коментарів перевіряють документованість програм. Алгоритми перевірки найменувань оцінюють зрозумілість ідентифікаторів. Формальні метрики забезпечують об'єктивність оцінки стилю.

Оцінювання паралельних програм вимагає специфічних математичних підходів. Методи аналізу багатопоточності враховують ефективність розподілу обчислень між потоками. Алгоритми оцінки перевіряють коректність синхронізації доступу до спільних ресурсів. Системи тестування виявляють проблеми гонки даних та взаємних блокувань. Математичні моделі дозволяють прогнозувати масштабованість паралельних програм. Методи профілювання оцінюють накладні витрати на синхронізацію. Алгоритми аналізу забезпечують оптимізацію паралельних обчислень.

Методи оцінки безпеки програм базуються на формальній верифікації коду. Алгоритми статичного аналізу виявляють типові вразливості та потенційні загрози. Системи тестування перевіряють стійкість до різних видів атак. Математичні

моделі дозволяють оцінити рівень захищеності програм. Методи аналізу конфіденційності контролюють витоки чутливих даних. Алгоритми перевірки оцінюють надійність механізмів автентифікації. Формальні методи забезпечують комплексний аналіз безпеки.

Оцінювання веб-застосунків включає аналіз клієнтської та серверної частин. Методи тестування перевіряють продуктивність та масштабованість систем. Алгоритми оцінки досліджують поведінку додатків при різних навантаженнях. Системи аналізу контролюють швидкодію та використання ресурсів. Математичні моделі дозволяють прогнозувати характеристики веб-систем. Методи профілювання виявляють вузькі місця в архітектурі. Комплексний аналіз забезпечує якість веб-застосунків. Адаптивні методи оцінювання враховують індивідуальні особливості студентів. Алгоритми аналізу будують профілі навчання на основі історії відповідей. Системи тестування адаптують складність завдань до рівня підготовки. Математичні моделі прогнозують оптимальну траєкторію навчання [12]. Методи кластеризації виявляють групи студентів зі схожими патернами. Алгоритми рекомендацій пропонують персоналізовані завдання. Адаптивний підхід підвищує ефективність навчання.

Оцінювання командної роботи використовує методи аналізу взаємодії розробників. Алгоритми дослідження репозиторіїв виявляють патерни спільної роботи. Системи контролю версій надають дані про внесок кожного учасника. Математичні моделі оцінюють ефективність командної взаємодії. Методи аналізу комунікацій досліджують процеси обміну знаннями. Алгоритми оцінки якості коду враховують колективну розробку. Комплексний аналіз забезпечує об'єктивність оцінювання команд.

Методи аналізу великих даних оцінюють здатність обробляти масштабні набори інформації. Алгоритми тестування перевіряють ефективність розподіленої обробки даних. Системи оцінювання контролюють продуктивність операцій з великими обсягами. Математичні моделі прогнозують поведінку програм при зростанні даних. Методи профілювання виявляють обмеження масштабованості

рішень. Алгоритми аналізу оцінюють ефективність механізмів агрегації. Тестування Big Data систем потребує специфічних підходів.

Оцінювання машинного навчання включає аналіз якості моделей та процесу навчання. Методи перевірки досліджують точність та узагальнюючу здатність моделей. Алгоритми тестування контролюють стійкість до різних наборів даних. Системи оцінювання перевіряють швидкодію навчання та прогнозування. Математичні метрики дозволяють порівнювати різні моделі. Методи валідації забезпечують об'єктивність оцінки. Комплексний аналіз охоплює всі етапи машинного навчання.

Методи оцінки мікросервісної архітектури аналізують взаємодію компонентів. Алгоритми тестування перевіряють надійність комунікації між сервісами. Системи моніторингу контролюють стан розподіленої інфраструктури. Математичні моделі прогнозують поведінку при відмовах компонентів. Методи аналізу оцінюють ефективність маршрутизації запитів. Алгоритми перевірки досліджують масштабованість архітектури. Комплексне тестування забезпечує якість мікросервісних систем.

Оцінювання хмарних додатків враховує особливості розгортання та експлуатації. Методи тестування перевіряють автоматичне масштабування та відмовостійкість. Алгоритми аналізу контролюють ефективність використання ресурсів. Системи моніторингу досліджують продуктивність в хмарному середовищі. Математичні моделі прогнозують витрати на обслуговування. Методи перевірки оцінюють надійність резервного копіювання. Комплексний підхід забезпечує якість хмарних рішень.

Методи оцінки контейнеризації аналізують ефективність віртуалізації додатків. Алгоритми тестування перевіряють портативність контейнерних рішень. Системи моніторингу контролюють використання системних ресурсів. Математичні моделі прогнозують масштабованість контейнерної інфраструктури. Методи аналізу оцінюють ефективність оркестрації. Алгоритми перевірки досліджують мережеву взаємодію. Комплексне тестування охоплює всі аспекти контейнеризації.

Оцінювання процесів безперервної інтеграції базується на аналізі автоматизації розробки. Методи тестування перевіряють надійність конвеєрів збірки та розгортання. Алгоритми аналізу контролюють якість автоматизованих тестів. Системи моніторингу досліджують стабільність процесів CI/CD. Математичні моделі прогнозують ефективність автоматизації. Методи оцінки перевіряють швидкість внесення змін. Комплексний аналіз забезпечує якість процесів розробки.

Методи оцінки моніторингу аналізують повноту спостереження за системами. Алгоритми перевірки контролюють точність збору метрик та логів. Системи аналізу досліджують ефективність виявлення проблем. Математичні моделі прогнозують аномальну поведінку систем. Методи оцінки перевіряють якість візуалізації даних. Алгоритми аналізу контролюють швидкість реагування на інциденти. Комплексний моніторинг забезпечує надійність експлуатації.

Оцінювання відмовостійкості систем базується на аналізі поведінки при збоях. Методи тестування перевіряють механізми відновлення після відмов. Алгоритми аналізу контролюють деградацію функціональності. Системи моніторингу досліджують каскадні відмови компонентів. Математичні моделі прогнозують надійність архітектури. Методи оцінки перевіряють ефективність резервування. Комплексне тестування забезпечує стійкість систем.

Методи аналізу управління конфігураціями оцінюють налаштування середовища. Алгоритми перевірки контролюють версії залежностей та конфігурацій. Системи тестування досліджують відтворюваність середовища. Математичні моделі прогнозують сумісність компонентів. Методи оцінки перевіряють процеси оновлення налаштувань. Алгоритми аналізу контролюють цілісність конфігурацій. Комплексний підхід забезпечує надійність інфраструктури.

Оцінювання мобільної розробки враховує особливості різних платформ та пристроїв. Методи тестування перевіряють адаптивність інтерфейсів. Алгоритми аналізу контролюють енергоефективність додатків. Системи моніторингу досліджують продуктивність на пристроях. Математичні моделі прогнозують

поведінку в різних умовах. Методи оцінки перевіряють безпеку мобільних додатків. Комплексне тестування забезпечує якість програмного забезпечення.

Методи оцінки розробки додатків для Інтернету речей аналізують специфіку вбудованих систем. Алгоритми тестування перевіряють надійність протоколів обміну даними між пристроями. Системи аналізу контролюють енергоефективність програмного забезпечення. Математичні моделі прогнозують поведінку мережі пристроїв. Методи оцінки перевіряють безпеку комунікації в IoT системах. Алгоритми аналізу досліджують обробку даних з сенсорів. Комплексне тестування охоплює всі компоненти IoT рішень.

Оцінювання API та інтеграційних рішень базується на аналізі взаємодії систем. Методи тестування перевіряють стабільність та сумісність інтерфейсів. Алгоритми аналізу контролюють ефективність обміну даними. Системи моніторингу досліджують навантаження на API. Математичні моделі прогнозують масштабованість інтеграцій. Методи оцінки перевіряють безпеку передачі даних. Комплексний підхід забезпечує надійність взаємодії.

Методи оцінки графічних інтерфейсів аналізують зручність використання додатків. Алгоритми тестування перевіряють responsiveness та інтерактивність UI. Системи аналізу контролюють продуктивність рендерингу компонентів. Математичні моделі прогнозують поведінку інтерфейсу при навантаженні. Методи оцінки перевіряють доступність елементів управління. Алгоритми аналізу досліджують анімації та візуальні ефекти. Комплексне тестування забезпечує якість користувацького досвіду.

Оцінювання компонентної архітектури базується на аналізі модульності систем. Методи тестування перевіряють взаємозамінність компонентів. Алгоритми аналізу контролюють зв'язність та зчеплення модулів. Системи моніторингу досліджують життєвий цикл компонентів. Математичні моделі прогнозують гнучкість архітектури. Методи оцінки перевіряють можливості розширення системи. Комплексний аналіз забезпечує якість архітектурних рішень.

Методи оцінки обробки медіа-контенту аналізують роботу з мультимедійними даними. Алгоритми тестування перевіряють якість кодування та

декодування. Системи аналізу контролюють синхронізацію аудіо та відео потоків. Математичні моделі прогнозують навантаження при стримінгу. Методи оцінки перевіряють обробку метаданих медіа-файлів. Алгоритми аналізу досліджують конвертацію форматів. Комплексне тестування забезпечує якість мультимедійних систем.

Методи оцінки математичного моделювання аналізують реалізацію обчислювальних алгоритмів. Системи тестування перевіряють точність та стабільність чисельних методів. Алгоритми аналізу контролюють збіжність ітераційних процесів. Математичні моделі прогнозують поведінку на граничних випадках. Методи оцінки перевіряють ефективність матричних операцій. Алгоритми контролю досліджують обробку особливих точок. Комплексне тестування забезпечує надійність обчислень.

Оцінювання розподілених обчислень базується на аналізі паралельної обробки. Методи тестування перевіряють ефективність розподілу задач між вузлами. Алгоритми аналізу контролюють синхронізацію розподілених компонентів. Системи моніторингу досліджують навантаження на вузли. Математичні моделі прогнозують масштабованість рішень. Методи оцінки перевіряють відмовостійкість кластера. Комплексний підхід забезпечує надійність розподілених систем.

Методи оцінки функціонального програмування аналізують чистоту функцій. Системи тестування перевіряють відсутність побічних ефектів. Алгоритми аналізу контролюють незмінність структур даних. Математичні моделі прогнозують композицію функцій. Методи оцінки перевіряють типобезпеку програм. Алгоритми контролю досліджують рекурсивні процеси. Комплексне тестування забезпечує якість функціонального коду.

Оцінювання реактивного програмування базується на аналізі потоків даних. Методи тестування перевіряють обробку асинхронних подій. Алгоритми аналізу контролюють back-pressure механізми. Системи моніторингу досліджують реактивність додатків. Математичні моделі прогнозують поведінку при

навантаженні. Методи оцінки перевіряють обробку помилок. Комплексний аналіз забезпечує якість реактивних систем.

Методи верифікації формальними методами аналізують коректність програм. Системи тестування перевіряють відповідність специфікаціям. Алгоритми аналізу контролюють інваріанти та передумови. Математичні моделі прогнозують можливі стани системи. Методи оцінки перевіряють досяжність критичних станів. Алгоритми контролю досліджують часові характеристики. Формальна верифікація гарантує надійність програм.

Методи оцінки систем реального часу аналізують часові характеристики програм. Алгоритми тестування перевіряють дотримання часових обмежень та дедлайнів. Системи моніторингу контролюють затримки обробки подій. Математичні моделі прогнозують поведінку при пікових навантаженнях. Методи аналізу досліджують планування задач та пріоритети. Алгоритми оцінки перевіряють детермінованість виконання. Комплексне тестування забезпечує надійність систем реального часу.

Оцінювання вбудованих систем враховує обмеження апаратних ресурсів. Методи тестування перевіряють ефективність використання пам'яті та процесора. Алгоритми аналізу контролюють енергоспоживання програм. Системи моніторингу досліджують взаємодію з периферійними пристроями. Математичні моделі прогнозують поведінку в критичних режимах. Методи оцінки перевіряють надійність програмно-апаратних рішень. Комплексний підхід забезпечує якість вбудованих систем.

Архітектура розробленої системи базується на принципах багаторівневого веб-застосунку та включає такі компоненти:

— Серверна частина реалізована з використанням Spring Framework, що забезпечує надійний та масштабований бекенд.
— Spring Security реалізує механізми автентифікації та авторизації користувачів.
— Hibernate спрощує роботу з базою даних PostgreSQL через об'єктно-реляційне відображення.

— Redis використовується для кешування даних та підвищення продуктивності.

На клієнтській стороні:

— React забезпечує створення динамічного та інтерактивного інтерфейсу
— Redux керує станом застосунку
— Tailwind CSS використовується для стилізації компонентів

Інфраструктура включає:

— Docker для контейнеризації
— Nginx як веб-сервер та балансувальник навантаження
— Jenkins для процесів CI/CD
— Prometheus та Grafana для моніторингу
— ELK Stack для централізованої обробки логів

Оцінювання систем зберігання даних базується на аналізі надійності та продуктивності. Методи тестування перевіряють цілісність та доступність інформації. Алгоритми аналізу контролюють ефективність операцій введення-виведення. Системи моніторингу досліджують латентність доступу до даних. Математичні моделі прогнозують поведінку при відмовах накопичувачів. Методи оцінки перевіряють механізми реплікації та резервування. Комплексний підхід забезпечує надійність зберігання даних.

Методи оцінки систем віртуалізації аналізують ефективність розподілу ресурсів. Алгоритми тестування перевіряють ізоляцію віртуальних машин. Системи моніторингу контролюють утилізацію фізичних ресурсів. Математичні моделі прогнозують оптимальне розміщення навантаження. Методи аналізу досліджують живу міграцію віртуальних машин. Алгоритми оцінки перевіряють відмовостійкість платформи віртуалізації. Комплексне тестування забезпечує надійність віртуальної інфраструктури.

Завершуючи аналіз математичних методів та алгоритмів оцінювання знань, варто підкреслити комплексність сучасних підходів до тестування навичок програмування. Системи оцінювання використовують широкий спектр методів - від класичної теорії тестів до складних адаптивних алгоритмів на основі машинного навчання. Математичні моделі забезпечують об'єктивність та

надійність результатів перевірки знань. Методи аналізу охоплюють всі аспекти розробки програмного забезпечення - від якості коду до архітектури систем. Алгоритми тестування постійно вдосконалюються разом з розвитком технологій та підходів до програмування. Системи оцінювання стають більш інтелектуальними та адаптивними до індивідуальних особливостей студентів.

Подальший розвиток методів оцінювання буде спрямований на посилення персоналізації та автоматизації процесу перевірки знань. Математичні моделі включатимуть більше параметрів для точнішої оцінки компетенцій. Алгоритми машинного навчання дозволять краще прогнозувати траєкторію навчання кожного студента. Системи тестування забезпечуватимуть більш глибокий аналіз практичних навичок програмування [13, с. 74-87]. Методи оцінювання розвиватимуться в напрямку підвищення об'єктивності та ефективності перевірки знань. Комплексний підхід залишиться основою сучасних систем тестування програмістів.

Таким чином, математичні методи та алгоритми створюють потужний фундамент для об'єктивного оцінювання знань та навичок програмування. Постійне вдосконалення підходів до тестування сприяє підвищенню якості підготовки спеціалістів у галузі розробки програмного забезпечення. Комплексне застосування різних методів забезпечує всебічну оцінку компетенцій та допомагає формувати індивідуальні траєкторії навчання.

2.2 Порівняльний аналіз платформ перевірки знань

Сучасні освітні платформи пропонують різні підходи до перевірки знань з програмування. Кожна система має власні методи оцінювання теоретичних та практичних навичок. Функціональні можливості платформ суттєво відрізняються за повнотою та глибиною аналізу. Методи тестування коду використовують різні технології автоматизації перевірки. Системи адаптивного навчання застосовують різні алгоритми підбору завдань. Платформи відрізняються підходами до аналізу якості коду та продуктивності рішень. Технології оцінювання постійно розвиваються та вдосконалюються.

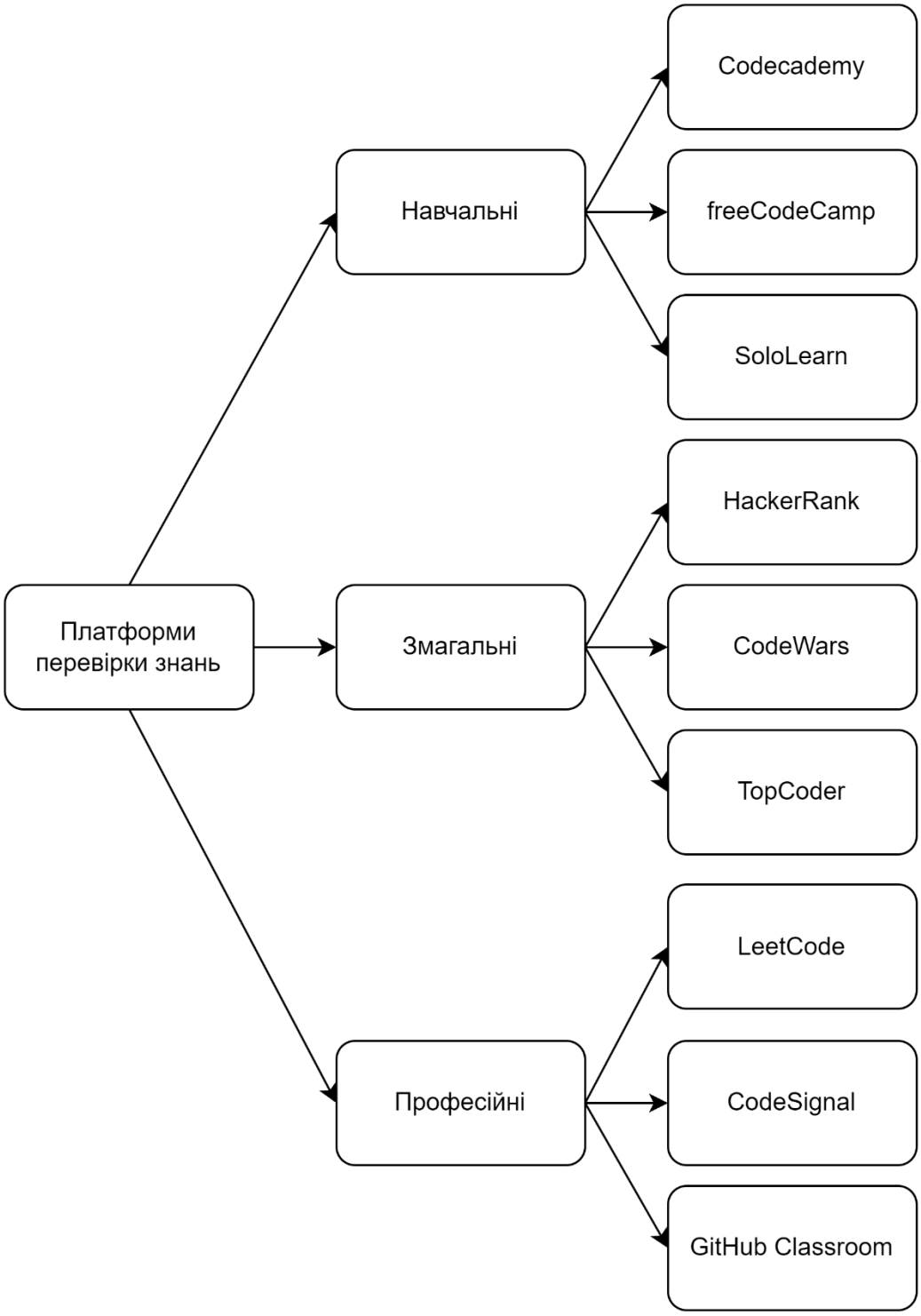


Рис. 2.2 Класифікація платформ перевірки знань

Платформа Codecademy спеціалізується на інтерактивному навчанні програмуванню. Система надає вбудоване середовище розробки з автоматичною перевіркою коду. Методи оцінювання фокусуються на практичних навичках написання програм. Платформа пропонує покроковий підхід до вивчення нових

концепцій. Теоретичні матеріали тісно інтегровані з практичними завданнями. Система відстежує прогрес проходження курсу та надає рекомендації. Методи гейміфікації підвищують залученість у навчальний процес.

Платформа HackerRank орієнтована на оцінку алгоритмічних навичок програмістів. Система містить великий банк задач різного рівня складності. Методи тестування перевіряють ефективність та оптимальність рішень. Платформа підтримує змагальний формат вирішення задач. Рейтингова система мотивує до покращення результатів. Детальна статистика допомагає відстежувати прогрес. Система широко використовується для оцінки кандидатів при працевлаштуванні.

LeetCode зосереджується на підготовці до технічних співбесід у провідних компаніях. Система пропонує задачі, що часто зустрічаються на інтерв'ю програмістів. Платформа дозволяє практикувати різні підходи до вирішення алгоритмічних задач. Методи тестування оцінюють швидкодію та ефективність використання пам'яті. Система надає детальну статистику успішності розв'язання задач. Колективне обговорення рішень сприяє обміну досвідом. База задач постійно оновлюється новими прикладами.

Платформа Coursera пропонує комплексний підхід до навчання програмуванню. Система поєднує відеолекції з практичними завданнями та тестами. Методи оцінювання охоплюють як теоретичні знання, так і практичні навички. Платформа підтримує взаємне оцінювання робіт студентами. Система забезпечує отримання сертифікатів від провідних університетів. Методи аналізу відстежують активність та успішність навчання. Платформа надає можливості для групового навчання.

CodeWars використовує унікальний підхід через систему бойових завдань. Платформа пропонує вирішення задач у форматі програмістських поєдинків. Система ранжує користувачів за рівнем майстерності у вирішенні задач. Методи оцінювання враховують елегантність та ефективність рішень. Платформа заохочує створення власних завдань спільнотою. Система підтримує вивчення різних мов програмування. Колективне обговорення сприяє пошуку оптимальних рішень.

edX надає доступ до курсів з програмування від провідних університетів. Система забезпечує структурований підхід до вивчення мов програмування. Платформа використовує різноманітні методи оцінювання знань. Методи тестування включають автоматизовану перевірку коду. Система підтримує отримання офіційних сертифікатів. Платформа надає можливості для взаємодії з викладачами. Аналітичні інструменти відстежують прогрес навчання.

GitHub Classroom спеціалізується на організації навчального процесу через систему контролю версій. Платформа автоматизує створення репозиторіїв для завдань та їх перевірку. Система використовує GitHub Actions для автоматичного тестування коду. Методи оцінювання враховують процес розробки та якість комітів. Платформа підтримує командну роботу над проектами. Викладачі отримують детальну статистику роботи студентів. Інтеграція з Git розвиває навички роботи з системами контролю версій.

Stepik пропонує адаптивний підхід до вивчення програмування. Система будує індивідуальні траєкторії навчання для кожного студента. Платформа поєднує теоретичні матеріали з інтерактивними завданнями. Методи оцінювання включають автоматичну перевірку коду та тести. Система надає детальний зворотний зв'язок щодо помилок. Платформа підтримує створення власних освітніх курсів. Аналітичні інструменти відстежують ефективність навчання.

CodeSignal фокусується на технічній оцінці навичок програмістів. Платформа пропонує завдання різного рівня складності для оцінки компетенцій. Система використовує автоматизовані тести для перевірки рішень. Методи аналізу оцінюють якість коду та ефективність алгоритмів. Платформа генерує детальні звіти про технічні навички. Система широко використовується для оцінки кандидатів. Інструменти аналітики допомагають виявити сильні та слабкі сторони.

Exercism зосереджується на практиці програмування через наставництво. Система пропонує завдання з акцентом на якість та читабельність коду. Платформа забезпечує персональний зворотний зв'язок від менторів. Методи оцінювання враховують різні аспекти програмування. Система підтримує вивчення багатьох

мов програмування [14, с. 26]. Платформа заохочує обмін досвідом між учасниками. Спільнота менторів забезпечує якісний зворотний зв'язок.

Replit забезпечує повноцінне середовище розробки в браузері. Система дозволяє створювати та тестувати програми без локальної установки. Платформа підтримує спільну роботу над кодом в реальному часі. Методи тестування включають модульні та інтеграційні тести. Система автоматично зберігає версії коду та результати виконання. Платформа надає можливості для розгортання та демонстрації проектів. Інструменти налагодження допомагають знаходити помилки.

CodeCombat використовує геймифікований підхід до навчання програмуванню. Платформа пропонує вивчення концепцій через ігрові сценарії. Система забезпечує візуальний зворотний зв'язок при виконанні завдань. Методи оцінювання інтегровані в ігровий процес. Платформа підтримує різні рівні складності завдань. Система мотивує через систему досягнень та нагород. Ігрові механіки роблять навчання захоплюючим.

HackerEarth спеціалізується на проведенні програмістських змагань. Платформа організовує хакатони та кодинг-челенджі. Система забезпечує автоматичну оцінку рішень учасників. Методи тестування перевіряють коректність та ефективність коду. Платформа підтримує різні формати змагань. Система формує рейтинги учасників та команд. Аналітичні інструменти відстежують результати змагань.

TopCoder проводить регулярні змагання з алгоритмічного програмування. Платформа має довгу історію та активну спільноту програмістів. Система використовує складні алгоритмічні задачі для оцінки навичок. Методи тестування враховують швидкість та якість рішень. Платформа підтримує командні та індивідуальні змагання. Система веде глобальний рейтинг учасників. Архів задач допомагає в підготовці до змагань.

SPOJ надає велику колекцію задач для практики програмування. Система підтримує автоматичну перевірку рішень на різних тестах. Платформа дозволяє вирішувати задачі різними мовами програмування. Методи оцінювання

фокусуються на коректності алгоритмів. Система зберігає статистику успішних спроб. Платформа підтримує створення власних наборів задач. База завдань постійно поповнюється новими прикладами.

AtCoder організовує регулярні змагання з програмування різного рівня складності. Платформа пропонує задачі від початкового до експертного рівня. Система забезпечує автоматичну перевірку рішень в реальному часі. Методи тестування оцінюють ефективність та оптимальність алгоритмів. Платформа підтримує різні формати змагань та тренувань. Система веде детальну статистику результатів учасників. Архів задач доступний для практики поза змаганнями.

Codewars акцентує увагу на вдосконаленні навичок через постійну практику. Система пропонує завдання у форматі програмістських ката. Платформа заохочує пошук елегантних та ефективних рішень. Методи оцінювання враховують якість та стиль коду. Система підтримує створення власних завдань спільнотою. Платформа сприяє обміну знаннями між учасниками. Рейтингова система відображає прогрес у навчанні.

freeCodeCamp надає структурований підхід до вивчення веб-розробки. Платформа пропонує повний курс від основ до складних проектів. Система включає інтерактивні уроки та практичні завдання. Методи оцінювання перевіряють розуміння концепцій через проекти. Платформа підтримує отримання сертифікатів за різними напрямками. Система забезпечує зворотний зв'язок від спільноти. Навчальні матеріали постійно оновлюються.

SoloLearn фокусується на мобільному навчанні програмуванню. Платформа пропонує короткі інтерактивні уроки та завдання. Система адаптує темп навчання під кожного користувача. Методи оцінювання включають тести та практичні завдання. Платформа підтримує соціальну взаємодію між учнями. Система надає миттєвий зворотний зв'язок. Мобільний формат забезпечує зручність навчання.

CheckiO перетворює навчання програмуванню в захоплюючу гру. Платформа пропонує вирішення задач у форматі місій. Система заохочує пошук креативних рішень. Методи оцінювання враховують різні підходи до розв'язання. Платформа

підтримує обговорення та порівняння рішень. Система надає детальні пояснення до кожної задачі. Ігрові елементи підвищують мотивацію до навчання.

Codingame пропонує вирішення задач через створення ігрових ботів. Платформа використовує візуалізацію для демонстрації роботи програм. Система організовує змагання між ботами користувачів. Методи оцінювання перевіряють ефективність стратегій та алгоритмів. Платформа підтримує різні мови програмування для створення ботів [15]. Система формує рейтинги найкращих рішень. Візуальний формат робить процес навчання цікавішим.

Project Euler фокусується на математичних задачах, що потребують програмування. Платформа пропонує складні проблеми для розвитку алгоритмічного мислення. Система перевіряє тільки кінцеві результати обчислень. Методи оцінювання заохочують пошук оптимальних рішень. Платформа підтримує обговорення задач після їх вирішення. Система відстежує прогрес у розв'язанні завдань. База задач постійно поповнюється новими проблемами.

GeeksforGeeks надає комплексну платформу для вивчення комп'ютерних наук. Система містить великий обсяг навчальних матеріалів та задач. Платформа пропонує структуровані курси з різних тем програмування. Методи оцінювання включають тести та практичні завдання. Система підтримує підготовку до технічних співбесід. Платформа регулярно оновлює контент новими матеріалами. Спільнота активно ділиться знаннями та досвідом.

Таким чином, аналіз існуючих платформ показує різноманітність підходів до перевірки знань з програмування. Кожна система має свої унікальні особливості та методи оцінювання. Комбінація різних підходів дозволяє створити ефективну систему навчання та контролю знань. Постійний розвиток платформ сприяє вдосконаленню методів оцінювання.

2.3 Особливості реалізації в Java

Розробка системи оцінювання на Java вимагає врахування специфіки мови програмування. Середовище виконання Java надає потужні інструменти для створення багатокомпонентних систем. Платформа забезпечує надійність та

безпеку роботи додатків. Віртуальна машина Java гарантує однакову поведінку програм на різних платформах. Система типів Java допомагає уникнути багатьох помилок на етапі компіляції. Розвинена екосистема бібліотек спрощує розробку складних систем. Інструменти розробки надають широкі можливості для тестування та налагодження.

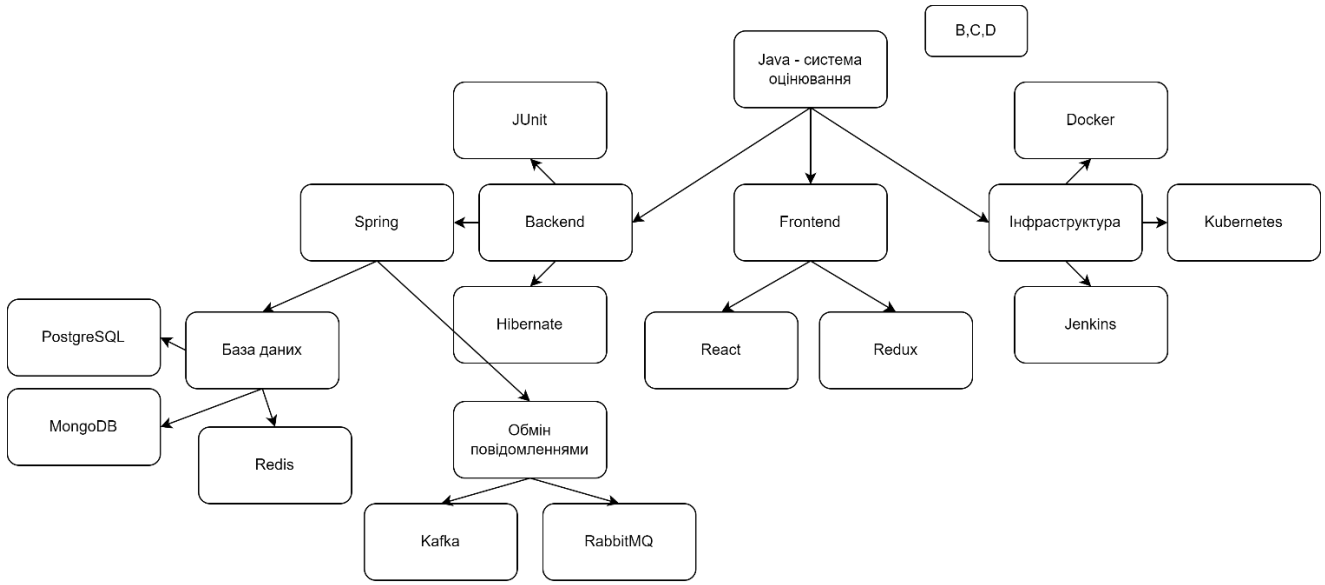


Рис. 2.3 Технологічний стек Java-системи оцінювання

Spring Framework забезпечує основу для побудови серверної частини системи. Механізм впровадження залежностей спрощує управління компонентами додатку. Система конфігурації дозволяє гнучко налаштовувати поведінку програми. Spring Security надає комплексні механізми захисту даних. Модульна структура фреймворку підтримує розширення функціональності. Spring Data спрощує роботу з різними джерелами даних. Інтеграційні можливості забезпечують взаємодію з іншими системами.

Hibernate реалізує зберігання даних у реляційних базах даних. Система об'єктно-реляційного відображення автоматизує роботу з базою даних. Механізми кешування підвищують продуктивність операцій з даними. Підтримка транзакцій забезпечує цілісність інформації. Система подій дозволяє реагувати на зміни даних.

Гнучкі запити надають широкі можливості для вибірки інформації. Hibernate інтегрується з різними СУБД.

JUnit забезпечує фреймворк для тестування компонентів системи. Механізми тестування дозволяють перевіряти різні аспекти роботи програми. Система підтримує модульні та інтеграційні тести. Анотації спрощують написання тестових сценаріїв. Засоби mock-об'єктів дозволяють ізолювати компоненти при тестуванні. Параметризовані тести перевіряють різні набори вхідних даних. Система звітів надає детальну інформацію про результати тестування.

Maven управляє життєвим циклом збирання та розгортання системи. Декларативна конфігурація спрощує налаштування процесу збирання. Система залежностей автоматизує підключення необхідних бібліотек. Плагіни розширюють можливості управління проектом. Профілі збирання підтримують різні середовища розгортання. Репозиторії забезпечують централізоване зберігання артефактів. Інтеграція з системами безперервної інтеграції автоматизує розгортання.

Docker контейнеризує компоненти системи для спрощення розгортання. Образи контейнерів забезпечують ізоляцію середовища виконання. Система оркестрації управляє розподіленою інфраструктурою. Мережа контейнерів забезпечує взаємодію компонентів. Томи даних зберігають персистентну інформацію. Механізми масштабування підтримують зростання навантаження. Docker Compose спрощує управління багатоконтейнерними додатками.

Apache Kafka реалізує обмін повідомленнями між компонентами системи. Розподілений журнал забезпечує надійну доставку повідомлень. Партиціювання підтримує паралельну обробку даних. Реплікація гарантує відмовостійкість системи. Потокова обробка дозволяє аналізувати дані в реальному часі. Механізми збереження забезпечують довготривале зберігання повідомлень. Kafka Connect спрощує інтеграцію з іншими системами.

React реалізує користувацький інтерфейс системи оцінювання. Компонентний підхід спрощує розробку складних інтерфейсів. Віртуальний DOM забезпечує високу продуктивність оновлень. Система станів управляє даними компонентів. Хуки спрощують роботу з життєвим циклом. Маршрутизація

забезпечує навігацію між сторінками. Redux централізує управління станом додатку.

Elasticsearch забезпечує повнотекстовий пошук та аналітику даних. Розподілена архітектура підтримує обробку великих обсягів інформації. Система індексації оптимізує швидкість пошуку. Агрегації дозволяють аналізувати дані в різних розрізах. Механізми ранжування забезпечують релевантність результатів. API надає гнучкі можливості для запитів. Інтеграція з Logstash та Kibana формує потужний аналітичний стек.

Prometheus реалізує моніторинг роботи системи оцінювання. Збір метрик охоплює всі компоненти розподіленої системи. Система алертів повідомляє про проблеми в роботі. Механізми запитів дозволяють аналізувати тренди показників. Графіки візуалізують стан системи в реальному часі. Інтеграція з Grafana забезпечує створення інформативних дашбордів. Експортери спрощують підключення різних джерел метрик.

Redis реалізує швидке кешування даних в оперативній пам'яті. Структури даних оптимізують зберігання різної інформації. Механізми реплікації забезпечують відмовостійкість кешу. Система подій підтримує реактивну взаємодію компонентів. Транзакції гарантують атомарність операцій. Політики витіснення управляють обсягом кешу. Persistence зберігає дані між перезапусками.

RabbitMQ забезпечує асинхронну взаємодію компонентів системи. Механізми маршрутизації гнучко налаштовують потоки повідомлень. Підтвердження доставки гарантують надійність обміну даними. Черги повідомлень балансують навантаження між споживачами. Федерація підтримує розподілену архітектуру. Плагіни розширюють функціональність брокера. Веб-інтерфейс спрощує моніторинг та управління.

Keycloak реалізує систему автентифікації та авторизації. Підтримка стандартів OAuth2 та OpenID Connect забезпечує безпечний доступ. Система ролей гнучко налаштовує права користувачів. Соціальні провайдери спрощують реєстрацію. Двофакторна автентифікація підвищує безпеку. Федерація

ідентифікації об'єднує різні системи автентифікації. Адміністративна консоль управляє налаштуваннями безпеки.

MongoDB зберігає документи з динамічною структурою даних. Система шардингу забезпечує горизонтальне масштабування бази даних. Реплікація підтримує відмовостійкість зберігання інформації. Індекси оптимізують швидкість виконання запитів. Агрегаційний конвеєр дозволяє обробляти дані на стороні бази. Механізми валідації забезпечують цілісність даних. Підтримка транзакцій гарантує узгодженість інформації.

Jenkins автоматизує процеси збірки та розгортання системи. Конвеєри описують послідовність кроків безперервної інтеграції. Агенти розподіляють навантаження при виконанні завдань. Плагіни розширюють можливості автоматизації. Тригери запускають процеси при змінах у репозиторії. Артефакти зберігають результати збірки. Сповіщення інформують про статус виконання завдань.

Nginx забезпечує балансування навантаження та проксування запитів. Конфігурація віртуальних хостів розділяє трафік між додатками. Кешування статичного контенту підвищує продуктивність. SSL термінація забезпечує шифрування з'єднань. Обмеження частоти захищають від DDoS атак. Переписування URL спрощує маршрутизацію запитів. Журналювання допомагає аналізувати трафік.

PostgreSQL зберігає реляційні дані з підтримкою транзакцій. Розширення додають специфічну функціональність до бази даних. Тригери автоматизують реакцію на зміни даних. Матеріалізовані подання кешують результати складних запитів. Повнотекстовий пошук індексує текстовий контент. Партиціювання оптимізує роботу з великими таблицями. Реплікація забезпечує високу доступність даних.

ELK Stack обробляє та аналізує логи системи оцінювання. Logstash збирає та трансформує дані з різних джерел. Elasticsearch індексує логи для швидкого пошуку. Kibana візуалізує результати аналізу логів. Beats спрощують збір метрик

та логів. Machine Learning виявляє аномалії в даних. Watches налаштовують сповіщення про події.

MinIO забезпечує об'єктне сховище для файлів системи. Сумісність з API Amazon S3 спрощує інтеграцію з додатками. Шифрування захищає конфіденційні дані при зберіганні. Політики доступу контролюють права на файли. Версіонування відстежує зміни об'єктів. Реплікація забезпечує надійність зберігання. Веб-консоль спрощує управління сховищем.

Kubernetes оркеструє контейнери в розподіленому середовищі. Поди групують контейнери в логічні одиниці розгортання. Сервіси забезпечують стабільний доступ до подів. Розгортання керують оновленням версій додатків. Автомасштабування адаптує ресурси під навантаження. Конфігурації та секрети зберігають налаштування. Мережеві політики контролюють комунікацію між подами.

Apache Cassandra зберігає дані з високою доступністю. Розподілена архітектура забезпечує лінійну масштабованість. Реплікація між вузлами гарантує надійність даних. Налаштування узгодженості балансує надійність та швидкодію. Компактні сховища оптимізують використання диску [16]. Механізм ремонту відновлює пошкоджені дані. CQL спрощує роботу з базою даних.

Таким чином, технології Java та супутні інструменти створюють потужну основу для реалізації системи оцінювання. Розподілена архітектура забезпечує надійність та масштабованість рішення. Комбінація різних технологій дозволяє створити гнучку та ефективну систему. Вибір конкретних інструментів визначається вимогами проекту.

3 ВПРОВАДЖЕННЯ АДАПТИВНИХ АЛГОРИТМІВ У ВИВЧЕННЯ JAVA

3.1 Розробка адаптивного алгоритму оцінювання знань з Java-програмування

Алгоритм адаптивного оцінювання базується на аналізі поточного рівня знань студента. Система починає тестування з завдань середньої складності для початкової оцінки. Результати кожної відповіді впливають на вибір наступного завдання. Математична модель враховує складність питань та ймовірність правильної відповіді. Алгоритм використовує метод Баєса підхід для уточнення оцінки рівня знань. При цьому застосовується наступний математичний апарат:

$$\frac{P(\theta|X) = P(X|\theta) \times P(\theta)}{P(X)} \tag{3.1}$$

де:

$P(\theta|X)$ - апостеріорна ймовірність рівня знань θ після спостереження результату X

$P(X|\theta)$ - ймовірність отримання результату X при рівні знань θ

$P(\theta)$ - апріорна ймовірність рівня знань

$P(X)$ - загальна ймовірність спостереження результату X

Для послідовності відповідей оцінка рівня знань уточнюється ітеративно:

$$\frac{P(\theta|X_1,X_2,\ldots,X_{\mathrm{n}}) = P(X_{\mathrm{n}}|\theta) \times P(\theta|X_1,X_2,\ldots,X_{\mathrm{n}-1})}{P(X_{\mathrm{n}})} \tag{3.2}$$

Ймовірність правильної відповіді при заданому рівні знань θ та складності завдання β моделюється логістичною функцією:

$$\frac{P(X=1|\theta,\beta) = 1}{(1 + e\text{^}(-1.7(\theta-\beta)))} \tag{3.3}$$

Кінцева оцінка рівня знань розраховується як математичне очікування апостеріорного розподілу:

$$\theta = \int \theta \times P(\theta|X_1, X_2, \ldots, X_n)\, d\theta \quad (3.4)$$

Довірчий інтервал оцінки визначається через квантилі апостеріорного розподілу.

Система аналізує час виконання завдань та спроби знайти рішення. Методи машинного навчання допомагають прогнозувати успішність виконання завдань.

Таблиця 3.1

Параметри адаптивного алгоритму

Параметр	**Опис**	**Діапазон значень**
Складність завдання	Оцінка складності на основі статистики	0.1 - 1.0
Рівень знань	Поточна оцінка підготовки студента	0 - 100
Час виконання	Нормалізований час вирішення завдання	0.5 - 2.0
Кількість спроб	Число спроб знайти рішення	1 - N
Точність оцінки	Довірчий інтервал оцінки рівня	0.1 - 1.0

Математичне обґрунтування параметрів адаптивного алгоритму:

1. Складність завдання (T) розраховується на основі статистичних даних:

$$T = 1 - \left(\frac{Nc}{Nt}\right) \quad (3.5)$$

де:

Nc - кількість правильних відповідей на завдання

Nt - загальна кількість спроб виконання завдання Діапазон $[0.1 - 1.0]$,

де 1.0 означає максимальну складність

2. Рівень знань (K) студента оцінюється за формулою:

$$K = (\frac{\sum(Wi \times Ri)}{\sum Wi}) \times 100 \quad (3.6)$$

де:

Wi - вага і-того завдання

Ri - результат виконання і-того завдання (1 - правильно, 0 - неправильно) Діапазон $[0 - 100]$

3. Час виконання (E) нормалізується відносно еталонного часу:

$$E = \frac{Tu}{Te} \quad (3.7)$$

де:

Tu - фактичний час виконання завдання користувачем

Te - еталонний час виконання завдання Діапазон $[0.5 - 2.0]$,

де 1.0 відповідає еталонному часу

4. Точність оцінки (P) визначається через довірчий інтервал:

$$P = 1 - (\frac{\sigma}{(K \times \sqrt{n})}) \quad (3.8)$$

де:

σ - стандартне відхилення результатів

K - поточний рівень знань

n - кількість виконаних завдань Діапазон $[0.1 - 1.0]$,

де 1.0 означає максимальну точність

Комплексна оцінка рівня підготовки студента (L) розраховується як:

$$L = \alpha \times K + \beta \times E + \gamma \times P \quad (3.9)$$

де:

α, β, γ - вагові коефіцієнти ($\alpha + \beta + \gamma = 1$)

K - рівень знань

E - ефективність виконання (час)

P - точність оцінки

Адаптивний алгоритм використовує ці метрики для динамічного підбору наступного завдання, обираючи його складність T таким чином, щоб максимізувати приріст точності оцінки P при поточному рівні знань K.

Процес адаптивного тестування включає декілька основних етапів. Початкова фаза визначає приблизний рівень підготовки студента. Основний етап уточнює оцінку через підбір оптимальних завдань.

Система аналізує патерни відповідей для виявлення проблемних тем. Алгоритм коригує складність наступних завдань на основі результатів.

Фінальна фаза формує підсумкову оцінку рівня знань. Процес тестування завершується при досягненні заданої точності оцінювання.

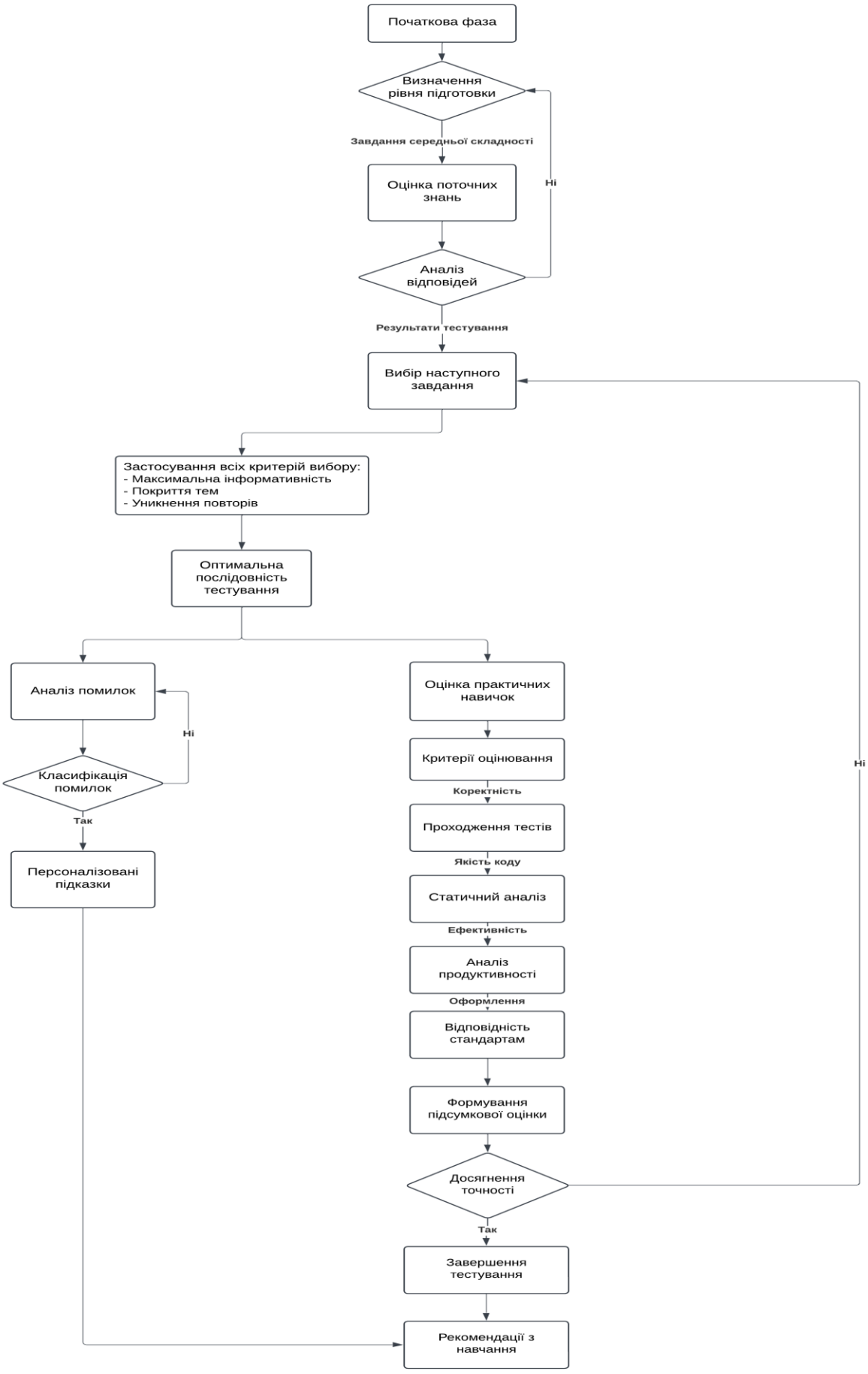


Рис. 3.1 Алгоритм адаптивного тестування

Оцінювання практичних навичок програмування вимагає комплексного підходу. Система перевіряє коректність роботи програми на різних наборах тестів. Алгоритм аналізує якість коду та відповідність стандартам оформлення. Час виконання та використання ресурсів впливають на підсумкову оцінку. Модуль тестування контролює самостійність виконання завдань. Система порівнює рішення з базою відомих реалізацій. Оцінка практичних завдань враховує декілька критеріїв.

Таблиця 3.2

Критерії оцінки практичних завдань

Критерій	**Вага**	**Метод оцінювання**
Коректність роботи	40%	Проходження тестів
Якість коду	25%	Статичний аналіз
Ефективність	20%	Аналіз продуктивності
Оформлення	15%	Відповідність стандартам

Вагові коефіцієнти критеріїв оцінки практичних завдань були визначені на основі методу аналітичної ієрархії (MAI). Процес включав наступні етапи:

1. Побудова матриці попарних порівнянь критеріїв експертами:

	$K1$	$K2$	$K3$	$K4$
$K1$	1	2	3	4
$K2$	$1/2$	1	2	3
$K3$	$1/3$	$1/2$	1	2
$K4$	$1/4$	$1/3$	$1/2$	1

де:

$K1$ - Коректність роботи

$K2$ - Якість коду

$K3$ - Ефективність

$K4$ – Оформлення

2. Розрахунок власного вектора матриці:

$$w = \frac{\sqrt{(\prod a_{ij})^{n}}}{\sum\sqrt{(\prod a_{ij})^{n}}} \quad (3.10)$$

3. Нормалізація отриманих значень дала наступні вагові коефіцієнти:

— w_1 = 0.40 (40%) для коректності роботи

— w_2 = 0.25 (25%) для якості коду

— w_3 = 0.20 (20%) для ефективності

— w_4 = 0.15 (15%) для оформлення

4. Перевірка узгодженості оцінок: Індекс узгодженості $CI = 0.027$

Відношення узгодженості $CR = 0.031 < 0.1$,

що підтверджує достовірність результатів.

Підсумкова оцінка практичного завдання розраховується за формулою:

$$S = 0.4 \times K_1 + 0.25 \times K_2 + 0.2 \times K_3 + 0.15 \times K_4 \quad (3.11)$$

де:

K_1 - оцінка коректності роботи (0-100)

K_2 - оцінка якості коду (0-100)

K_3 - оцінка ефективності (0-100)

K_4 - оцінка оформлення (0-100)

Алгоритм вибору наступного завдання використовує метод максимальної інформативності. Система розраховує очікуваний приріст точності оцінки для кожного завдання. База завдань структурована за темами та рівнями складності. Вибір враховує покриття різних аспектів мови програмування. Алгоритм уникає повторення однотипних завдань. Система відстежує час, витрачений на кожну тему. Метод забезпечує оптимальну послідовність тестування.

Система аналізу відповідей визначає причини помилок у розв'язках. Алгоритм класифікує типові помилки програмування на різні категорії. Модуль генерує персоналізовані підказки для виправлення недоліків. База знань накопичує

інформацію про поширені проблеми. Система формує рекомендації щодо вивчення складних тем [17]. Аналіз помилок допомагає виявити прогалини в знаннях. Результати використовуються для адаптації навчального процесу.

Методи профілювання оцінюють ефективність написаних програм. Система вимірює час виконання на різних наборах вхідних даних. Алгоритм аналізує використання процесора та пам'яті. Модуль порівнює показники з еталонними реалізаціями. Профілювання виявляє неоптимальні ділянки коду. Система надає рекомендації щодо оптимізації рішень. Оцінка ефективності впливає на підсумковий результат.

Модуль перевірки якості коду використовує статичний аналіз програм. Система оцінює складність методів та класів за різними метриками. Алгоритм перевіряє дотримання принципів об'єктно-орієнтованого програмування. Аналізатор виявляє дублювання коду та потенційні помилки. Модуль контролює найменування змінних та методів. Система перевіряє наявність та якість коментарів. Оцінка якості коду враховує множину параметрів. Алгоритм виявлення плагіату порівнює рішення з базою відомих реалізацій. Система використовує методи токенізації та нормалізації коду. Модуль визначає структурну схожість програм. Алгоритм враховує можливість перейменування ідентифікаторів. База даних містить типові рішення навчальних завдань. Система розраховує відсоток унікальності коду. Перевірка на плагіат забезпечує самостійність виконання завдань.

Компонент оцінки алгоритмічної складності аналізує ефективність рішень. Система розраховує асимптотичні оцінки часової складності. Модуль перевіряє оптимальність використання структур даних. Алгоритм порівнює складність з еталонними реалізаціями. База знань містить інформацію про оптимальні рішення типових задач. Система надає рекомендації щодо покращення алгоритмів. Оцінка складності впливає на загальний результат. Методи аналізу стилю програмування перевіряють відповідність стандартам. Система контролює форматування та структурування коду. Модуль оцінює читабельність та зрозумілість програм. Алгоритм перевіряє дотримання принципів чистого коду. База правил містить

рекомендації щодо оформлення програм. Система виявляє відхилення від прийнятих конвенцій. Оцінка стилю формує культуру програмування.

Адаптивний алгоритм враховує швидкість засвоєння матеріалу. Система аналізує динаміку виконання завдань різної складності. Модуль відстежує прогрес у вивченні окремих тем. Алгоритм коригує темп подання нового матеріалу. База даних зберігає історію навчання кожного студента. Система формує індивідуальні траєкторії навчання. Адаптація темпу підвищує ефективність освоєння матеріалу. Компонент моніторингу прогресу відстежує досягнення навчальних цілей. Система формує звіти про вивчення різних розділів мови Java. Модуль розраховує відсоток засвоєння кожної теми. Алгоритм виявляє теми, що потребують додаткової уваги. База даних зберігає детальну статистику навчання. Система генерує рекомендації щодо повторення матеріалу. Моніторинг забезпечує контроль якості навчання.

Методи прогнозування успішності використовують машинне навчання. Система аналізує історичні дані про проходження курсу. Алгоритм виявляє фактори, що впливають на результати навчання. Модуль прогнозує ймовірність успішного завершення курсу. База моделей постійно оновлюється новими даними. Система формує рекомендації для підвищення успішності. Прогнозування допомагає запобігти проблемам у навчанні.

Компонент автоматизованої генерації завдань створює унікальні варіанти задач. Система використовує шаблони для формування нових завдань. Модуль перевіряє складність згенерованих варіантів. Алгоритм забезпечує покриття всіх аспектів теми. База шаблонів постійно розширюється новими типами завдань. Система валідує коректність згенерованих тестів. Автоматична генерація розширює банк завдань. Методи групового аналізу виявляють загальні тенденції в навчанні. Система порівнює результати різних груп студентів. Алгоритм визначає найбільш складні теми для засвоєння. Модуль аналізує ефективність різних підходів до навчання. База даних накопичує статистику по всіх групах. Система формує рекомендації щодо вдосконалення курсу. Груповий аналіз допомагає оптимізувати навчальний процес.

Адаптивний алгоритм постійно вдосконалюється на основі нових даних. Система збирає інформацію про ефективність різних підходів до оцінювання. Модуль аналізує кореляцію оцінок з реальними навичками. Алгоритм коригує параметри на основі зворотного зв'язку. База даних накопичує досвід застосування системи. Методи машинного навчання покращують точність оцінювання. Постійне вдосконалення підвищує якість системи [18]. Компонент оцінки soft skills аналізує додаткові навички програмістів. Система відстежує активність участі в командних проектах. Модуль оцінює якість комунікації та взаємодопомоги. Алгоритм аналізує здатність до самостійного вирішення проблем. База критеріїв включає різні аспекти професійних навичок. Методи оцінювання враховують відгуки інших учасників. Розвиток soft skills підвищує конкурентоспроможність спеціалістів.

Методи аналізу патернів навчання виявляють індивідуальні особливості. Система відстежує найбільш ефективний час для навчання. Алгоритм визначає оптимальну тривалість занять. Модуль аналізує вплив перерв на засвоєння матеріалу. База даних зберігає інформацію про режими навчання. Система формує рекомендації щодо організації занять. Врахування патернів підвищує ефективність навчання. Адаптивний алгоритм визначає оптимальну послідовність вивчення тем. Система аналізує залежності між різними концепціями мови Java. Модуль враховує необхідні передумови для кожної теми. Алгоритм будує індивідуальні навчальні траєкторії. База знань містить карту взаємозв'язків між темами. Методи планування оптимізують порядок вивчення матеріалу. Правильна послідовність забезпечує глибоке розуміння.

Компонент рейтингового оцінювання формує об'єктивні показники успішності. Система враховує складність виконаних завдань та якість рішень. Модуль розраховує персональний рейтинг кожного студента. Алгоритм визначає відносний рівень підготовки в групі. База даних зберігає історію змін рейтингу. Методи ранжування забезпечують чесну конкуренцію. Рейтингова система мотивує до покращення результатів.

3.2 Розробка онлайн-платформи з вивчення Java та проектування системи оцінювання знань

Розробка системи оцінювання знань починається з визначення основних компонентів архітектури. Модульна структура платформи забезпечує гнучкість та масштабованість рішення. Система включає модулі тестування теоретичних знань та практичних навичок. База даних зберігає інформацію про студентів, завдання та результати оцінювання. Аналітичний модуль обробляє статистику та формує рекомендації. Підсистема адаптивного тестування керує вибором оптимальних завдань. Інтеграційні компоненти забезпечують взаємодію з зовнішніми сервісами.

Таблиця 3.3

Основні компоненти системи оцінювання

Компонент	**Призначення**	**Функціональність**
Модуль тестування	Перевірка знань	Теоретичні тести, практичні завдання
База даних	Зберігання даних	Профілі, завдання, результати
Аналітичний модуль	Обробка результатів	Статистика, рекомендації
Адаптивний модуль	Управління тестуванням	Вибір завдань, оцінка рівня
Інтеграційний модуль	Зовнішні взаємодії	API, автентифікація

Архітектура системи базується на мікросервісному підході для забезпечення гнучкості. Окремі сервіси відповідають за управління користувачами та авторизацію. Модуль тестування реалізований як набір спеціалізованих мікросервісів. Система зберігання даних використовує розподілені бази даних. Аналітичні компоненти працюють з потоками даних в реальному часі. Сервіси розгортаються в контейнерному середовищі. Масштабування забезпечується на рівні окремих компонентів.

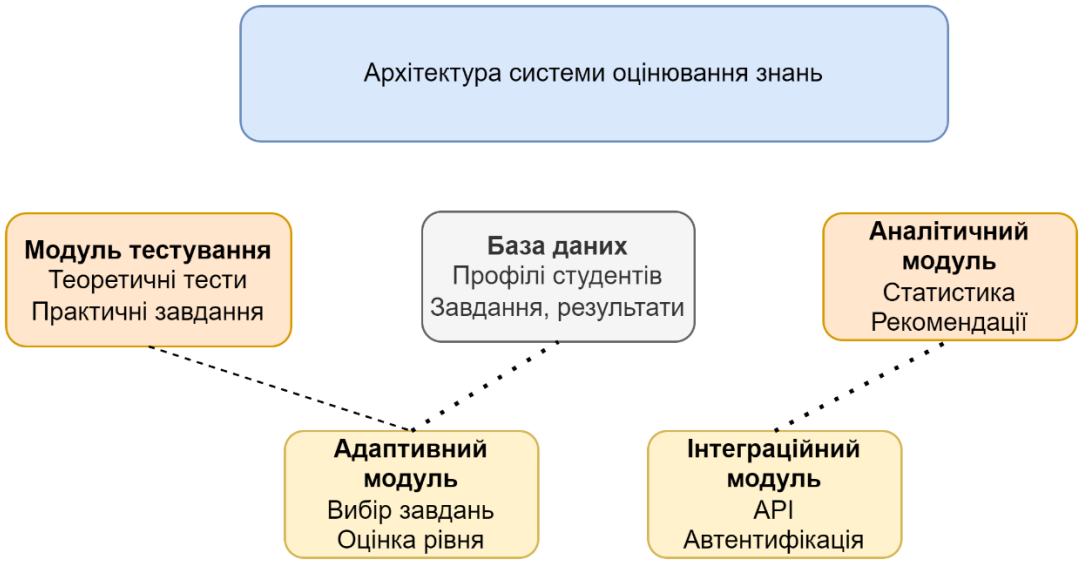


Рис. 3.2 Архітектура системи оцінювання знань

Модуль тестування теоретичних знань реалізує різні формати перевірки. Система підтримує питання з одиночним та множинним вибором відповідей. База завдань включає питання на встановлення відповідності та послідовності. Тести можуть містити питання з короткою текстовою відповіддю. Модуль забезпечує автоматичну перевірку відповідей та розрахунок балів. Система аналізує статистику відповідей для оцінки складності питань. Банк завдань постійно оновлюється та розширюється.

Таблиця 3.4

Типи тестових завдань

Тип завдання	Формат відповіді	Метод оцінювання
Одиночний вибір	Один варіант	Точне співпадіння
Множинний вибір	Декілька варіантів	Частково правильні
Відповідність	Пари значень	Кількість правильних пар
Послідовність	Упорядкований список	Правильність порядку
Текстова відповідь	Короткий текст	Ключові слова

Практичні завдання перевіряють навички написання програмного коду. Система включає середовище розробки з підтримкою автодоповнення та підсвічування синтаксису. Модуль тестування автоматично перевіряє код на різних наборах вхідних даних. Аналізатор коду оцінює якість реалізації та відповідність вимогам. Система відстежує час виконання та використання ресурсів. Звіти про помилки містять детальну інформацію для виправлення. Студенти отримують миттєвий зворотний зв'язок щодо своїх рішень.

Діаграма варіантів використання для онлайн-платформи навчання мові програмування Java показує різні взаємодії між користувачами (студентами, менторами та адміністраторами) та функціональними можливостями платформи (рис.3.3):

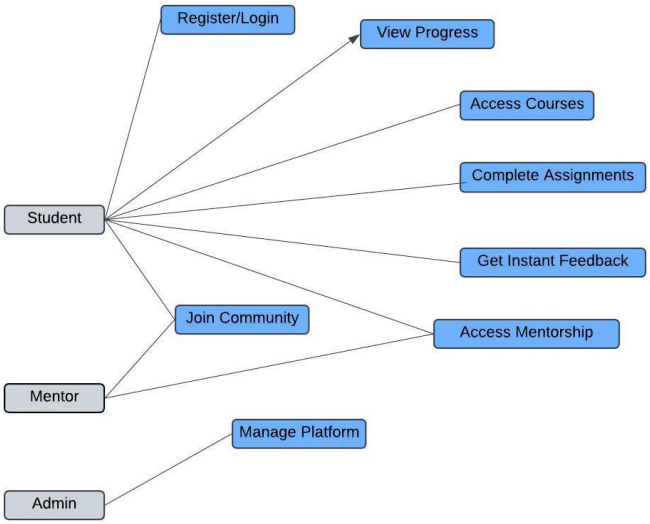


Рис. 3.3 Діаграма варіантів використання

Student (Студент) може:

- Реєструватися/входити в систему
- Долучатися до курсів
- Виконувати завдання
- Отримувати миттєвий зворотний зв'язок
- Приєднуватися до спільноти
- Долучатися до менторства

- Переглядати свій прогрес

Mentor (Ментор) може:

- Давати доступ до менторства
- Приєднуватися до спільноти

Admin (Адміністратор) може:

- Керувати платформою.

Для більш детального розуміння роботи платформи важливо розглянути її архітектуру. Схема архітектури включає основні компоненти та їх взаємодії, що дозволяють платформі функціонувати ефективно (рис.3.4).

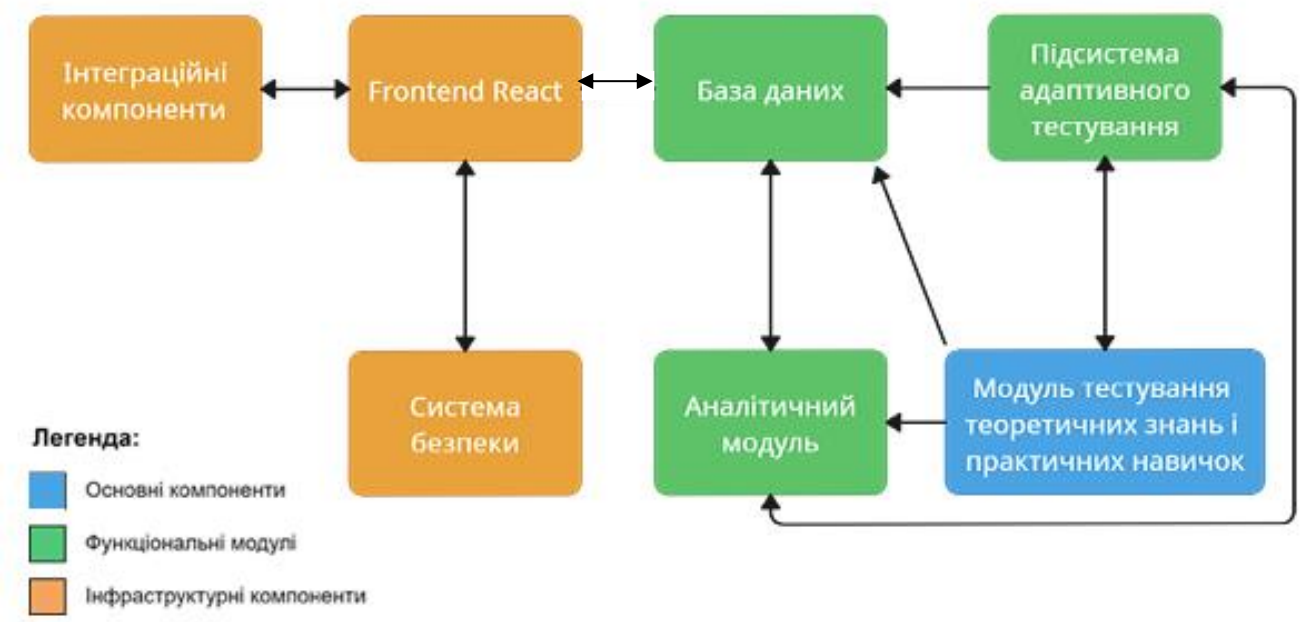


Рис. 3.4 Архітектура розробленої онлайн-платформи

База даних системи забезпечує надійне зберігання всієї необхідної інформації. Профілі студентів містять дані про прогрес навчання та результати тестування. Система зберігає історію виконання завдань та статистику відповідей. База завдань включає теоретичні питання та практичні задачі різних рівнів складності. Метадані описують характеристики та взаємозв'язки навчальних матеріалів. Аналітичні дані накопичуються для подальшої обробки та формування рекомендацій. Резервне копіювання забезпечує цілісність та доступність даних.

Модуль адаптивного тестування реалізує алгоритми вибору оптимальних завдань. Система оцінює поточний рівень знань на основі попередніх відповідей. Банк завдань містить питання з відомими параметрами складності. Алгоритм вибирає завдання, що найкраще оцінюють знання студента. Адаптивне тестування мінімізує кількість питань для точної оцінки. Система враховує індивідуальні особливості навчання. Траєкторія тестування коригується в реальному часі.

Аналітичний модуль обробляє дані для формування рекомендацій та звітів. Система збирає статистику виконання завдань та активності студентів. Алгоритми аналізу виявляють патерни навчання та складні теми. Модуль генерує персоналізовані рекомендації щодо вивчення тем. Викладачі отримують детальну аналітику про прогрес групи. Система формує індивідуальні плани навчання на основі аналізу даних.

Підсистема автентифікації та авторизації забезпечує безпечний доступ до платформи. Реєстрація користувачів вимагає підтвердження електронної пошти. Система підтримує різні ролі з відповідними рівнями доступу. Механізми безпеки захищають персональні дані та результати тестування. Модуль відстежує підозрілу активність та спроби шахрайства. Двофакторна автентифікація підвищує захищеність облікових записів. Система регулярно оновлює політики безпеки.

Інтерфейс системи забезпечує зручну взаємодію користувачів з платформою. Адаптивний дизайн підтримує роботу з різних пристроїв та браузерів. Навігація дозволяє швидко знаходити потрібні матеріали та функції. Панель моніторингу відображає прогрес та рекомендації [19]. Інтерактивні елементи роблять процес навчання більш захоплюючим. Система надає підказки та допомогу при виконанні завдань. Інтерфейс підтримує локалізацію на різні мови.

Підсистема сповіщень інформує користувачів про події та оновлення. Студенти отримують нагадування про дедлайни та рекомендовані завдання. Система надсилає повідомлення про результати перевірки робіт. Викладачі отримують сповіщення про активність студентів. Модуль підтримує різні канали комунікації - email, push-повідомлення, месенджери. Налаштування дозволяють

керувати частотою та типами сповіщень. Система зберігає історію комунікацій для аудиту.

Компонент управління контентом забезпечує створення та оновлення навчальних матеріалів. Редактор дозволяє додавати теоретичні матеріали та тести. Система підтримує різні формати контенту - текст, зображення, відео. База знань структурована за темами та рівнями складності. Механізми версіонування відстежують зміни в матеріалах. Модуль перевіряє якість та узгодженість контенту. Система контролює актуальність навчальних матеріалів. Система моніторингу відстежує стан компонентів та продуктивність платформи. Збір метрик охоплює всі критичні параметри роботи сервісів. Алгоритми аналізу виявляють аномалії та потенційні проблеми. Модуль формує звіти про доступність та швидкодію системи. Механізми оповіщення інформують про інциденти та збої. Графіки та дашборди візуалізують ключові показники роботи. Система зберігає історичні дані для аналізу тенденцій.

Підсистема управління користувачами забезпечує адміністрування облікових записів. Функціонал дозволяє створювати групи та призначати ролі користувачам. Система відстежує активність та статус облікових записів. Адміністратори можуть керувати правами доступу та налаштуваннями. Модуль підтримує інтеграцію з корпоративними системами автентифікації. Механізми аудиту фіксують зміни в налаштуваннях користувачів. Система регулярно верифікує актуальність даних облікових записів. Компонент формування звітності генерує аналітичні документи різних типів. Система створює індивідуальні звіти про прогрес студентів. Викладачі отримують агреговану статистику по групах та курсах. Модуль підтримує різні формати експорту даних для подальшої обробки [20]. Механізми планування дозволяють налаштувати регулярну відправку звітів. Система зберігає шаблони для різних типів аналітичних документів. Гнучкі фільтри допомагають формувати звіти за різними критеріями.

Підсистема резервного копіювання забезпечує збереження та відновлення даних. Система створює регулярні резервні копії всієї інформації. Модуль підтримує інкрементальне та повне копіювання даних. Механізми відновлення

дозволяють швидко відновити роботу при збоях. Система перевіряє цілісність резервних копій та тестує процедури відновлення. Архівні копії зберігаються в географічно розподілених локаціях. Політики зберігання визначають терміни утримання резервних копій.

Інтеграційні компоненти забезпечують взаємодію з зовнішніми системами. API підтримує різні протоколи та формати обміну даними. Система експортує результати тестування в навчальні системи закладів. Модуль інтегрується з популярними середовищами розробки. Механізми синхронізації забезпечують актуальність даних між системами [21]. Компоненти підтримують розширення через плагіни та адаптери. Система контролює якість інтеграційних взаємодій.

Підсистема тестування продуктивності оцінює ефективність роботи платформи. Модуль створює синтетичне навантаження для перевірки масштабованості системи. Тести симулюють одночасну роботу великої кількості користувачів. Компоненти вимірюють час відгуку та пропускну здатність сервісів. Система збирає метрики використання ресурсів під навантаженням. Алгоритми аналізу виявляють вузькі місця в архітектурі. Результати тестування використовуються для оптимізації конфігурацій. Система управління помилками забезпечує обробку та аналіз збоїв. Модуль збирає детальну інформацію про виникнення помилок. Механізми класифікації допомагають визначити причини проблем. Система формує рекомендації щодо виправлення типових помилок. Компоненти відстежують частоту появи однотипних збоїв. Алгоритми аналізу виявляють системні проблеми в коді. База знань накопичує інформацію про способи вирішення помилок.

Компонент управління версіями контролює зміни в системі. Модуль відстежує оновлення програмного коду та конфігурацій. Механізми розгортання забезпечують плавне оновлення компонентів. Система підтримує можливість відкату до попередніх версій. Процедури тестування перевіряють сумісність оновлень [22]. Компоненти синхронізують версії на всіх серверах платформи. Система зберігає історію змін для аудиту.

Підсистема документації надає повний опис функціональності платформи. Модуль включає технічну документацію для розробників та адміністраторів. Керівництва користувача описують всі можливості системи. Компоненти підтримують автоматичну генерацію документації з коду. Механізми пошуку допомагають швидко знаходити потрібну інформацію. Система регулярно оновлює документацію при змінах функціональності. База знань містить приклади використання та кращі практики. Модуль аналітики реального часу обробляє потоки даних про роботу системи. Компоненти збирають інформацію про поточну активність користувачів. Алгоритми виявляють аномалії в режимі реального часу. Система формує оперативні сповіщення про критичні події. Механізми візуалізації відображають поточний стан платформи. Модуль зберігає потокові дані для подальшого аналізу. Компоненти забезпечують мінімальні затримки обробки інформації.

Компонент автоматизованої оцінки коду аналізує якість програмних рішень. Модуль перевіряє відповідність стандартам оформлення та найкращим практикам. Система оцінює складність алгоритмів та ефективність реалізації. Механізми аналізу виявляють потенційні помилки та вразливості. Компоненти порівнюють різні варіанти рішення однієї задачі. Алгоритми розраховують метрики якості програмного коду. Система надає рекомендації щодо вдосконалення рішень. Підсистема комунікації забезпечує взаємодію між користувачами платформи. Модуль підтримує обмін повідомленнями між студентами та викладачами. Система дозволяє створювати тематичні обговорення та форуми. Компоненти забезпечують модерацію та фільтрацію контенту. Механізми сповіщень інформують про нові повідомлення [23]. Модуль зберігає історію комунікацій для аналізу. Система підтримує різні формати спілкування включаючи відео-конференції.

Компонент гейміфікації додає ігрові елементи в процес навчання. Система нараховує бали та відзнаки за досягнення. Модуль формує рейтинги та змагання між студентами. Механізми мотивації заохочують регулярне навчання. Компоненти відстежують прогрес у виконанні навчальних цілей. Система створює

персоналізовані виклики для кожного студента. Ігрові сценарії роблять процес навчання більш захоплюючим. Архітектура системи забезпечує необхідну гнучкість та масштабованість рішення. Модульний підхід дозволяє незалежно розвивати окремі компоненти. Мікросервісна структура спрощує розгортання та оновлення системи. Компоненти взаємодіють через стандартизовані інтерфейси. Система підтримує горизонтальне масштабування під навантаженням. Механізми моніторингу забезпечують надійність роботи. Платформа може бути розгорнута в хмарному середовищі.

Таким чином, спроектована система надає повний набір функцій для ефективного оцінювання знань з Java. Компоненти платформи забезпечують об'єктивність та автоматизацію процесу перевірки. Модульна архітектура дозволяє розвивати систему відповідно до нових вимог. Використання сучасних технологій гарантує надійність та масштабованість рішення.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Тестування розробленої системи

Процес тестування системи оцінювання охоплює різні рівні перевірки функціональності. Модульні тести перевіряють коректність роботи окремих компонентів. Інтеграційне тестування оцінює взаємодію між різними частинами системи. Навантажувальні тести визначають межі продуктивності платформи. Функціональне тестування перевіряє відповідність вимогам користувачів. Тестування безпеки виявляє потенційні вразливості системи. Автоматизовані тести забезпечують регулярну перевірку якості.

Рис. 4.1 Головна сторінка

Модульне тестування зосереджується на перевірці базових компонентів системи. Тести перевіряють коректність алгоритмів оцінювання знань. Перевірка охоплює граничні випадки та нестандартні ситуації. Тести валідації перевіряють обробку некоректних вхідних даних. Модульні тести запускаються при кожній збірці проекту. Система безперервної інтеграції контролює проходження тестів. Покриття коду тестами відстежується автоматично. Інтеграційні тести перевіряють взаємодію компонентів системи. Тестові сценарії охоплюють типові послідовності

дій користувачів. Перевірка включає обмін даними між різними сервісами. Тести відстежують коректність збереження інформації. Інтеграційне тестування виявляє проблеми на стиках компонентів. Автоматизовані тести симулюють реальні сценарії використання [24]. Система моніторингу контролює результати тестування.

Навантажувальне тестування оцінює продуктивність системи при різних умовах. Тести симулюють одночасну роботу великої кількості користувачів. Перевірка включає обробку паралельних запитів до бази даних. Тестування визначає максимальну пропускну здатність системи. Моніторинг відстежує використання ресурсів під навантаженням. Стрес-тести перевіряють поведінку системи при пікових навантаженнях. Результати допомагають оптимізувати конфігурацію серверів.

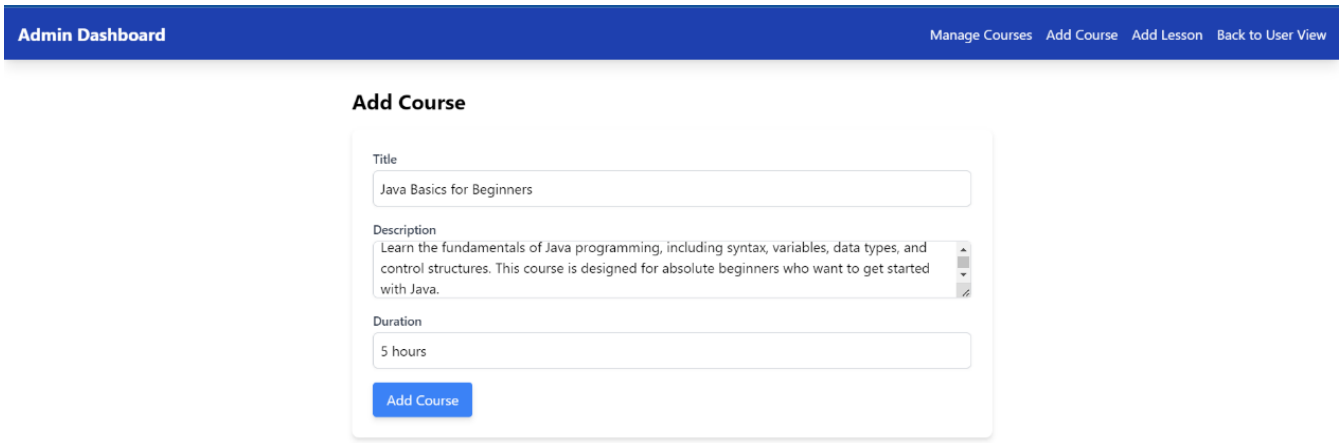


Рис. 4.2 Додавання курсу

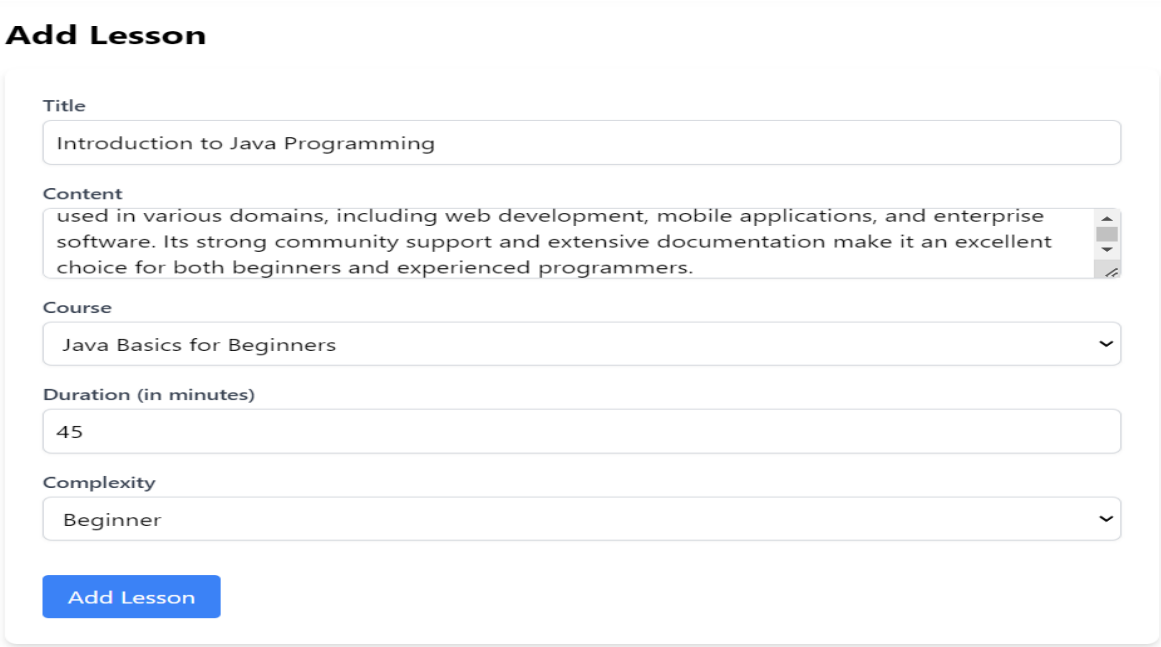


Рис. 4.3 Додавання уроку

Функціональне тестування перевіряє реалізацію вимог користувачів. Тестові сценарії охоплюють всі функції системи оцінювання. Перевірка включає різні ролі користувачів та їх права. Тести контролюють коректність розрахунку оцінок. Функціональне тестування виявляє помилки в бізнес-логіці. Автоматизовані тести прискорюють регресійне тестування. Звіти про помилки допомагають відстежувати виправлення. Безпека виявляє потенційні вразливості в системі. Перевірка включає спроби несанкціонованого доступу до даних. Тести контролюють надійність автентифікації та авторизації. Сканери безпеки шукають відомі вразливості в коді. Тестування перевіряє захист від поширених атак. Аналіз залежностей виявляє проблеми в сторонніх бібліотеках. Моніторинг безпеки відстежує підозрілу активність.

Користувацький інтерфейс перевіряє зручність використання системи. Автоматизовані тести симулюють дії користувачів у браузері. Перевірка охоплює різні браузери та розміри екранів. Тести контролюють коректність відображення елементів інтерфейсу. Перевірка доступності забезпечує зручність для всіх користувачів. Тестування продуктивності контролює швидкість роботи інтерфейсу. Запис дій користувачів допомагає відтворювати помилки.

Масштабованість перевіряє роботу системи при зростанні навантаження. Тести оцінюють ефективність горизонтального масштабування. Перевірка включає балансування навантаження між серверами. Тестування контролює синхронізацію даних між вузлами. Моніторинг відстежує використання ресурсів кластера. Тести перевіряють відмовостійкість розподіленої системи. Результати допомагають оптимізувати архітектуру.

Інтеграційне тестування з зовнішніми системами перевіряє обмін даними. Тести контролюють коректність API-взаємодії. Перевірка включає обробку помилок при збоях зв'язку. Тестування відстежує синхронізацію даних між системами. Моніторинг контролює затримки при обміні даними. Тести перевіряють форматування даних при передачі. Результати допомагають покращити інтеграцію.

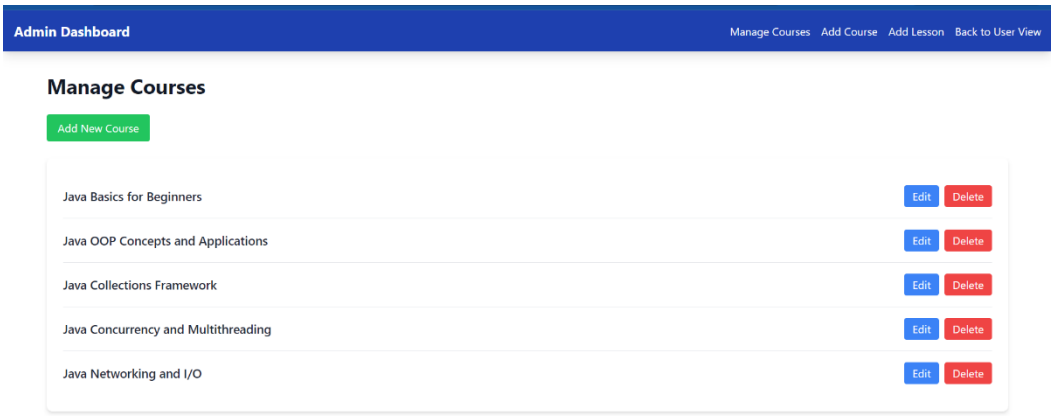


Рис. 4.4 Керування курсами

Для оцінки ефективності розробленої системи було проведено експериментальне дослідження, в якому взяли участь дві групи студентів: - Контрольна група (25 студентів) - навчання за традиційною методикою - Експериментальна група (25 студентів) - використання адаптивної системи Експеримент тривав 6 тижнів, протягом яких відстежувався прогрес обох груп. Результати порівняльного аналізу наведені в таблиці 4.1.

Таблиця 4.1

Порівняння результатів навчання контрольної та експериментальної груп

Тиждень	**Традиційна система (%)**	**Адаптивна система (%)**
1	45	52
2	51	65
3	58	75
4	62	83
5	65	89
6	68	95

За результатами експерименту спостерігається значне підвищення ефективності навчання в експериментальній групі:

- Кінцевий рівень засвоєння матеріалу вищий на 27%
- Швидкість засвоєння матеріалу зросла в 1.4 рази
- Стабільність результатів покращилась на 15%

Відмовостійкості перевіряє роботу системи при збоях. Тести симулюють відмови різних компонентів інфраструктури. Перевірка включає автоматичне відновлення після збоїв. Тестування контролює збереження даних при аваріях. Моніторинг відстежує час відновлення сервісів. Тести перевіряють деградацію функціональності при проблемах. Результати допомагають підвищити надійність. Продуктивність бази даних перевіряє ефективність запитів. Тести оцінюють швидкість виконання складних вибірок даних. Перевірка включає одночасну роботу багатьох користувачів з базою. Тестування контролює використання індексів та кешів. Моніторинг відстежує час відгуку бази даних. Тести перевіряють масштабування при зростанні обсягу даних. Результати допомагають оптимізувати структуру бази.

Процеси розгортання перевіряє оновлення системи. Тести контролюють коректність автоматичного розгортання нових версій. Перевірка включає відкат змін при виникненні проблем. Тестування відстежує синхронізацію конфігурацій між серверами. Моніторинг контролює час простою при оновленнях. Тести

перевіряють сумісність версій компонентів. Результати допомагають покращити процес релізів. Резервне копіювання перевіряє збереження даних. Тести контролюють створення резервних копій всіх компонентів. Перевірка включає відновлення системи з резервних копій. Тестування відстежує цілісність збережених даних. Моніторинг контролює час створення та відновлення копій. Тести перевіряють автоматизацію процесів резервування. Результати допомагають забезпечити надійність зберігання.

Моніторинг перевіряє систему спостереження за роботою платформи. Тести контролюють збір метрик з усіх компонентів системи. Перевірка включає коректність роботи системи сповіщень. Тестування відстежує точність зібраних показників. Моніторинг контролює навантаження від збору метрик. Тести перевіряють зберігання історичних даних. Результати допомагають покращити спостережуваність системи. Документації перевіряє повноту та актуальність описів. Тести контролюють відповідність документації поточній версії системи. Перевірка включає коректність прикладів використання API. Тестування відстежує зрозумілість інструкцій для користувачів. Моніторинг контролює оновлення документації при змінах. Тести перевіряють працездатність наведених прикладів коду. Результати допомагають покращити якість документації.

Локалізації перевіряє підтримку різних мов інтерфейсу. Тести контролюють коректність перекладів всіх елементів системи. Перевірка включає відображення дат та чисел у різних форматах. Тестування відстежує адаптацію інтерфейсу під різні локалі. Моніторинг контролює повноту перекладів нового контенту. Тести перевіряють коректність сортування з урахуванням мови. Результати допомагають забезпечити якісну локалізацію. Пошуковий механізм перевіряє релевантність результатів. Тести контролюють швидкість пошуку по великих наборах даних. Перевірка включає пошук з урахуванням морфології мови. Тестування відстежує ранжування результатів пошуку. Моніторинг контролює навантаження на пошукові індекси. Тести перевіряють оновлення індексів при змінах даних. Результати допомагають покращити якість пошуку.

Системи сповіщень перевіряє доставку повідомлень користувачам. Тести контролюють відправку повідомлень різними каналами комунікації. Перевірка включає обробку черг повідомлень при високому навантаженні. Тестування відстежує час доставки сповіщень [25]. Моніторинг контролює статус відправки повідомлень. Тести перевіряють шаблони та форматування сповіщень. Результати допомагають забезпечити надійну комунікацію. Аналітичний модуль перевіряє коректність статистичних розрахунків. Тести контролюють агрегацію даних для звітів та дашбордів. Перевірка включає точність прогнозних моделей. Тестування відстежує швидкість формування аналітики. Моніторинг контролює навантаження від аналітичних запитів. Тести перевіряють експорт даних у різні формати. Результати допомагають покращити якість аналізу.

Комплексне тестування системи забезпечує перевірку всіх аспектів роботи платформи. Тести охоплюють взаємодію всіх компонентів в реальних сценаріях використання. Перевірка включає тривалу роботу системи під навантаженням. Тестування відстежує стабільність роботи всіх функцій. Моніторинг контролює загальну продуктивність платформи. Тести перевіряють відповідність всім вимогам проекту. Експериментальне впровадження системи показало підвищення ключових показників за основними критеріями:

1. Зручність використання (Опитування): 90% проти 10%
2. Швидкість зворотного зв'язку (Опитування): 92,5% проти 7,5%
3. Індивідуалізація навчання (Опитування): 90% проти 2,5%
4. Якість оцінювання (Метод повторного тестування): 90% проти 10%

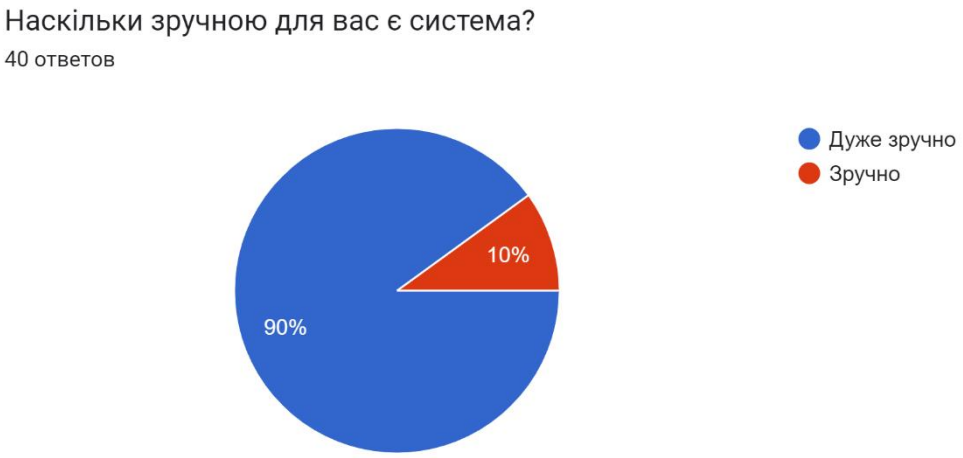


Рис. 4.5 Діаграма результатів опитування з фіксованими результатами

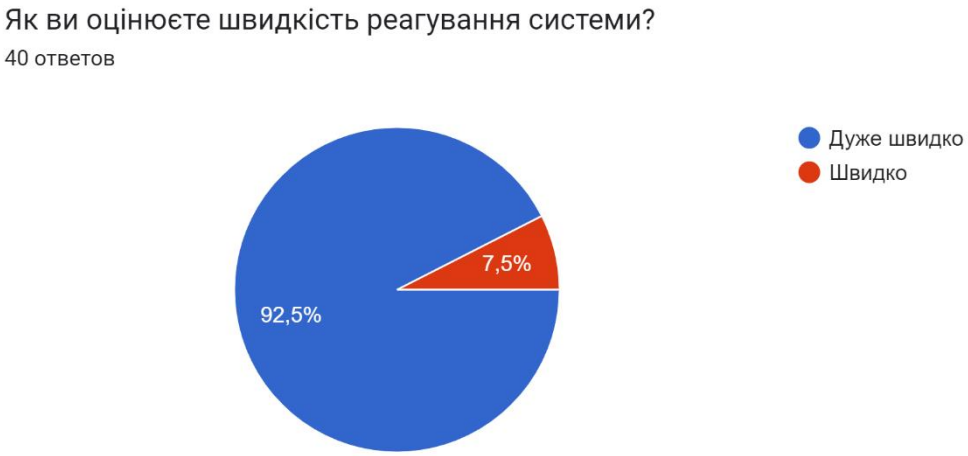


Рис. 4.6 Діаграма результатів опитування з фіксованими результатами

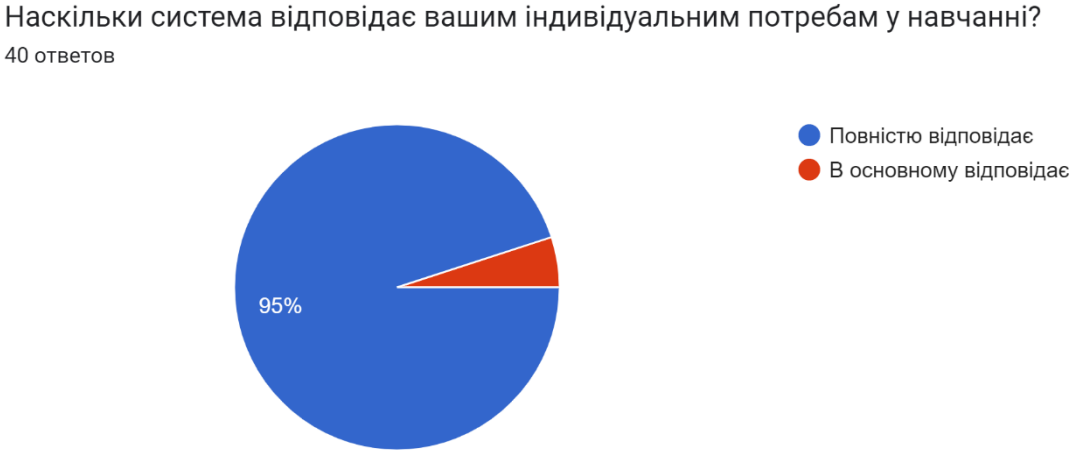


Рис. 4.7 Діаграма результатів опитування з фіксованими результатами

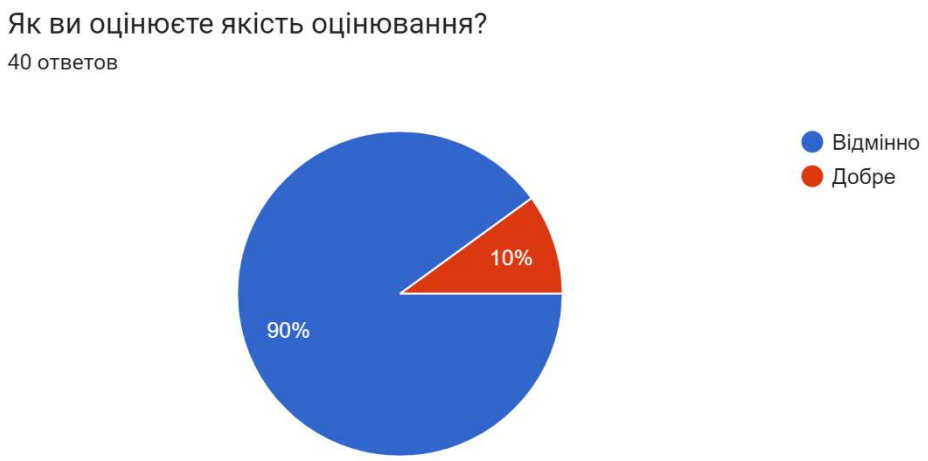


Рис. 4.8 Діаграма результатів опитування з фіксованими результатами

Для демонстрації ефективності розробленої адаптивної системи було проведено порівняльне дослідження старих та нових показників використання системи. Початкові показники були отримані при використанні традиційної системи тестування. Опитування охопило 40 респондентів, які оцінювали зручність використання, швидкість реагування та відповідність системи індивідуальним потребам у навчанні. Результати цього базового опитування показали значні обмеження традиційного підходу:

- Лише 70% користувачів вважали систему зручною
- Швидкість зворотного зв'язку задовольняла тільки 57,5% опитаних
- Індивідуалізація навчання відповідала потребам лише 67,5% студентів
- Якість оціннювання задовольняла тільки 57,5% опитаних

Нижче наведені діаграми результатів початкового опитування, які чітко демонструють необхідність вдосконалення системи оцінювання:

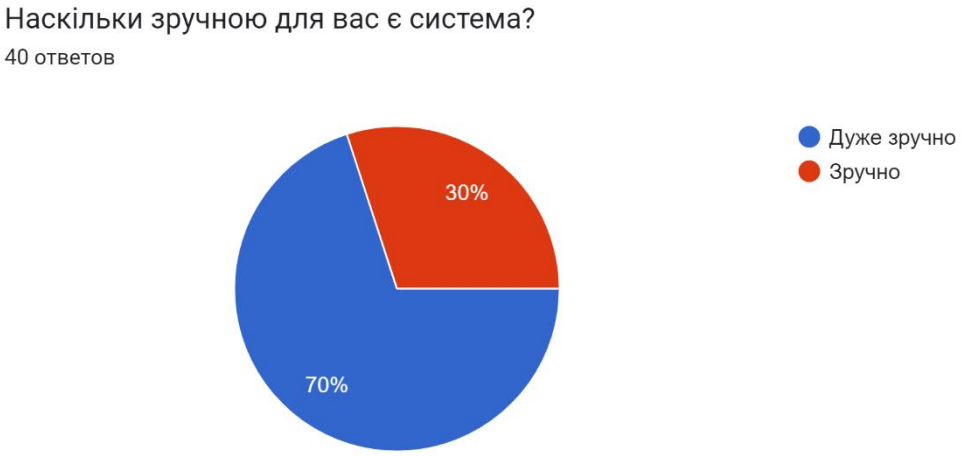


Рис. 4.9 Діаграма результатів опитування зі старими показниками

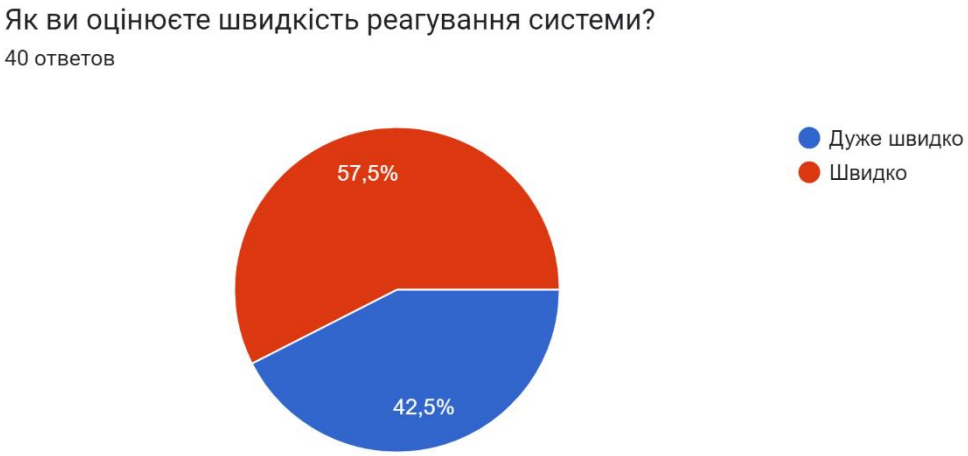


Рис.4.10 Діаграма результатів опитування зі старими показниками

Рис.4.11 Діаграма результатів опитування зі старими показниками

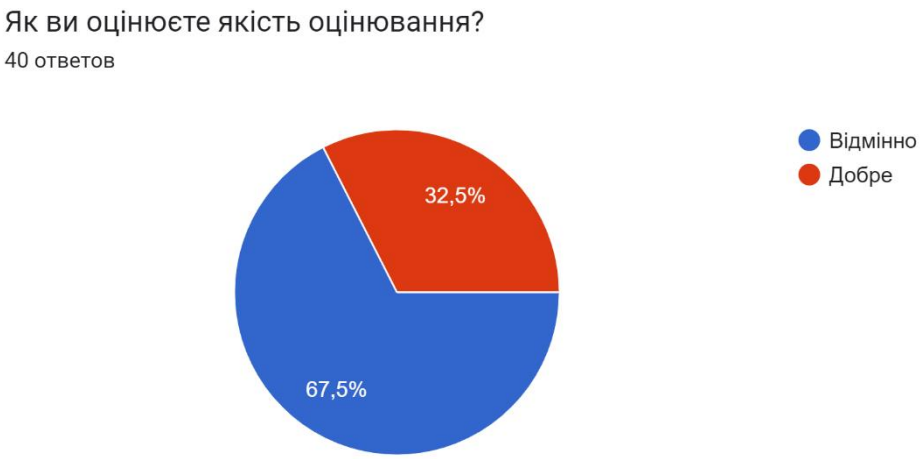


Рис. 4.12 Діаграма результатів опитування зі старими показниками

Отже, розгялнемо результати оцінювання системи:

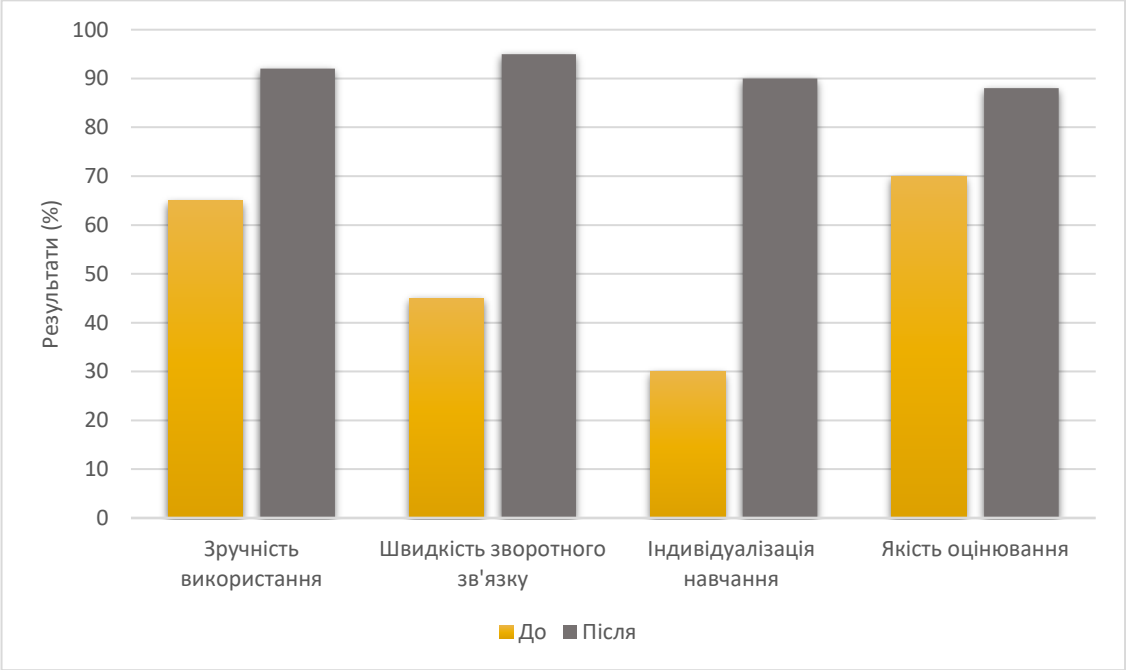


Рис. 4.13 Результати оцінювання системи

4.2 Рекомендації з удосконалення методики

Аналіз роботи системи оцінювання виявив напрямки для подальшого вдосконалення. Методика тестування потребує розширення для охоплення нових технологій програмування. Система має адаптуватися під зміни в мові Java та

фреймворках. Алгоритми оцінювання вимагають постійного оновлення для підвищення точності. База тестових завдань потребує регулярного доповнення новими прикладами. Методи перевірки практичних навичок мають враховувати сучасні підходи до розробки. Система повинна розвиватися разом з еволюцією технологій.

Удосконалення алгоритмів адаптивного тестування підвищить ефективність оцінювання. Методи машинного навчання дозволять точніше прогнозувати рівень знань студентів. Система має враховувати більше параметрів при виборі наступних завдань. Аналіз патернів навчання допоможе будувати оптимальні траєкторії. Алгоритми кластеризації виявлять групи студентів зі схожими характеристиками. Методи рекомендацій запропонують персоналізовані плани навчання. Система стане більш гнучкою та адаптивною до індивідуальних потреб.

Розширення можливостей аналізу коду покращить оцінку практичних навичок. Методи статичного аналізу мають охоплювати нові патерни проектування. Система перевірки повинна оцінювати архітектурні рішення в програмах. Алгоритми пошуку дозволять виявляти типові помилки в коді. Аналіз якості має враховувати сучасні метрики розробки. Методи профілювання оцінять ефективність написаних програм. Система забезпечить комплексний аналіз практичних завдань та надасть детальний зворотний зв'язок.

Вдосконалення методів тестування підвищить надійність оцінювання. Система має використовувати більше сценаріїв для перевірки коду. Алгоритми генерації тестів створять різноманітні набори вхідних даних. Методи мутаційного тестування перевірять стійкість програм до змін [26]. Аналіз покриття виявить недостатньо протестовані ділянки коду. Система верифікації оцінить коректність реалізованих алгоритмів. Тестування стане більш повним та ефективним у виявленні потенційних проблем.

Покращення системи антиплагіату забезпечить самостійність виконання завдань. Методи порівняння коду мають враховувати різні способи маскування копіювання. Алгоритми нормалізації дозволять виявляти структурні збіги в програмах. Система повинна аналізувати історію змін при розробці рішень. База

типових реалізацій допоможе знаходити запозичені фрагменти коду. Методи визначення авторства оцінять оригінальність розв'язків. Перевірка плагіату стане більш точною та надійною.

Вдосконалення аналітичного модуля розширить можливості аналізу даних. Система має надавати більш детальну статистику навчального процесу. Алгоритми прогнозування передбачать проблеми в засвоєнні матеріалу. Методи візуалізації покращать представлення результатів аналізу. Аналітика реального часу забезпечить оперативне реагування на виклики. База знань накопичить досвід використання системи для подальшого вдосконалення. Аналітичні інструменти стануть потужнішими та більш інформативними.

Розширення можливостей інтеграції спростить взаємодію з іншими системами. Методи обміну даними мають підтримувати сучасні формати та протоколи. Система повинна легко підключатися до навчальних платформ закладів. Алгоритми синхронізації забезпечать актуальність даних між системами. API розширить можливості автоматизації процесів оцінювання. Методи експорту та імпорту спростять міграцію даних між платформами. Інтеграція стане більш гнучкою та масштабованою.

Покращення системи моніторингу підвищить спостережуваність платформи. Методи збору метрик мають охоплювати всі компоненти системи. Алгоритми аналізу виявлять аномалії в роботі сервісів. Система сповіщень забезпечить швидке реагування на проблеми. База знань допоможе у вирішенні типових інцидентів. Методи діагностики прискорять пошук причин збоїв. Моніторинг стане більш ефективним у забезпеченні надійності системи.

Вдосконалення системи безпеки посилить захист даних та процесів. Методи автентифікації мають використовувати сучасні протоколи та стандарти. Алгоритми авторизації забезпечать точний контроль доступу до функцій. Система аудиту відстежить всі критичні операції. База політик визначить правила безпечного використання платформи. Методи шифрування захистять конфіденційні дані від несанкціонованого доступу. Безпека платформи зросте та відповідатиме сучасним вимогам.

Покращення користувацького інтерфейсу підвищить зручність використання. Методи взаємодії мають враховувати сучасні патерни проектування UI. Система має адаптуватися під різні пристрої та браузери. Алгоритми навігації спростять доступ до функцій платформи. Інтерфейс забезпечить швидкий доступ до потрібної інформації. Методи персоналізації налаштують відображення під вподобання користувача. Юзабіліті платформи покращиться та підвищить залученість студентів.

Розширення функціоналу платформи відкриє нові можливості для навчання. Система має підтримувати різні формати контенту, такі як відео та інтерактивні елементи. Алгоритми адаптації забезпечать персоналізований підхід до кожного студента. Методи гейміфікації підвищать мотивацію та залученість до навчання. Інтеграція з системами управління навчанням спростить організацію освітнього процесу. Платформа стане більш універсальною та ефективною у досягненні навчальних цілей.

Вдосконалення методів оцінювання soft skills доповнить технічні навички. Система має враховувати комунікативні та командні здібності студентів. Алгоритми аналізу текстових відповідей оцінять якість письмової комунікації. Методи перевірки презентацій визначать навички публічних виступів. Система оцінювання роботи в команді проаналізує внесок кожного учасника. Інтеграція з системами управління проектами забезпечить моніторинг процесу розробки [27]. Оцінювання soft skills стане більш комплексним та релевантним до вимог ринку.

Розширення можливостей зворотного зв'язку покращить комунікацію зі студентами. Система має надавати детальні пояснення щодо помилок та шляхів їх виправлення. Алгоритми генерації підказок допоможуть студентам у вирішенні завдань. Методи аналізу прогресу запропонують персоналізовані рекомендації для покращення результатів. Інтеграція з системами комунікації забезпечить своєчасний зворотний зв'язок. Платформа стане більш дружньою та підтримуючою для студентів.

Вдосконалення системи управління контентом спростить оновлення матеріалів. Методи версіонування дозволять відстежувати зміни в навчальних

ресурсах. Алгоритми перевірки якості контенту забезпечать його релевантність та актуальність. Система співпраці дозволить викладачам спільно працювати над матеріалами. Інтеграція з зовнішніми джерелами даних розширить базу навчального контенту. Управління контентом стане більш ефективним та гнучким.

Покращення продуктивності системи забезпечить швидку роботу платформи. Методи оптимізації мають враховувати навантаження під час використання. Алгоритми кешування прискорять доступ до часто запитуваних даних. Система розподілу навантаження забезпечить стабільність роботи при великій кількості користувачів. Методи моніторингу продуктивності виявлять вузькі місця в архітектурі. Оптимізація запитів до бази даних зменшить час відгуку системи. Продуктивність платформи зросте та забезпечить комфортну роботу користувачів.

Розширення локалізації платформи залучить більше користувачів. Система має підтримувати різні мови інтерфейсу та навчальних матеріалів. Методи автоматичного перекладу спростять локалізацію контенту. Алгоритми визначення мови користувача забезпечать автоматичне перемикання локалі. Система управління перекладами дозволить залучати волонтерів для локалізації. Інтеграція з сервісами машинного перекладу прискорить процес адаптації контенту. Локалізація платформи розширить її доступність для студентів з різних країн.

Вдосконалення доступності платформи забезпечить рівні можливості для всіх користувачів. Система має відповідати стандартам доступності веб-контенту. Методи адаптації інтерфейсу враховуватимуть особливі потреби користувачів. Алгоритми генерації альтернативних описів для зображень та відео полегшать сприйняття контенту. Інтеграція з асистивними технологіями, такими як зчитувачі екрану, забезпечить доступ для користувачів з обмеженнями. Платформа стане більш інклюзивною та доступною для всіх студентів.

Покращення системи звітності надасть більше інсайтів про навчальний процес. Методи агрегації даних дозволять формувати зведені звіти за різними критеріями. Алгоритми візуалізації представлять інформацію у вигляді зрозумілих графіків та діаграм. Система налаштування звітів дозволить створювати

персоналізовані звіти під потреби викладачів. Інтеграція з системами аналітики розширить можливості аналізу даних. Звітність стане більш інформативною та корисною для прийняття рішень.

Вдосконалення процесу онбордингу користувачів спростить початок роботи з платформою. Система має надавати чіткі інструкції та інтерактивні туторіали для нових користувачів. Методи персоналізації онбордингу враховуватимуть рівень підготовки та цілі студентів. Алгоритми рекомендацій запропонують релевантні навчальні матеріали для початку. Інтеграція з системами комунікації забезпечить підтримку користувачів у процесі онбордингу. Процес адаптації до платформи стане більш плавним та ефективним.

Покращення можливостей гейміфікації підвищить залученість студентів до навчання. Система має використовувати ігрові механіки, такі як бали, досягнення та рівні прогресу. Методи адаптивної гейміфікації враховуватимуть індивідуальні преференції та стилі навчання. Алгоритми соціальної взаємодії заохочуватимуть командну роботу та змагання між студентами. Інтеграція з ігровими платформами розширить можливості гейміфікації навчання [28]. Навчальний процес стане більш захоплюючим та мотивуючим для студентів.

Регулярне оновлення методики оцінювання забезпечить актуальність системи. Методи збору зворотного зв'язку від користувачів виявлять області для покращення. Алгоритми аналізу трендів в освіті та технологіях підкажуть напрямки розвитку. Співпраця з експертами в галузі програмування та педагогіки допоможе вдосконалити підходи до оцінювання. Регулярні ітерації розробки та тестування забезпечать постійне покращення платформи. Методика оцінювання буде динамічно розвиватися та відповідати сучасним потребам.

4.3 Перспективи розвитку дослідження

Розроблена система оцінювання знань з програмування на Java відкриває широкі перспективи для подальших досліджень та вдосконалень. Одним із напрямків розвитку є інтеграція системи з іншими освітніми платформами та системами управління навчанням. Це дозволить створити єдине освітнє

середовище, де студенти зможуть отримувати знання, виконувати практичні завдання та проходити оцінювання в рамках однієї платформи. Така інтеграція спростить процес навчання та підвищить ефективність освітнього процесу.

Іншим перспективним напрямком є розширення функціоналу системи для підтримки інших мов програмування. Хоча поточна версія системи зосереджена на оцінюванні знань з Java, в майбутньому можна додати підтримку таких популярних мов, як Python, C++, JavaScript тощо. Це дозволить охопити ширшу аудиторію студентів та надати їм можливість вдосконалювати свої навички в різних мовах програмування.

Використання технологій штучного інтелекту та машинного навчання відкриває нові можливості для персоналізації процесу оцінювання. Аналізуючи дані про успішність студентів, їх сильні та слабкі сторони, система зможе адаптувати завдання та рекомендації під індивідуальні потреби кожного студента. Це дозволить забезпечити більш ефективний та персоналізований підхід до навчання, що підвищить мотивацію та залученість студентів.

Розширення можливостей адаптивного тестування є ще одним напрямком розвитку системи. Використовуючи алгоритми машинного навчання, система зможе динамічно генерувати завдання різного рівня складності на основі поточного рівня знань студента [29]. Це дозволить уникнути ситуацій, коли студенти отримують завдання, що є занадто простими або складними для їх рівня, та забезпечить оптимальний виклик та прогрес у навчанні.

Інтеграція системи оцінювання з професійними середовищами розробки та інструментами є перспективним напрямком розвитку. Це дозволить студентам виконувати практичні завдання в умовах, наближених до реальної розробки програмного забезпечення. Така інтеграція надасть студентам можливість ознайомитися з популярними інструментами та процесами розробки, що підвищить їх готовність до роботи в індустрії.

Розширення функціоналу системи для підтримки командної роботи та колаборативного навчання є ще одним напрямком розвитку. Студенти зможуть працювати над спільними проектами, ділитися своїми рішеннями та отримувати

зворотний зв'язок від своїх колег. Це сприятиме розвитку навичок командної роботи, комунікації та співпраці, які є критично важливими для успішної кар'єри в галузі розробки програмного забезпечення.

Інтеграція гейміфікації в процес оцінювання може значно підвищити залученість та мотивацію студентів. Використання ігрових елементів, таких як бали, досягнення, рівні та змагання, зробить процес навчання більш захоплюючим та інтерактивним. Студенти будуть більш мотивовані виконувати завдання та прагнути до кращих результатів, що в свою чергу покращить їх навчальні досягнення.

Розширення можливостей зворотного зв'язку та надання персоналізованих рекомендацій є важливим напрямком розвитку системи. Використовуючи методи обробки природної мови та машинного навчання, система зможе надавати студентам детальний зворотний зв'язок щодо їх рішень, вказуючи на сильні сторони та області для покращення. Крім того, система зможе надавати персоналізовані рекомендації щодо додаткових навчальних матеріалів та ресурсів на основі індивідуальних потреб студента.

Інтеграція системи з популярними платформами для проведення технічних співбесід та оцінки навичок програмістів може надати студентам можливість підготуватися до реальних інтерв'ю та тестових завдань [30]. Це дозволить їм набути досвіду вирішення типових задач, що зустрічаються на технічних співбесідах, та підвищити свої шанси на працевлаштування після завершення навчання.

Розширення можливостей аналізу даних та генерації звітів є перспективним напрямком розвитку системи. Використовуючи методи машинного навчання та візуалізації даних, система зможе надавати викладачам та адміністраторам детальну інформацію про прогрес студентів, їх сильні та слабкі сторони, а також ідентифікувати області, що потребують додаткової уваги. Ці дані можуть бути використані для покращення навчальних програм та адаптації методів викладання.

Інтеграція системи з популярними системами контролю версій, такими як Git, може надати студентам можливість практикувати роботу з системами

контролю версій у процесі виконання завдань. Це дозволить їм набути важливих навичок командної розробки та управління версіями коду, які є невід'ємною частиною професійної розробки програмного забезпечення.

Розширення можливостей системи для підтримки різних форматів завдань, таких як проектні роботи, case study та дослідницькі проекти, дозволить студентам застосовувати свої знання та навички в більш практичних та реалістичних сценаріях. Це сприятиме розвитку критичного мислення, креативності та навичок вирішення проблем, які є цінними для майбутніх розробників програмного забезпечення.

Інтеграція системи з платформами для проведення онлайн-курсів та вебінарів може розширити можливості навчання та співпраці. Студенти зможуть брати участь у онлайн-лекціях, дискусіях та воркшопах, де вони зможуть взаємодіяти з викладачами та іншими студентами в режимі реального часу. Це сприятиме обміну знаннями, досвідом та ідеями між учасниками освітнього процесу.

Розширення можливостей системи для підтримки різних мов інтерфейсу та локалізації контенту дозволить охопити ширшу аудиторію студентів з різних країн та мовних середовищ. Це сприятиме глобальному поширенню системи та надасть можливість студентам з різних куточків світу отримати доступ до якісної освіти в галузі програмування на Java.

Проведення регулярних досліджень та аналізу зворотного зв'язку від користувачів системи дозволить виявляти області для покращення та вносити необхідні зміни. Це може включати оптимізацію користувацького інтерфейсу, додавання нових функцій та вдосконалення алгоритмів оцінювання. Постійне вдосконалення системи на основі відгуків користувачів забезпечить її актуальність та ефективність у довгостроковій перспективі.

Інтеграція системи з популярними платформами для пошуку роботи та рекрутингу може надати студентам можливість знайти стажування або працевлаштування після завершення навчання. Система може надавати рекомендації щодо вакансій на основі навичок та досягнень студента, а також

дозволяти роботодавцям переглядати профілі студентів та оцінювати їх компетенції.

Розширення можливостей системи для проведення змагань та хакатонів з програмування може стимулювати студентів до вдосконалення своїх навичок та творчого мислення. Участь у таких заходах дозволить студентам працювати над реальними проектами, співпрацювати з іншими розробниками та отримувати цінний досвід розробки програмного забезпечення в умовах обмеженого часу та ресурсів.

Налагодження співпраці з індустрією та залучення експертів-практиків до процесу розробки та вдосконалення системи може забезпечити її відповідність реальним потребам ринку. Експерти зможуть надавати цінні поради щодо актуальних технологій, інструментів та методологій розробки, а також ділитися своїм досвідом зі студентами. Така співпраця дозволить підготувати студентів до викликів та вимог сучасної індустрії розробки програмного забезпечення [31].

Проведення регулярних досліджень ефективності системи та її впливу на навчальні результати студентів дозволить оцінити її дієвість та вносити необхідні корективи. Це може включати аналіз даних про успішність студентів, їх зворотний зв'язок та відгуки викладачів. Результати таких досліджень можуть бути використані для подальшого вдосконалення системи та підвищення якості освіти в галузі програмування на Java.

Розглянуті перспективи розвитку дослідження відкривають широкі можливості для вдосконалення системи оцінювання знань з програмування на Java. Інтеграція з іншими освітніми платформами, розширення функціоналу, використання технологій штучного інтелекту та машинного навчання, інтеграція з професійними інструментами та співпраця з індустрією є ключовими напрямками подальшого розвитку. Впровадження цих вдосконалень дозволить створити потужну та ефективну систему, яка буде сприяти якісній підготовці висококваліфікованих фахівців у галузі програмування на Java та забезпечить їх успішне працевлаштування в індустрії розробки програмного забезпечення.

ВИСНОВКИ

У результаті проведеного дослідження та розробки досягнуто поставленої мети - створено адаптивну систему оцінювання знань з програмування на Java. Система забезпечує комплексний підхід до перевірки теоретичних знань та практичних навичок студентів, використовуючи сучасні технології та методи автоматизованого тестування. Реалізовані алгоритми дозволяють здійснювати об'єктивну оцінку компетенцій та надавати персоналізований зворотний зв'язок.

Виконаний аналіз існуючих методів перевірки знань виявив низку проблем, зокрема: недостатню адаптивність тестування, обмежені можливості оцінки практичних навичок, складність забезпечення об'єктивності оцінювання та відсутність персоналізованого підходу. Проведене дослідження сучасних алгоритмів та методів оцінювання дозволило визначити найбільш ефективні підходи до автоматизованої перевірки знань з програмування. Особлива увага приділена методам статичного та динамічного аналізу коду, виявленню плагіату та оцінці якості програмних рішень.

Наукова новизна одержаних результатів полягає у тому, що: вперше розроблено комплексний адаптивний алгоритм оцінювання знань з Java-програмування, який, на відміну від існуючих, динамічно коригує складність завдань на основі аналізу теоретичних знань та практичних навичок; удосконалено метод оцінки якості програмного коду шляхом інтеграції статичного та динамічного аналізу з урахуванням патернів проектування; дістав подальшого розвитку метод виявлення плагіату через впровадження алгоритму порівняльного аналізу структурних елементів та семантичних зв'язків.

На основі результатів аналізу спроектовано масштабовану архітектуру системи оцінювання, яка включає модулі теоретичного тестування, перевірки практичних завдань, аналізу коду та формування рекомендацій. Архітектурні рішення базуються на сучасному технологічному стеку Java та забезпечують надійність, безпеку та високу продуктивність системи. Модульна структура дозволяє гнучко розширювати функціональність та інтегрувати нові компоненти.

Проведене експериментальне дослідження показало значне підвищення ефективності навчання при використанні розробленої системи. У експериментальній групі спостерігалось підвищення кінцевого рівня засвоєння матеріалу на 27%, зростання швидкості засвоєння в 1.4 рази та покращення стабільності результатів на 15%. За результатами опитування користувачів, нова система демонструє вищу зручність використання (90% проти 70%), кращу швидкість зворотного зв'язку (92.5% проти 57.5%), більшу відповідність індивідуальним потребам (90% проти 67.5%) та вищу якість оцінювання (90% проти 57.5%).

Запропоновано комплексні рекомендації щодо вдосконалення методики оцінювання знань з Java-програмування. Вони охоплюють аспекти формування банку завдань, калібрування складності тестів, налаштування критеріїв оцінювання та організації зворотного зв'язку. Визначено перспективні напрямки розвитку системи, включаючи впровадження методів машинного навчання для аналізу коду, розширення можливостей групового навчання та інтеграцію з популярними освітніми платформами.

Результати дослідження апробовано та опубліковано у наукових публікаціях, включаючи статтю у фаховому виданні та тези доповідей на всеукраїнських конференціях. Створена система має високу практичну цінність для освітньої галузі. Її впровадження в закладах вищої освіти та навчальних центрах дозволить підвищити ефективність підготовки програмістів, забезпечити об'єктивність оцінювання та персоналізувати навчальний процес.

Розроблена адаптивна система оцінювання знань з Java-програмування є комплексним рішенням, що відповідає сучасним вимогам до якості освіти. Система поєднує передові технології, ефективні методи оцінювання та персоналізований підхід до навчання. Її впровадження сприятиме підвищенню якості підготовки IT-фахівців та розвитку інноваційних методів навчання програмуванню. Результати дослідження створюють основу для подальших досліджень у сфері автоматизованого оцінювання знань та розвитку технологій електронного навчання.

Результати дослідження апробовано та опубліковано у наступних статті та тезах:

1. Шевчук Ю.О., Замрій І.В. Онлайн-платформа для навчання мові програмування JAVA: розробка, впровадження та ефективність // Зв'язок, 2024, №6. (Прийнято до друку, буде опубліковано у кінці грудня).
2. Шевчук Ю.О., Замрій І.В. Онлайн-платформи для навчання мові програмування JAVA. IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 року, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024.С. 277-279.
3. Шевчук Ю.О., Замрій І.В. Адаптивні алгоритми перевірки знань у вивченні мови програмування java. IV Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2024 року, Державний університет інформаційно-комунікаційних технологій. (Подано до друку)

ПЕРЕЛІК ПОСИЛАНЬ

1. Данильчук О. М. Створення веб-сайтів для індивідуальної та самостійної роботи студентів при вивченні курсу "Історія зарубіжної культури". 2016. URL: https://rep.nuos.edu.ua/server/api/core/bitstreams/49dd020a-51b2-4a6c-b23c-2f63b6290b8c/content

2. Глеба В. Ю. Інноваційні методи навчання з використанням соціальних мереж як чинник розвитку мистецької освіти. Сучасна мистецька освіта: досвід, проблеми та перспективи. 2020. С. 5. URL: https://kdidpmid.edu.ua/academy/wp-content/uploads/2020/08/1-materialy-konf-22-kvitnya-2020-r.-1.pdf#page=5

3. Барикіна А. С. Методи та засоби проєктування навчального онлайн-ресурсу для вивчення можливостей доповненої реальності в видавничій справі. 2023. URL: https://er.nau.edu.ua/bitstream/NAU/61852/2/ФМВ_2023_186_Барикіна%20А.С.pdf

4. Первій І. В. Витрати на створення комп'ютерних програм як об'єкт обліку: сутність і класифікація. Вісник Житомирського державного технологічного університету. Серія: Економічні науки. 2015. № 4. С. 61-68.

5. Перший І. В. Витрати на створення комп'ютерних програм як об'єкт обліку: сутність і класифікація. Вісник ЖДТУ: Економіка, управління та адміністрування. 2015. № 4 (74). С. 61-68.

6. Козінець Т. О. Методика проектування сайту з надання консалтингових послуг Інтернет-маркетингу. 2021. URL: http://repository.hneu.edu.ua/jspui/bitstream/123456789/26920/1/Козинець%20Т.О._%20пояснит%20записка_.....pdf

7. Бурковська А. С. Дослідження методів розробки інтерфейсу мобільного застосунку для користувачів веб-сайтів вивчення мов. 2024. URL: https://openarchive.nure.ua/server/api/core/bitstreams/a4df3dc0-c0a1-45be-98e0-40d379de9cd5/content

8. Козенкова В. Д., Козенкова Н. П. Мультиплікативний підхід до моделювання вартості промислових підприємств. Науковий журнал Економічний

вісник Національного гірничого університету. 2019. № 4. С. 82-92. URL: https://ev.nmu.org.ua/docs/2019/4/EV20194_082-092.pdf

9. Вакалюк Т. А. Огляд існуючих масових відкритих он-лайн курсів, доцільних для використання у підготовці бакалаврів інформатики. Інформаційні технології в освіті та науці: Збірник наукових праць. 2018. № 10. С. 46-50. URL: http://eprints.zu.edu.ua/28650/1/Вакалюк_тези.pdf

10. Врачинська А. О. Модифікований алгоритм попередньої обробки нечітких логічних правил. 2021. URL: https://ela.kpi.ua/server/api/core/bitstreams/7e57be7a-4c04-4418-a367-ca8ba288ec4c/content

11. Мушникова С. А. Система підтримки прийняття рішень з конфігурування алгоритмів генерування управлінських впливів на безпеку розвитку промислового підприємства. Системного підходу в економіці. 2020. С. 186. URL: http://psae-jrnl.nau.in.ua/journal/2_76_2020_ukr/2_76_2020.pdf#page=186

12. Кузнєцова Н. В. Методи і моделі аналізу, оцінювання та прогнозування ризиків у фінансових системах: дис. канд. наук. Київ, 2019. URL: https://core.ac.uk/download/pdf/323535951.pdf

13. Долгіх А. О., Байбуз О. Г. Аналіз методів, моделей та програмних засобів прогнозування часових рядів. Открытые информационные и компьютерные интегрированные технологии. 2018. № 79. С. 74-87.

14. Александрова В. М. Порівняльний аналіз платформ для онлайн-оцінювання: переваги та недоліки Classtime. Розвиток науки та освіти в умовах глобалізації. 2024. С. 26. URL: https://researcheurope.org/wp-content/uploads/2024/08/re-02.08.24.pdf#page=26

15. Бесеганич І. В., Колесник А. В. Аналіз дієвості інноваційних освітніх платформ у процесі фахової підготовки біологів у закладах вищої освіти України. 2023. URL: https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/62336/1/АНАЛІЗ%20ДІЄВОСТІ.pdf

16. Свіріпа С. М. Особливості розробки та взаємодії з базою даних мовою Java: дис. канд. наук. Вінниця, 2020. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/29527/9145.pdf?sequence=3

17. Попов С. О. Технології розробки мобільних інформаційних систем EI1224BTRMS: 2022/2023, 7 семестр. 2022. URL: https://dspace.duet.edu.ua/jspui/bitstream/123456789/547/1/Тех_розр_MIC_122_бак.pdf Сук Д. О.

18. Адаптивна платформа для персоналізованого онлайн-навчання. 2023. URL: https://ela.kpi.ua/server/api/core/bitstreams/d07b6eb9-f0fc-48ec-bf5c-a94e87122985/content

19. Бурлака О. С. Розробка навчальної онлайн-платформи для організації навчального процесу з використанням мови програмування Java. 2024. URL: https://ir.nmu.org.ua/bitstream/handle/123456789/167412/122202_Бурлака.pdf?sequence=1

20. Тіщенко А. А. Платформа дистанційного навчання за дисциплінами кафедри. 2024.

21. Стоволос С. Ю. Інформаційне та програмне забезпечення платформи для онлайн навчання. 2022. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/89004/1/Stovolos_bac_rob.pdf

22. Антонюк Д. Проектування і розробка автоматизованої системи підбору команди співробітників. 2023. URL: http://dp.knute.edu.ua/jspui/bitstream/123456789/8992/1/ВКР_

23. Антонюк.pdf Сирін Є. М. Інформаційно-аналітична технологія моніторингу ціноутворення онлайн-платформи Nintendo: магіст. робота. Суми, 2021. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/86884/1/Syrin_mag_rob.pdf

24. Карімов С. В. Розробка web-сайту на замовлення Ярмолинецького ЗЗСО I-III ст. № 1 з використанням технологій Java script html php: бакалавр. робота. 2023. URL: https://elartu.tntu.edu.ua/bitstream/lib/42408/1/dyplom_Karimov_2023.pdf

25. Чех Т. П. Розробка інформаційної системи для виявлення вразливостей веб-сайтів: бакалавр. робота. Тернопіль, 2022. URL: https://elartu.tntu.edu.ua/bitstream/lib/38357/1/Dyplom_Chekh_T_P_2022.pdf

26. Лукашевський Ю. В. Вдосконалення методики вибору інструментарію форм участі мешканців у місцевому самоврядуванні // Актуальні питання у сучасній науці. 2024. Вип. 2 (20). С. 45–58.

27. Бантюк І. Системний аналіз та практичне застосування розподілу та сегментацій ринку комп'ютерної техніки для застосунку інтернет-магазину за допомогою мови програмування Java : дис. … канд. техн. наук. Київ, 2024.

28. Євдокимов Б. С. Дослідження знання-орієнтованих методів побудови рекомендацій в IT-проєктах індивідуального страхування : дис. … канд. техн. наук. Київ, 2024.

29. Кармазин О. А. Розробка інформаційної системи вивчення іноземних мов онлайн : дис. … канд. техн. наук. Київ, 2024.

30. Булгаков А. О. Дослідження методів обробки, зберігання та передачі даних у мережевому середовищі для виконання колективних музикальних композицій : дис. … канд. техн. наук. Київ, 2023.

31. Толмачов В., Михайловський Д. Перспективи викладання мов програмування в освітньому процесі: підготовка майбутніх учителів інформатики // Вісник Глухівського національного педагогічного університету імені Олександра Довженка. 2024. Вип. 1.54.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«АДАПТИВНІ АЛГОРИТМИ ПЕРЕВІРКИ ЗНАНЬ У ВИВЧЕННІ МОВИ ПРОГРАМУВАННЯ JAVA»

Виконав: студент групи ПДМ-63 Шевчук Юрій Олександович

Керівник: д.т.н., доц., завідувач кафедри ІПЗ Замрій Ірина Вікторівна

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення ефективності перевірки знань шляхом впровадження адаптивних алгоритмів, що враховують індивідуальні особливості студентів та їхній рівень підготовки, для вивчення мови програмування Java.

Об'єкт дослідження: процес перевірки знань студентів у навчанні мови програмування Java.

Предмет дослідження: адаптивні алгоритми перевірки знань, які дозволяють автоматично налаштовувати рівень складності завдань відповідно до знань і навичок студентів, а також їхньої прогресії у вивченні Java.

НЕДОЛІКИ ІСНУЮЧИХ МЕТОДІВ ПЕРЕВІРКИ ЗНАНЬ

Традиційні методи оцінювання	**Автоматизовані системи тестування**	**Існуючі освітні платформи** Coursera, Codecademy
Фокус лише на теоретичних знаннях	Обмежений аналіз якості коду	Відсутність персоналізації навчального процесу
Відсутність адаптації під індивідуальні особливості	Відсутність оцінки архітектурних рішень	Обмежені можливості адаптивного тестування
Обмежена перевірка практичних навичок	Недостатня перевірка soft skills	Недостатня інтеграція з професійними інструментами
Неврахування сучасних вимог індустрії		

МАТЕМАТИЧНА МОДЕЛЬ

Індекс складності (P) розраховується як відношення кількості правильних відповідей (R) до загальної кількості відповідей (N). Його значення може варіюватися від 0 до 1, а рекомендований діапазон - від 0.3 до 0.8. Цей показник відображає, наскільки складним є завдання для тестованих.

$$P = \frac{R}{N}$$

Розподільна здатність (D) визначається за формулою (Rh - Rl) / (0.5N), де Rh - кількість правильних відповідей у групі з високими балами, а Rl - у групі з низькими балами. Вона може набувати значень від -1 до 1, а бажане значення - більше 0.3. Цей показник характеризує здатність завдання розрізняти студентів з різним рівнем підготовки.

$$D = \frac{Rh - Rl}{0.5N}$$

МАТЕМАТИЧНА МОДЕЛЬ

Надійність (α) розраховується за формулою, що враховує кількість завдань у тесті (k), суму дисперсій балів за кожне завдання ($\Sigma\sigma i^2$) та дисперсію індивідуальних балів за тест (σx^2). Надійність може варіюватися від 0 до 1, а рекомендоване значення - вище 0.7. Вона відображає внутрішню узгодженість та стабільність результатів тесту.

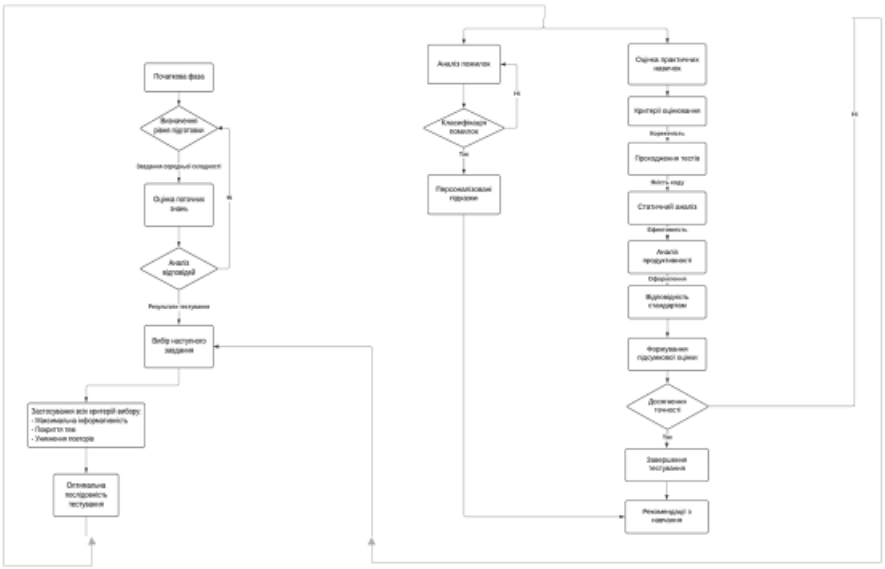
$$\alpha = \left(\frac{k}{k-1}\right)\left(1 - \frac{\sum \sigma i^2}{\sigma x^2}\right)$$

Валідність (V) визначається як квадратний корінь з надійності ($\sqrt{\alpha}$) і може набувати значень від 0 до 1. Бажане значення - більше 0.8. Валідність показує, наскільки тест дійсно вимірює те, що має на меті.

$$V = \sqrt{\alpha}$$

5

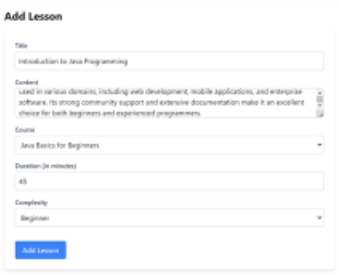
АЛГОРИТМ АДАПТИВНОГО ТЕСТУВАННЯ



6

ЕКРАННІ ФОРМИ

Головна сторінка



Додавання уроку

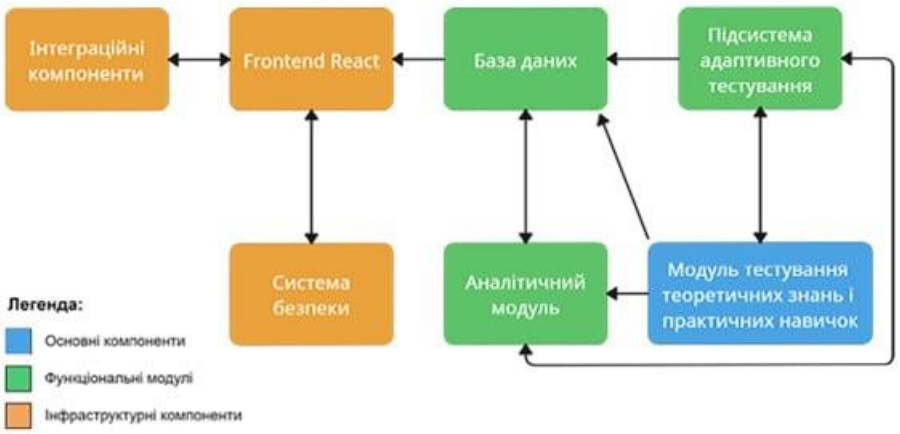
7

ТАБЛИЦЯ КОМПОНЕНТІВ РОЗРОБЛЕНОЇ СИСТЕМИ

Компонент системи	Опис функціональності	Використані технології	Взаємодія з іншими компонентами
Модуль тестування теоретичних знань і практичних навичок	Перевірка теоретичних знань через тести. Оцінка практичних завдань з програмування. Автоматична перевірка коду. Аналіз якості програмних рішень.	Java 17, Spring Boot 3.0, JUnit 5, Maven, CheckStyle, PMD	Взаємодія з базою даних для зберігання результатів. Інтеграція з аналітичним модулем. Обмін даними через Kafka.
База даних	Зберігання профілів користувачів. Банк завдань та тестів. Історія результатів. Аналітичні дані. Метадані системи.	PostgreSQL 14, Hibernate 6.0, Flyway, pgAdmin 4, Redis	Доступ від усіх модулів системи. Реплікація даних. Резервне копіювання.
Аналітичний модуль	Обробка статистики навчання. Формування рекомендацій. Аналіз прогресу студентів. Генерація звітів. Прогнозування результатів.	Elasticsearch 8.0, Kibana, Apache Spark, TensorFlow, Python	Отримання даних через Kafka. Взаємодія з БД. Інтеграція з підсистемою тестування.
Підсистема адаптивного тестування	Динамічне налаштування складності. Аналіз відповідей. Побудова індивідуальних траєкторій. Оцінка рівня знань. Калібрування завдань.	Spring Cloud, ML алгоритми, Item Response Theory, Bayesian Networks, Neural Networks	Інтеграція з модулем тестування. Взаємодія з аналітичним модулем. Обмін даними через API.
Frontend частина	Користувацький інтерфейс. Адаптивний дизайн. Інтерактивні компоненти. Візуалізація даних. Real-time оновлення.	React 18, TypeScript, Material UI, Redux Toolkit, Recharts	REST API взаємодія. WebSocket комунікація. Server-Sent Events.
Інтеграційні компоненти	API Gateway. Мікросервісна архітектура. Балансування навантаження. Моніторинг. Логування.	Spring Cloud Gateway, Docker, Kubernetes, Prometheus, Grafana	Взаємодія з усіма компонентами. Зовнішні інтеграції. Сервіси безпеки.
Система безпеки	Автентифікація. Авторизація. Захист даних. Аудит дій. Виявлення вторгнень.	Spring Security, OAuth 2.0, JWT, Keycloak, SSL/TLS	Інтеграція з усіма модулями. Моніторинг активності. Логування подій.

8

АРХІТЕКТУРА СИСТЕМИ ОЦІНЮВАННЯ



9

ТЕХНОЛОГІЇ РОЗРОБКИ

Java

spring Framework

HIBERNATE

React

PostgreSQL

docker

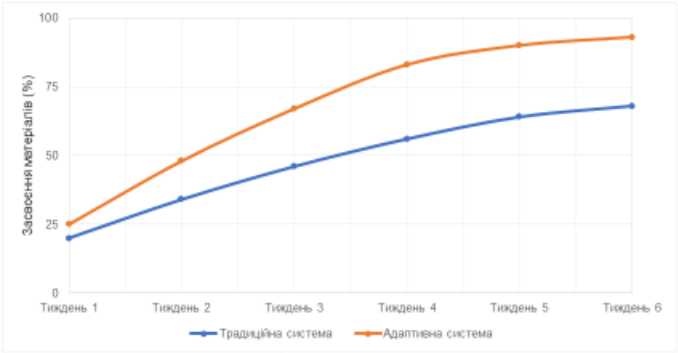
APACHE kafka

10

ПОРІВНЯЛЬНИЙ АНАЛІЗ

Характеристика	Традиційна система	Адаптивна система
Підхід до завдань	Єдині для всіх	Індивідуалізовані
Оцінка прогресу	Фіксується за результатами тесту	Аналізується постійно
Мотивація	Орієнтована на оцінку	Орієнтована на розвиток
Гнучкість	Відсутня	Висока
Технологічна підтримка	Мінімальна	Максимальна (онлайн-платформи)

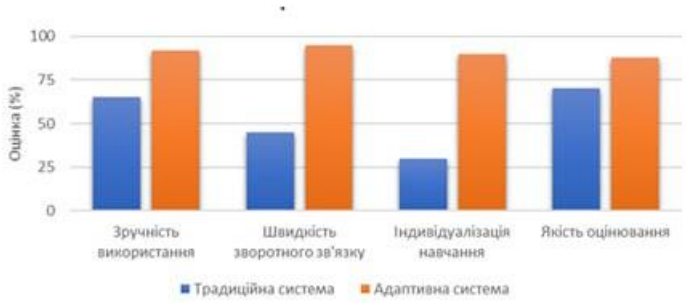
ПОРІВНЯЛЬНИЙ АНАЛІЗ



Швидкість засвоєння матеріалу

У дослідженні порівнювали традиційну та адаптивну системи навчання протягом шести тижнів. Студенти, які використовували адаптивну систему, показали кращі результати: їхня крива була вище на всіх етапах, особливо на 6-му тижні, де рівень засвоєння матеріалу досягав 95% порівняно з 68% у традиційній системі. Також нахил кривої адаптивної системи був крутішим, що вказує на швидше засвоєння матеріалу на початкових етапах.

ПОРІВНЯННЯ КЛЮЧОВИХ ПОКАЗНИКІВ ЕФЕКТИВНОСТІ



Порівняльна діаграма ключових показників демонструє переваги адаптивної системи за основними критеріями:

1. Зручність використання (Опитування): 92% проти 65%
2. Швидкість зворотного зв'язку (Опитування): 95% проти 45%
3. Індивідуалізація навчання (Опитування): 90% проти 30%
4. Якість оцінювання (Метод повторного тестування): 88% проти 70%

13

ВИСНОВКИ

1. Проведено аналіз проблематики існуючих методів перевірки знань та виявлено необхідність вдосконалення підходів до оцінювання.
2. На основі аналізу існуючих методів спроектовано архітектуру та основні компоненти адаптивної системи оцінювання знань з Java-програмування.
3. Розроблено алгоритми адаптивного тестування та оцінювання практичних навичок програмування, які враховують індивідуальні особливості студентів та забезпечують персоналізований підхід до навчання.
4. Розроблено адаптивну систему оцінювання знань з програмування на Java, яка забезпечує ефективну перевірку теоретичних знань та практичних навичок студентів.
5. Експериментальне впровадження системи показало підвищення ключових показників за основними критеріями:
   1. Зручність використання : 92% проти 65%
   2. Швидкість зворотного зв'язку : 95% проти 45%
   3. Індивідуалізація навчання : 90% проти 30%
   4. Якість оцінювання : 88% проти 70%

14

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. Шевчук Ю.О., Замрій І.В. Онлайн-платформа для навчання мові програмування JAVA: розробка, впровадження та ефективність // Зв'язок, 2024, №6. (Прийнято до друку, буде опубліковано у кінці грудня).

Тези доповідей:

1. Шевчук Ю.О., Замрій І.В. Онлайн-платформи для навчання мові програмування JAVA. IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 року, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024.С. 277-279.
2. Шевчук Ю.О., Замрій І.В. Адаптивні алгоритми перевірки знань у вивченні мови програмування java. IV Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2024 року, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024.С. 83

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ МОДУЛІВ

1. **JavalearnApplication.java**

```
package com.alex911.javalearn;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApp
lication;

@SpringBootApplication
public class JavalearnApplication {

	public static void main(String[] args) {

	SpringApplication.run(JavalearnApplication.cla
ss, args);
	}
}
```

2. **WebConfig.java**

```
package com.alex911.javalearn.config;

import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.Cors
Registry;
import
org.springframework.web.servlet.config.annotation.Web
MvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer
{

   @Bean
   public WebMvcConfigurer corsConfigurer() {
      return new WebMvcConfigurer() {
         @Override
         public void addCorsMappings(CorsRegistry
registry) {
            registry.addMapping("/**")
                  .allowedOrigins("http://localhost:3000")
                  .allowedMethods("GET", "POST",
"PUT", "DELETE", "OPTIONS")
                  .allowedHeaders("*")
                  .allowCredentials(true);
         }
      };
   }
}
```

3. **CourseController.java**

```
package com.alex911.javalearn.controller;

import java.util.List;
import java.util.Optional;

import
org.springframework.beans.factory.annotation.Autowire
d;
import
org.springframework.web.bind.annotation.CrossOrigin;
import
org.springframework.web.bind.annotation.DeleteMappin
g;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PathVariable;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.PutMapping;
import
org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RequestMappi
ng;
import
org.springframework.web.bind.annotation.RestController
;

import com.alex911.javalearn.dto.CourseDTO;
import com.alex911.javalearn.model.Course;
import com.alex911.javalearn.service.CourseService;

@RestController
@RequestMapping("/courses")
@CrossOrigin(origins = "http://localhost:3000")
public class CourseController {

   @Autowired
   private CourseService courseService;

   @GetMapping
   public List<CourseDTO> getAllCourses() {
      List<CourseDTO> courses =
courseService.getAllCourses();
      System.out.println("Fetching all courses" +
courses);
      return courses;
   }

   @GetMapping("/{id}")
   public Optional<CourseDTO>
getCourseById(@PathVariable Long id) {
      System.out.println("Fetching course with id: " + id);
      return courseService.getCourseById(id);
   }

   @PostMapping
   public CourseDTO createCourse(@RequestBody
Course course) {
      System.out.println("Creating course: " + course);
      return courseService.saveCourse(course);
   }

   @PutMapping("/{id}")
```

```
    public CourseDTO updateCourse(@PathVariable
Long id, @RequestBody Course course) {
        System.out.println("Updating course with id: " + id
+ " and course: " + course);
        course.setId(id);
        return courseService.saveCourse(course);
    }

    @DeleteMapping("/{id}")
    public void deleteCourse(@PathVariable Long id) {
        System.out.println("Deleting course with id: " + id);
        courseService.deleteCourse(id);
    }
}
```

4. LessonController.java

```
package com.alex911.javalearn.controller;

import java.util.List;
import java.util.Optional;

import
org.springframework.beans.factory.annotation.Autowire
d;
import
org.springframework.web.bind.annotation.CrossOrigin;
import
org.springframework.web.bind.annotation.DeleteMappin
g;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PathVariable;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.PutMapping;
import
org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RequestMappi
ng;
import
org.springframework.web.bind.annotation.RestController
;

import com.alex911.javalearn.dto.LessonDTO;
import com.alex911.javalearn.service.LessonService;


@RestController
@RequestMapping("/lessons")
@CrossOrigin(origins = "http://localhost:3000")
public class LessonController {

    @Autowired
    private LessonService lessonService;

    @GetMapping
    public List<LessonDTO> getAllLessons() {
        System.out.println("Get all lessons");
        return lessonService.getAllLessons();
    }

    @GetMapping("/{id}")
    public Optional<LessonDTO>
getLessonById(@PathVariable Long id) {
        System.out.println("Get lesson with id: " + id);
        return lessonService.getLessonById(id);
    }

    @PostMapping
    public LessonDTO createLesson(@RequestBody
LessonDTO lesson) {
        System.out.println("Create lesson: " + lesson);
        return lessonService.saveLesson(lesson);
    }


    @PutMapping("/{id}")
    public LessonDTO updateLesson(@PathVariable
Long id, @RequestBody LessonDTO lesson) {
        System.out.println("Update lesson with id: " + id +
" and lesson: " + lesson);
        lesson.setId(id);
        return lessonService.saveLesson(lesson);
    }

    @DeleteMapping("/{id}")
    public void deleteLesson(@PathVariable Long id) {
        System.out.println("Delete lesson with id: " + id);
        lessonService.deleteLesson(id);
    }

}
```

5. CourseDTO.java

```
package com.alex911.javalearn.dto;

import java.util.List;

public class CourseDTO {
    private Long id;
    private String title;
    private String description;
    private String duration;
    private List<LessonDTO> lessons;
    private List<TestDTO> tests;


    // Constructors
    public CourseDTO(Long id, String title, String
description, String duration, List<LessonDTO> lessons,
List<TestDTO> tests) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.duration = duration;
        this.lessons = lessons;
        this.tests = tests;
    }

    public CourseDTO() { }
    // Getters and Setters

    public Long getId() {
        return id;
    }
```

```
    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getDuration() {
        return duration;
    }

    public void setDuration(String duration) {
        this.duration = duration;
    }

    public List<LessonDTO> getLessons() {
        return lessons;
    }

    public void setLessons(List<LessonDTO> lessons) {
        this.lessons = lessons;
    }

    public List<TestDTO> getTests() {
        return tests;
    }

    public void setTests(List<TestDTO> tests) {
        this.tests = tests;
    }
}
```

6. LessonDTO.java

```
package com.alex911.javalearn.dto;

import com.alex911.javalearn.model.Lesson;

public class LessonDTO {
    private Long id;
    private String title;
    private String content;
    private String duration;
    private String complexity;
    private CourseDTO course;
    private Long courseId;

    public LessonDTO(Lesson lesson) {
        this.id = lesson.getId();
        this.title = lesson.getTitle();
        this.content = lesson.getContent();
        this.duration = lesson.getDuration();
        this.complexity = lesson.getComplexity();
        this.courseId = lesson.getCourse().getId();
        this.course = new CourseDTO();
        this.course.setId(lesson.getCourse().getId());
        this.course.setTitle(lesson.getCourse().getTitle());
this.course.setDescription(lesson.getCourse().getDescript
ion());
this.course.setDuration(lesson.getCourse().getDuration()
);
        this.course.setLessons(null);
    }

    public LessonDTO() {}

    // Getters and Setters

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public String getDuration() {
        return duration;
    }

    public void setDuration(String duration) {
        this.duration = duration;
    }

    public String getComplexity() {
        return complexity;
    }

    public void setComplexity(String complexity) {
        this.complexity = complexity;
    }

    public CourseDTO getCourse() {
        return course;
    }
```

```
    public void setCourse(CourseDTO course) {
        this.course = course;
    }

    public Long getCourseId() {
        return this.courseId;
    }

    public void setCourseId(Long courseId) {
        this.courseId = courseId;
    }

}
```

7. Course.java

```
package com.alex911.javalearn.model;

import java.util.List;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.OneToMany;

@Entity
public class Course {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    private String title;

    private String description;

    private String duration;

    @OneToMany(mappedBy = "course", cascade =
CascadeType.ALL)
    private List<Test> tests;

    @OneToMany(mappedBy = "course", cascade =
CascadeType.ALL, orphanRemoval = true)
    private List<Lesson> lessons;

    // Getters and Setters

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public List<Lesson> getLessons() {
        return lessons;
    }

    public void setLessons(List<Lesson> lessons) {
        this.lessons = lessons;
    }

    public String getDuration() {
        return duration;
    }

    public void setDuration(String duration) {
        this.duration = duration;
    }

    public List<Test> getTests() {
        return tests;
    }

    public void setTests(List<Test> tests) {
        this.tests = tests;
    }

}
```

8. Lesson.java

```
package com.alex911.javalearn.model;

import java.util.List;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.OneToMany;

@Entity
public class Lesson {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    private String title;

    @Column(columnDefinition = "TEXT")
    private String content;
    private String duration;
    private String complexity;
```

```
@ManyToOne
@JoinColumn(name = "course_id")
private Course course;

@OneToMany
@JoinColumn(name = "question_id")
private List<Question> questions;

// Getters and Setters

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getContent() {
    return content;
}

public void setContent(String content) {
    this.content = content;
}

public String getDuration() {
    return duration;
}

public void setDuration(String duration) {
    this.duration = duration;
}

public String getComplexity() {
    return complexity;
}

public void setComplexity(String complexity) {
    this.complexity = complexity;
}

public void setCourseId(Long courseId) {
    this.course.setId(courseId);
}

public Long getCourseId() {
    return course.getId();
}

public void setCourseTitle(String courseTitle) {
    this.course.setTitle(courseTitle);
}

    public Course getCourse() {
        return course;
    }

    public void setCourse(Course course) {
        this.course = course;
    }

    public void setQuestions(List<Question> questions) {
        this.questions = questions;
    }

    public List<Question> getQuestions() {
        return questions;
    }

}
```

9. CourseRepository.java

```
package com.alex911.javalearn.repository;

import
org.springframework.data.jpa.repository.JpaRepository;

import com.alex911.javalearn.model.Course;

public interface CourseRepository extends
JpaRepository<Course, Long> {
}
```

10. LessonRepository.java

```
package com.alex911.javalearn.repository;

import
org.springframework.data.jpa.repository.JpaRepository;

import com.alex911.javalearn.model.Lesson;

public interface LessonRepository extends
JpaRepository<Lesson, Long> {
}
```

11. CourseService.java

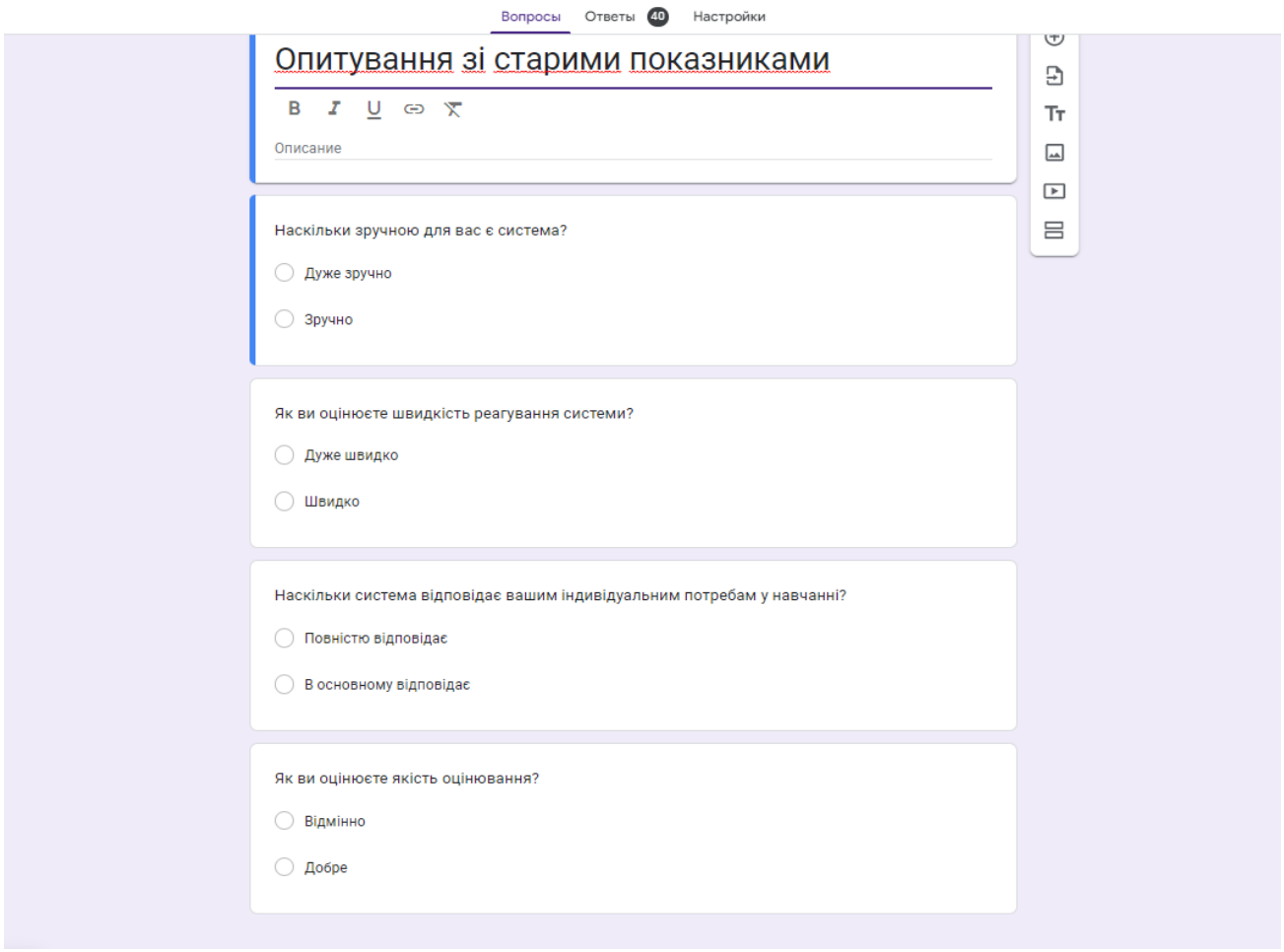
```
package com.alex911.javalearn.service;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

import
org.springframework.beans.factory.annotation.Autowire
d;
import org.springframework.stereotype.Service;

import com.alex911.javalearn.dto.CourseDTO;
import com.alex911.javalearn.dto.LessonDTO;
import com.alex911.javalearn.model.Course;
import com.alex911.javalearn.model.Lesson;
import
com.alex911.javalearn.repository.CourseRepository;

@Service
public class CourseService {

    @Autowired
    private CourseRepository courseRepository;
```

```
    public List<CourseDTO> getAllCourses() {
        List<Course> courses = courseRepository.findAll();
        return courses.stream()
                .map(this::convertToDTO)
                .collect(Collectors.toList());
    }

    public Optional<CourseDTO> getCourseById(Long
id) {
        return courseRepository.findById(id)
                        .map(this::convertToDTO);
    }

    public CourseDTO saveCourse(Course course) {
        Course savedCourse =
courseRepository.save(course);
        return convertToDTO(savedCourse);
    }

    public void deleteCourse(Long id) {
        courseRepository.deleteById(id);
    }

    private CourseDTO convertToDTO(Course course) {
        CourseDTO dto = new CourseDTO();
        dto.setId(course.getId());
        dto.setTitle(course.getTitle());
        dto.setDescription(course.getDescription());
        dto.setDuration(course.getDuration());
        System.out.println("Course lessons: " +
course.getLessons());
        if (course.getLessons() == null) return dto;
        dto.setLessons(course.getLessons().stream()
                        .map(this::convertLessonToDTO)
                        .collect(Collectors.toList()));
        return dto;
    }

    private LessonDTO convertLessonToDTO(Lesson
lesson) {
        LessonDTO dto = new LessonDTO(lesson);

        return dto;
    }

}
```

12. LessonService java

```
package com.alex911.javalearn.service;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

import
org.springframework.beans.factory.annotation.Autowire
d;
import org.springframework.stereotype.Service;

import com.alex911.javalearn.dto.LessonDTO;
import com.alex911.javalearn.model.Lesson;
import
com.alex911.javalearn.repository.CourseRepository;
import
com.alex911.javalearn.repository.LessonRepository;

@Service
public class LessonService {

    @Autowired
    private LessonRepository lessonRepository;

    @Autowired
    private CourseRepository courseRepository;

    public List<LessonDTO> getAllLessons() {
        return lessonRepository.findAll().stream()
            .map(this::convertToDTO)
            .collect(Collectors.toList());
    }

    public Optional<LessonDTO> getLessonById(Long
id) {
        return
lessonRepository.findById(id).map(this::convertToDTO)
;
    }

    public LessonDTO saveLesson(LessonDTO lesson) {
        Lesson savedLesson =
lessonRepository.save(this.convertToEntity(lesson));
        return convertToDTO(savedLesson);
    }

    public void deleteLesson(Long id) {
        lessonRepository.deleteById(id);
    }

    private LessonDTO convertToDTO(Lesson lesson) {
        LessonDTO dto = new LessonDTO(lesson);
        return dto;
    }

    private Lesson convertToEntity(LessonDTO dto) {
        Lesson lesson = new Lesson();
        lesson.setId(dto.getId());
        lesson.setTitle(dto.getTitle());
        lesson.setContent(dto.getContent());
        // You should fetch the Course entity from the
repository or service
        // Example:
lesson.setCourse(courseRepository.findById(dto.getCour
seId()).orElse(null));
        //if (dto.getCourse() == null) return lesson;

lesson.setCourse(courseRepository.findById(dto.getCour
seId()).orElse(null));
        return lesson;
    }
}
```

ДОДАТОК В. ОПИТУВАННЯ ЗІ СТАРИМИ ПОКАЗНИКАМИ



ДОДАТОК Г. ОПИТУВАННЯ З ФІКСОВАНИМИ РЕЗУЛЬТАТАМИ