Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА

«_____»_____2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ТЕМИ «COLLECTIONS. TREESET» ДИСЦИПЛІНИ «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступень бакалавр

Виконавець роботи Сліпаченко Анна Олександрівна
_____«___»_____2026р.

Науковий керівник к.ф.-м.н., Олексійчук Юрій Федорович
_____ «___»_____2026р.

Рецензент

ПОЛТАВА 2026 р.

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Олена ОЛЬХОВСЬКА
«__» _____20__р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка навчального програмного забезпечення з теми «Collections. TreeSet» дисципліни «Об'єктно-орієнтоване програмування»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Сліпаченко Анна Олександрівна
затверджена наказом ректора № від «__» _____ 20 р.
Термін подання студентом роботи «__» _____20 р.
Вихідні дані до кваліфікаційної роботи: публікації з теми навчальних тренажерів у дистанційних курсах з комп'ютерних наук, матеріали дистанційного курсу з дисципліни «Об'єктно-орієнтоване програмування» з теми «Java Collections Framework», а також технічна документація Java API щодо класу TreeSet та червоно-чорного дерева, на основі якої розробляється навчальний тренажер.
3. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП

1. ПОСТАНОВКА ЗАДАЧІ

1.1. Змістовна постановка задачі розробки навчального тренажеру з теми «Collections. TreeSet»

1.2. Вимоги до тренажера

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд існуючих навчальних тренажерів та програмного забезпечення з тематики структур даних і колекцій Java

2.2. Позитивні аспекти оглянутих програмних розробок

2.3. Вади оглянутих програмних розробок

2.4. Необхідність та актуальність розробки тренажеру з теми «Collections. TreeSet»

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Проєктування архітектури навчального тренажеру

3.2. Графічне представлення архітектури (UML-діаграми)

3.3. Обґрунтування вибору програмних засобів для реалізації тренажеру

4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис процесу програмної реалізації елементів тренажеру.

4.2. Опис розробленого тренажеру, вхідних даних та програмних обмежень

4.3. Перевірка валідності та дослідження можливостей тренажеру

4.4. Інструкція користувача тренажеру

ВИСНОВКИ

СПИСОК ЛІТЕРАТУРИ

ДОДАТОК А

Перелік графічного матеріалу: 3-4 аркуші графічного матеріалу, інші необхідні ілюстрації.

Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постанова задачі			
Інформаційний огляд			
Теоретична частина			
Практична реалізація			

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 20 р.

Здобувач вищої освіти _____

Науковий керівник _____

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____

(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2026 р.

Секретар ЕК_____ _____

(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«__» _____ 202_ р.

Погоджено
Науковий керівник_____
_____
«__» _____ 202_ р.

План

кваліфікаційної роботи на тему
«Розробка навчального програмного забезпечення з теми «Collections. TreeSet» дисципліни «Об'єктно-орієнтоване програмування»
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
ступеня бакалавр
Прізвище, ім'я, по батькові Сліпаченко Анна Олександрівна

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
1.1. Змістовна постановка задачі розробки навчального тренажеру з теми «Collections. TreeSet»
1.2. Вимоги до тренажера
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Огляд існуючих навчальних тренажерів та програмного забезпечення з тематики структур даних і колекцій Java
2.2. Позитивні аспекти оглянутих програмних розробок
2.3. Вади оглянутих програмних розробок
2.4. Необхідність та актуальність розробки тренажеру з теми «Collections. TreeSet»
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Проєктування архітектури навчального тренажеру
3.2. Графічне представлення архітектури (UML-діаграми)
3.3. Обґрунтування вибору програмних засобів для реалізації тренажеру
4. ПРАКТИЧНА ЧАСТИНА
4.1. Опис процесу програмної реалізації елементів тренажеру.
4.2. Опис розробленого тренажеру, вхідних даних та програмних обмежень
4.3. Перевірка валідності та дослідження можливостей тренажеру
4.4. Інструкція користувача тренажеру
ВИСНОВКИ

Здобувач вищої освіти _____ _____
«____» _____ 202_ р.

РЕФЕРАТ

Записка: 74 с., 27 рис., 9 таблиць, 1 додаток, 25 джерел.

COLLECTIONS, TREESET, НАВЧАЛЬНИЙ ТРЕНАЖЕР, JAVA, RED-BLACK TREE, ДИСТАНЦІЙНЕ НАВЧАННЯ, WEB-ЗАСТОСУНОК.

Об'єкт розробки – дистанційне навчання студентів спеціальності 122 «Комп'ютерні науки» при вивченні теми «Java Collections Framework».

Мета роботи – проєктування та програмна реалізація навчального тренажеру з теми «Collections. TreeSet» як засобу підтримки дистанційного навчання студентів спеціальності «Комп'ютерні науки».

Методи дослідження – методи об'єктно-орієнтованого та компонентного програмування, архітектурний підхід клієнт-сервер, метод порівняльного аналізу програмного забезпечення, функціональне тестування.

Здійснено огляд шести існуючих програмних рішень у сфері навчальних тренажерів та платформ для вивчення структур даних. Виявлено їх позитивні та негативні сторони. Побудовано модель навчального тренажеру та сформульовано функціональні і нефункціональні вимоги до системи. Спроєктовано трирівневу клієнт-серверну архітектуру з використанням Java 17, Spring Boot 3.2, SQLite та HTML/CSS/JavaScript. Реалізовано сім функціональних модулів: теоретичний модуль, модуль інтерактивної візуалізації з покроковою анімацією балансування червоно-чорного дерева, модуль покрокових задач з двадцятьма задачами трьох рівнів складності, модуль тестування з банком тридцяти питань, модуль статистики прогресу студента, адміністративну панель для викладача та модуль порівняльної візуалізації TreeSet, HashSet і LinkedHashSet. Проведено функціональне тестування п'ятдесяти тестових сценаріїв.

Результати роботи можуть бути проваджені у навчальний процес при викладанні дисципліни «Об'єктно-орієнтоване програмування» як інструмент для самостійної підготовки студентів та контролю знань викладачем.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ……..…….....7

ВСТУП..... 8

1 ПОСТАНОВКА ЗАДАЧІ……10

1.1 Змістовна постановка задачі розробки навчального тренажеру з теми «Collections. TreeSet»..... 10

1.2 Модель навчального тренажеру та її характеристики..... 11

2ІНФОРМАЦІЙНИЙ ОГЛЯД……...17

2.1 Огляд існуючих навчальних тренажерів та програмного забезпечення з тематики структур даних і колекцій Java..... 17

2.2 Позитивні аспекти оглянутих програмних розробок..... 22

2.3 Вади оглянутих програмних розробок..... 23

2.4 Необхідність та актуальність розробки тренажеру з теми «Collections. TreeSet»..... 25

3ТЕОРЕТИЧНА ЧАСТИНА……..28

3.1 Проєктування архітектури навчального тренажеру..... 28

3.2 Графічне представлення архітектури (UML-діаграми)..... 31

3.3 Обґрунтування вибору програмних засобів для реалізації тренажеру..... 36

4ПРАКТИЧНА ЧАСТИНА……40

4.1 Опис процесу програмної реалізації елементів тренажеру..... 40

4.2 Опис розробленого тренажеру, вхідних даних та програмних обмежень..... 42

4.3 Перевірка валідності та дослідження можливостей тренажеру..... 44

4.4 Інструкція користувача тренажеру..... 48

ВИСНОВКИ..... 62

СПИСОК ЛІТЕРАТУРИ..... 64

ДОДАТОК А..... 67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

IIFE – Immediately Invoked Function Expression

JPA – Java Persistence API

JSON – JavaScript Object Notation

JVM – Java Virtual Machine

LTS – Long-Term Support

MVC – Model-View-Controller

REST – Representational State Transfer

SVG – Scalable Vector Graphics

UML – Unified Modeling Language

ВСТУП

Сучасна вища технічна освіта переживає суттєву трансформацію під впливом процесу цифровізації. Традиційні форми подачі навчального матеріалу – лекції, підручники, статичні конспекти – дедалі більше поступаються місцем інтерактивним інструментам, які залучають студента до активної взаємодії з навчальним контентом. Особливо це актуально для дисциплін, що вимагають не лише теоретичного розуміння, а й практичного осмислення алгоритмів та структур даних.

Робота присвячена розробці повноцінного навчального тренажеру з теми «Collections. TreeSet» для студентів спеціальності 122 «Комп'ютерні науки». Ця тема є частиною дисципліни «Об'єктно-орієнтоване програмування» і охоплює один із ключових компонентів Java Collections Framework – клас TreeSet, побудований на основі самобалансувального червоно-чорного дерева. Розуміння внутрішньої логіки цієї структури даних є важливою компетенцією для будь-якого Java-розробника, проте засвоєння теми викликає у студентів значні труднощі саме через абстрактний характер алгоритмів балансування.

Актуальність роботи зумовлена відсутністю на ринку навчального програмного забезпечення спеціалізованого україномовного інтерактивного тренажеру, який би поєднував теоретичні пояснення, інтерактивну візуалізацію внутрішньої структури TreeSet, систему задач з підказками та тестування в єдиному середовищі з прив'язкою до Java API.

Метою роботи є проєктування та програмна реалізація повноцінного навчального тренажеру з теми «Collections. TreeSet» як засобу підтримки дистанційного навчання студентів спеціальності «Комп'ютерні науки».

Об'єктом розробки є процес дистанційного навчання студентів спеціальності 122 «Комп'ютерні науки» при вивченні теми «Java Collections Framework».

Предметом розробки є навчальний тренажер з теми «Collections. TreeSet» як складова курсу з дисципліни «Об'єктно-орієнтоване програмування».

Для досягнення поставленої мети необхідно вирішити *такі завдання*:

– проаналізувати існуючі навчальні тренажери та програмне забезпечення з тематики структур даних і колекцій Java;

– побудувати модель навчального тренажеру та сформулювати вимоги до системи;

– розробити покрокову анімацію балансування червоно-чорного дерева;

– реалізувати систему збереження та відображення прогресу студента між сеансами;

– розробити адміністративну панель для викладача з переглядом результатів групи;

– розширити банк задач та тестових питань, додати рівні складності;

– реалізувати модуль порівняльної візуалізації TreeSet, HashSet та LinkedHashSet;

– провести тестування тренажеру.

Методи дослідження: методи об'єктно-орієнтованого та компонентного програмування, архітектурний підхід клієнт-сервер, метод порівняльного аналізу програмного забезпечення.

Практичне значення роботи полягає в тому, що розроблений тренажер може бути безпосередньо впроваджений у навчальний процес при викладанні дисципліни «Об'єктно-орієнтоване програмування» та використаний як методичний інструмент для самостійної підготовки студентів.

Робота складається з чотирьох розділів. У першому розділі сформульовано постановку задачі та побудовано модель тренажеру. У другому розділі здійснено інформаційний огляд існуючих рішень. У третьому розділі описано теоретичну основу розробки. У четвертому розділі викладено практичну реалізацію тренажеру.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовна постановка задачі розробки навчального тренажеру з теми «Collections. TreeSet»

У рамках дисципліни «Об'єктно-орієнтоване програмування» студенти вивчають тему «Java Collections Framework». Серед усіх колекцій Java особливе місце займає клас TreeSet – реалізація множини на основі самобалансувального червоно-чорного дерева. Ця тема поєднує два рівні знань. Практичний рівень передбачає вміння правильно використовувати API класу, а теоретичний вимагає розуміння внутрішніх механізмів зберігання та балансування даних.

Студенти, як правило, добре запам'ятовують синтаксис методів класу TreeSet, але мають труднощі з розумінням того, що відбувається всередині структури при кожній операції. Зокрема, алгоритми балансування червоно-чорного дерева – повороти та перефарбування вузлів – є абстрактними для студента, якщо він бачить лише статичний текст або схему у підручнику. Відсутність інтерактивного інструменту, який би дозволяв спостерігати за цими процесами в реальному часі, є прогалиною у навчально-методичному забезпеченні дисципліни.

Метою кваліфікаційної роботи є розробка повноцінного навчального тренажеру з теми «Collections. TreeSet», який відповідає вимогам реального використання у навчальному процесі. Задача ставиться у такий спосіб.

Дано: тему навчальної дисципліни «Collections. TreeSet», цільову аудиторію – студентів спеціальності 122 «Комп'ютерні науки» першого-другого курсів, технологічну платформу – Java 17, Spring Boot 3.2, HTML/CSS/JavaScript, базу даних H2.

Знайти: архітектуру та програмну реалізацію навчального тренажеру, що включає такі функціональні модулі:

- теоретичний модуль з повним охопленням теми, прикладами коду та підсвіченням синтаксису;

- модуль інтерактивної візуалізації червоно-чорного дерева з покроковою анімацією балансування та підтримкою всіх навігаційних методів TreeSet;
- модуль задач з не менше ніж двадцятьма задачами трьох рівнів складності та ієрархічною системою підказок;
- модуль тестування з банком не менше тридцяти питань та випадковою вибіркою для кожного тестування;
- модуль статистики прогресу студента – збереження результатів, динаміка по часу, рекомендації щодо повторення;
- адміністративна панель для викладача – перегляд результатів групи, управління банком задач і питань;
- модуль порівняльної візуалізації – паралельне відображення TreeSet, HashSet та LinkedHashSet для одного набору даних.

Умови та обмеження: застосунок повинен коректно працювати на операційних системах Linux, macOS та Windows без додаткового налаштування. Для запуску потрібна лише Java 17+ та браузер. Інтерфейс виключно українською мовою. Дані про прогрес студентів зберігаються у вбудованій базі даних H2.

1.2 Модель навчального тренажеру та її характеристики

Навчальний тренажер розглядається як програмна система, яка моделює процес вивчення теми «Collections. TreeSet» і забезпечує зворотний зв'язок студенту на кожному кроці навчання. Модель тренажеру будується на основі концепції адаптивного навчання – тобто система реагує на дії та результати студента і підлаштовує подачу матеріалу відповідно до його рівня знань [1].

З функціональної точки зору тренажер описується кортежем T = (M, S, R, F), де M – множина навчальних модулів, S – стан студента (прогрес, результати, рівень складності), R – правила переходу між станами, F – функція зворотного зв'язку. Такий підхід дозволяє формально описати поведінку системи та обґрунтувати архітектурні рішення.

Множина навчальних модулів M включає сім компонентів. Теоретичний модуль M_1 забезпечує подачу структурованого теоретичного матеріалу з п'яти тем, кожна з яких містить текстові пояснення, таблиці та приклади коду. Модуль візуалізації M_2 реалізує інтерактивне графічне відображення стану червоно-чорного дерева після виконання кожної операції, включаючи покрокову анімацію балансування. Модуль задач M_3 надає банк покрокових задач з ієрархічною системою підказок та автоматичною перевіркою відповідей. Модуль тестування M_4 реалізує підсумкове тестування з випадковою вибіркою питань та детальним розбором результатів. Модуль статистики M_5 зберігає та відображає прогрес студента в динаміці. Адміністративний модуль M_6 надає викладачу інструменти для управління навчальним контентом та перегляду результатів. Модуль порівняння M_7 забезпечує паралельну візуалізацію різних реалізацій Set.

Стан студента S визначається вектором показників – це кількість переглянутих тем теорії, кількість розв'язаних задач кожного рівня складності, результати проходжень тесту в динаміці, середній відсоток правильних відповідей, кількість використаних підказок. На основі цих показників система формує рекомендації. Наприклад, пропонує повторити теоретичний матеріал якщо результат тесту нижче 60%, або переходити до задач вищого рівня якщо студент успішно впорався із задачами базового рівня.

Функція зворотного зв'язку F реалізується на трьох рівнях. Миттєвий зворотний зв'язок надається після кожної дії. Наприклад, результат операції над деревом, правильність відповіді на задачу, обрана відповідь у тесті. Підсумковий зворотний зв'язок надається після завершення тесту у вигляді детального розбору кожного питання. Довгостроковий зворотний зв'язок реалізується через модуль статистики, де студент бачить динаміку своїх результатів у часі.

З архітектурної точки зору тренажер реалізує трирівневу клієнт-серверну архітектуру. Рівень подання реалізований як вебінтерфейс на HTML/CSS/JavaScript, що забезпечує доступність з будь-якого браузера без встановлення додаткового програмного забезпечення. Рівень бізнес-логіки реалізований на Java 17 з використанням Spring Boot 3.2 і відповідає за обробку запитів, управління деревом,

перевірку відповідей та формування зворотного зв'язку. Рівень даних реалізований на основі вбудованої бази даних H2, що зберігає інформацію про користувачів, їх прогрес та результати.

Взаємодія між рівнями здійснюється через REST API у форматі JSON. Така архітектура забезпечує чітке розділення відповідальності між компонентами та спрощує подальше розширення системи.

Характеристики тренажеру, що описують вимоги до системи, можна поділити на функціональні та нефункціональні.

Функціональні вимоги описують що саме система повинна вміти робити. Тобто конкретні функції та можливості, які студент або викладач може використовувати під час роботи з тренажером [2]. Вони визначають поведінку системи з точки зору користувача. Перелік функціональних вимог наведено у таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги до навчального тренажеру

№	Вимога	Опис	Пріоритет
Ф1	Операція add()	Додавання цілочисельного елементу до дерева з перевіркою дублікатів та відображенням результату	Обов'язкова
Ф2	Операція remove()	Видалення елементу з дерева з повідомленням якщо елемент відсутній	Обов'язкова
Ф3	Операція contains()	Перевірка наявності елементу з відображенням результату true/false	Обов'язкова
Ф4	Операція first() / last()	Отримання мінімального та максимального елементу множини	Обов'язкова
Ф5	Операція floor() / ceiling()	Пошук найближчого елементу знизу та зверху від заданого значення	Обов'язкова
Ф6	Операція headSet() / tailSet() / subSet()	Візуальне виділення підмножини елементів на дереві	Обов'язкова

Продовження табл. 1.1

Ф7	Покрокова анімація балансування	Відображення проміжних станів дерева під час поворотів та перефарбування вузлів при операціях add() та remove()	Обов'язкова
Ф8	Теоретичний модуль	П'ять тематичних розділів з поясненнями, таблицями та прикладами Java-коду з підсвіченням синтаксису	Обов'язкова
Ф9	Банк задач	Не менше двадцяти покрокових задач, розподілених за трьома рівнями складності, з ієрархічною системою підказок	Обов'язкова
Ф10	Банк тестових питань	Не менше тридцяти питань з вибором однієї відповіді, випадкова вибірка десяти для кожного тестування	Обов'язкова
Ф11	Автоматична перевірка відповідей	Перевірка відповідей на задачі та тест з детальним поясненням правильної відповіді	Обов'язкова
Ф12	Збереження прогресу студента	Зберігання результатів задач та тестів між сеансами з відображенням динаміки у часі	Обов'язкова
Ф13	Адміністративна панель	Перегляд результатів усіх студентів групи, управління банком задач та питань	Обов'язкова
Ф14	Модуль порівняння	Паралельна візуалізація TreeSet, HashSet та LinkedHashSet для одного набору вхідних даних	Бажана
Ф15	Рекомендації студенту	Автоматичні рекомендації щодо повторення матеріалу на основі результатів тестування	Бажана

Нефункціональні вимоги описують як система повинна це робити. Тобто якісні характеристики, яким повинна відповідати реалізація. Наприклад, швидкість роботи, сумісність з операційними системами та браузерами, надійність, зручність розгортання [3]. Вони не залежать від конкретного функціоналу, але суттєво

впливають на придатність системи до реального використання у навчальному процесі. Перелік нефункціональних вимог наведено у таблиці 1.2.

Таблиця 1.2 – Нефункціональні вимоги до навчального тренажеру

№	Категорія	Вимога	Критерій прийнятності
НФ1	Продуктивність	Час відповіді сервера для операцій над деревом	Не більше 200 мс
НФ2	Продуктивність	Час завантаження головної сторінки	Не більше 2 секунд
НФ3	Продуктивність	Тривалість анімації одного кроку балансування	Від 300 до 1000 мс, регулюється користувачем
НФ4	Сумісність	Підтримувані операційні системи	Linux, macOS 12+, Windows 10/11
НФ5	Сумісність	Підтримувані браузери	Chrome 90+, Firefox 88+, Safari 14+
НФ6	Розгортання	Спосіб запуску	Однією командою java -jar без додаткового налаштування
НФ7	Розгортання	Необхідне програмне забезпечення	Java 17+ та браузер
НФ8	Локалізація	Мова інтерфейсу	Виключно українська
НФ9	Надійність	Коректна обробка некоректних вхідних даних	Повідомлення про помилку без аварійного завершення
НФ10	Надійність	Збереження даних при закритті браузера	Прогрес зберігається автоматично після кожної дії
НФ11	Масштабованість	Максимальна кількість елементів у дереві для коректної візуалізації	До 20 елементів з коректним відображенням

Продовження табл. 1.2

НФ12	Супроводжуваність	Структура коду	Модульна архітектура з розділенням на controller, service, model
НФ13	Безпека	Валідація вхідних даних	Перевірка діапазону значень від -999 до 999 на стороні сервера

Визначені функціональні та нефункціональні вимоги є основою для проєктування архітектури тренажеру та прийняття технічних рішень у наступних розділах. Виконання всіх обов'язкових функціональних вимог та нефункціональних характеристик є критерієм успішності розробки.

2 ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд існуючих навчальних тренажерів та програмного забезпечення з тематики структур даних і колекцій Java

Перед початком розробки власного тренажеру було здійснено огляд існуючого програмного забезпечення, яке так чи інакше стосується теми «Collections. TreeSet» у мові Java. Огляд охоплює як спеціалізовані навчальні тренажери, так і більш загальні інструменти для вивчення алгоритмів та структур даних.

Visualgo (visualgo.net) – це відомий безкоштовний веб-сайт для візуалізації алгоритмів та структур даних, розроблений командою Національного університету Сінгапуру [4]. Він містить візуалізацію бінарного дерева пошуку та червоно-чорного дерева, що є основою TreeSet у Java. Зображення інтерфейсу наведено на рисунку 1.1.

Рисунок 1.1 – Інтерфейс ресурсу Visualgo

VisuAlgo підтримує більше тридцяти структур даних та алгоритмів, серед яких бінарне дерево пошуку, AVL-дерево та червоно-чорне дерево. Для кожної структури реалізовано покроковий режим виконання операцій. Користувач може самостійно обирати швидкість анімації та переходити між кроками у будь-якому напрямку. Кожен крок супроводжується текстовим поясненням того що відбувається на даному етапі. Ресурс підтримує декілька мов інтерфейсу та надає можливість вводити власні вхідні дані для спостереження за поведінкою структури.

LeetCode (leetcode.com) – одна з найпопулярніших платформ для практики програмування [7]. Містить велику кількість задач на використання TreeSet та інших колекцій Java. Інтерфейс ресурсу наведено на рисунку 1.2.

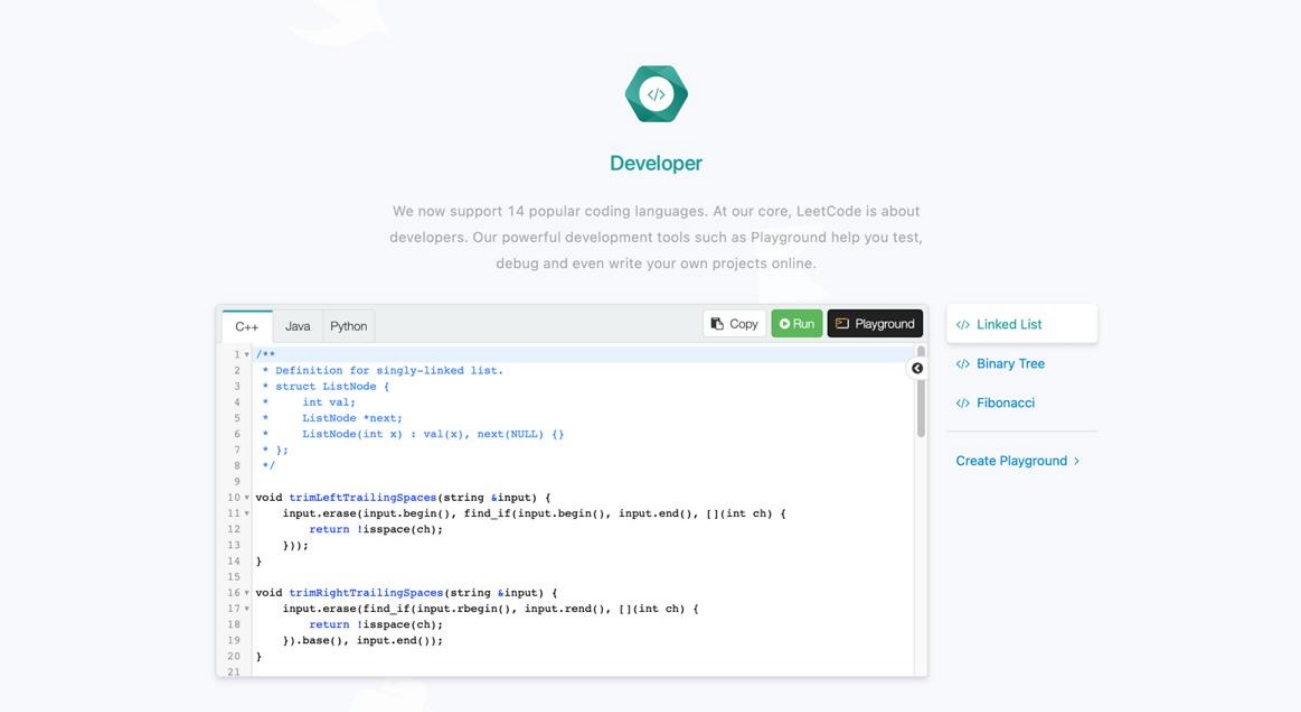

Рисунок 1.2 – Інтерфейс ресурсу LeetCode

LeetCode містить понад три тисячі задач, розподілених за рівнями складності – easy, medium та hard. Платформа підтримує виконання та автоматичну перевірку коду безпосередньо у браузері без встановлення середовища розробки. Для більшості задач доступна статистика виконання – час роботи та обсяг пам'яті порівняно з рішеннями інших користувачів. Значна частина задач передбачає

використання TreeSet та інших колекцій Java як оптимальний підхід до розв'язання. Платформа також містить розділ з навчальними матеріалами та можливість обговорення задач у спільноті.

Data Structure Visualizations (www.cs.usfca.edu/~galles/visualization) – навчальний сайт університету Сан-Франциско з візуалізацією різних структур даних [9], зокрема червоно-чорного дерева. Інтерфейс ресурсу наведено на рисунку 1.3.

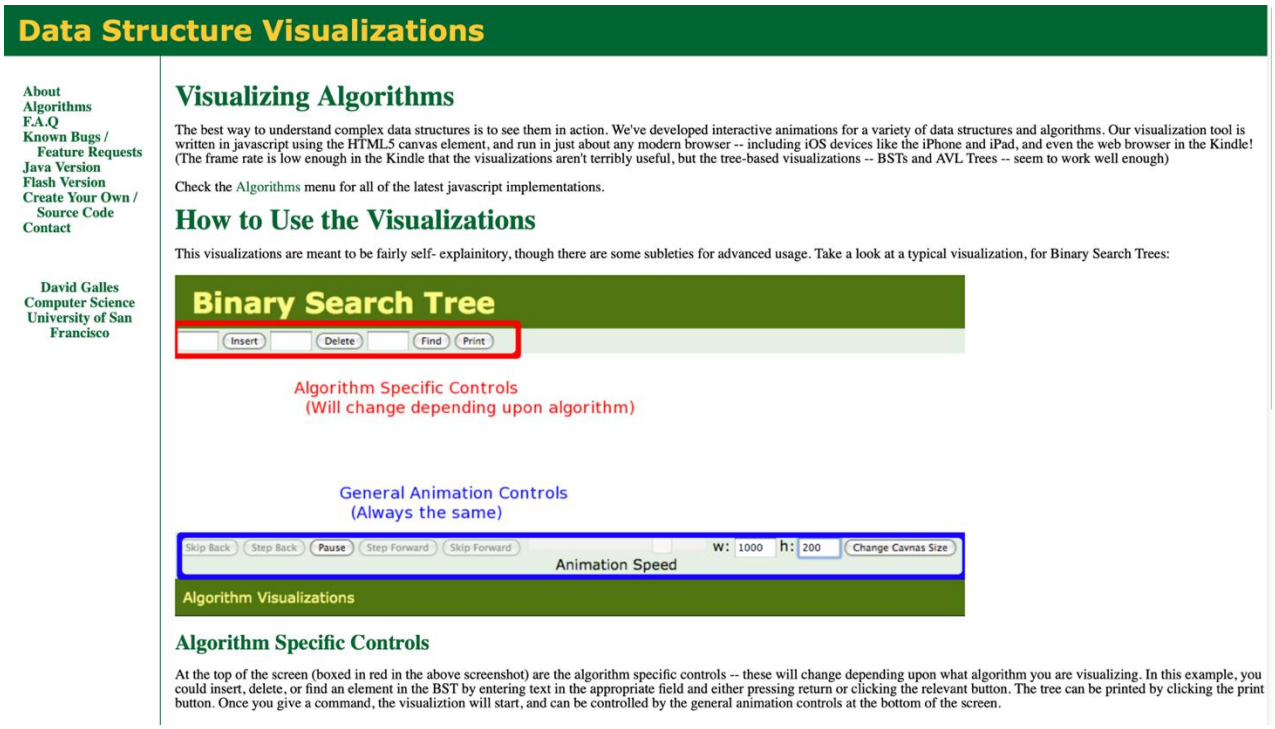

Рисунок 1.3 – Інтерфейс Data Structure Visualizations

Сайт університету Сан-Франциско містить візуалізацію понад п'ятдесяти структур даних та алгоритмів сортування. Для червоно-чорного дерева реалізовано операції вставки та видалення з відображенням проміжних станів під час балансування. Ресурс є повністю безкоштовним та не потребує реєстрації. Інтерфейс побудований за єдиним принципом для всіх структур – поле введення значення та кнопки операцій, що спрощує навігацію між різними розділами сайту.

Окремої уваги заслуговують онлайн-платформи загального призначення, що використовуються у навчанні програмування.

На платформі Coursera доступні курси з алгоритмів і структур даних від провідних університетів світу, зокрема Princeton University («Algorithms, Part I/II» by Robert Sedgewick) та UC San Diego (рис. 1.4).

Ці курси містять відеолекції, автоматично перевірені завдання та теоретичні матеріали. Проте їх основний недолік з точки зору даної роботи – відсутність специфічної прив'язки до Java Collections Framework та зокрема до класу TreeSet. Студент отримує загальні знання про дерева, але не розуміє як ці знання відображаються у конкретних методах Java API.

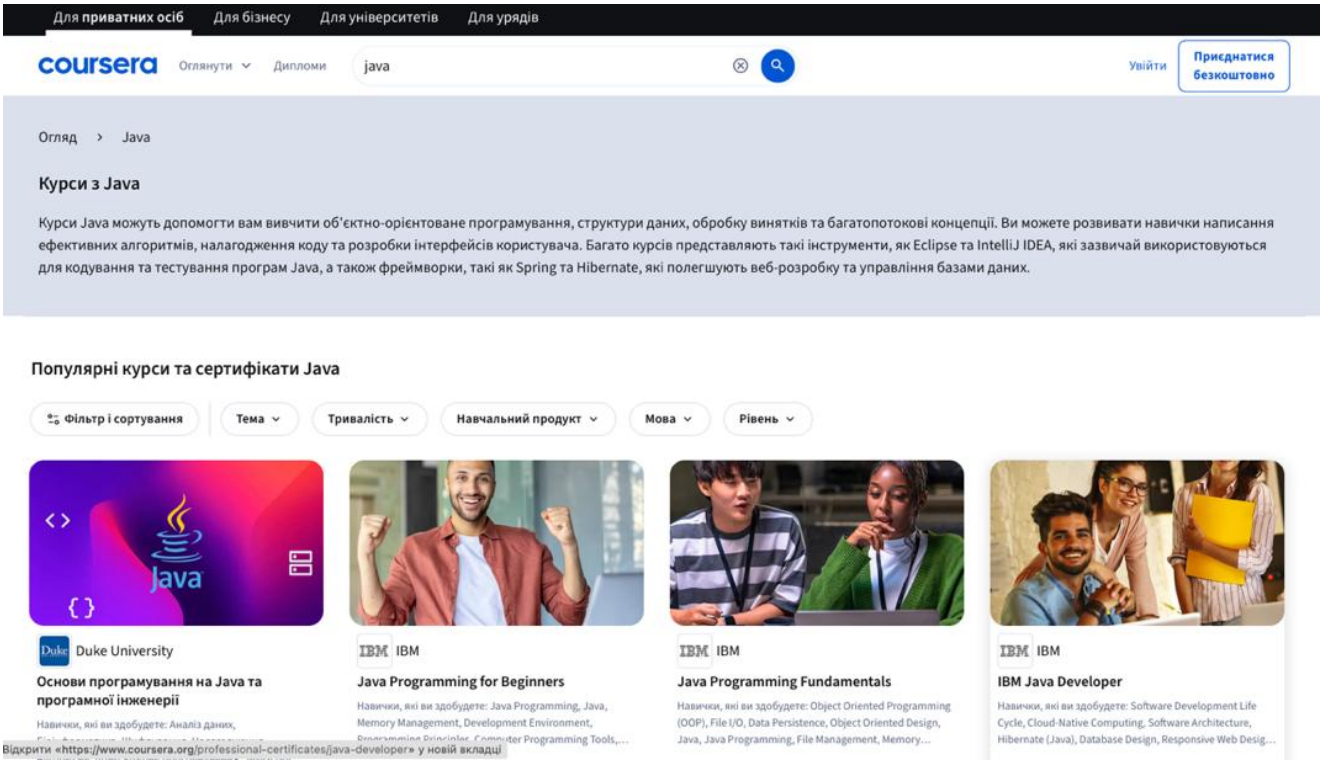

Рисунок 1.4 – Інтерфейс Coursera

Платформа Prometheus (prometheus.org.ua) є провідною українською освітньою онлайн-платформою, що містить курси від вітчизняних університетів та викладачів (рис. 1.5). На Prometheus розміщено курси з основ програмування та веб-розробки, однак тема Java Collections Framework представлена недостатньо – наявні курси охоплюють переважно базовий синтаксис Java та введення в ООП.

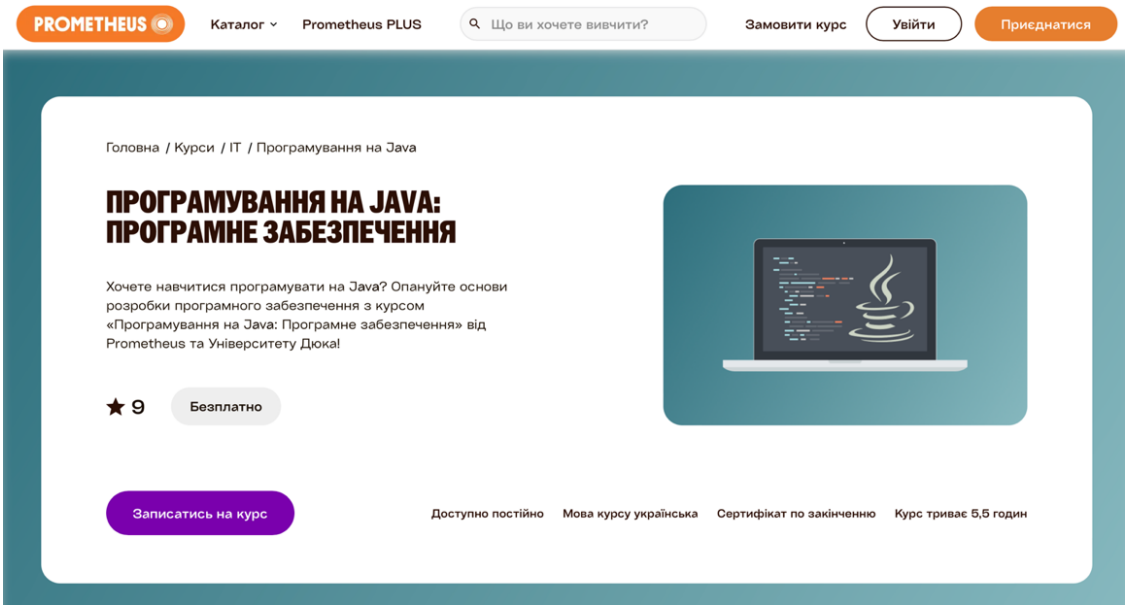

Рисунок 1.5 – Інтерфейс Data Structure Coursera

Geeks for Geeks (geeksforgeeks.org) – популярний англомовний ресурс з великою кількістю статей про структури даних та алгоритми, зокрема детальними поясненнями роботи червоно-чорного дерева та класу TreeSet у Java (рис. 1.6). Ресурс містить теоретичні пояснення, приклади коду та задачі з рішеннями.

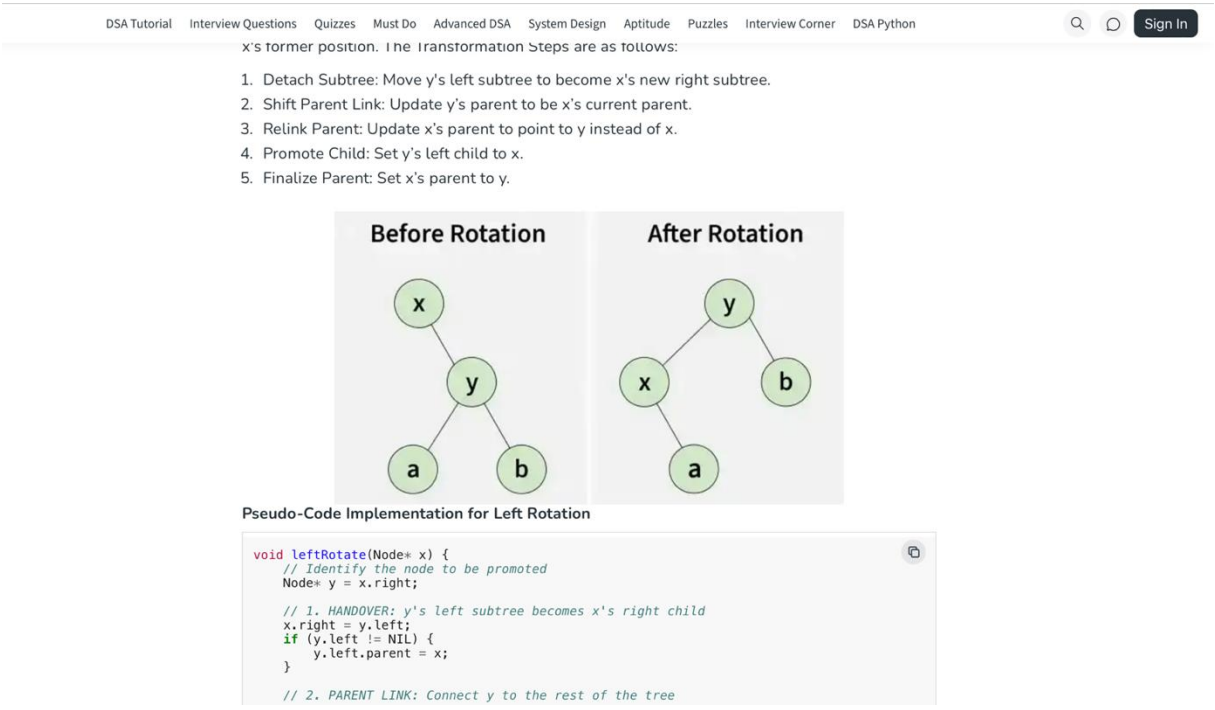

Рисунок 1.6 – Інтерфейс Geeks for Geeks

Geeks for Geeks містить детальні статті про роботу класу TreeSet у Java – від базових операцій та навігаційних методів до порівняння з іншими реалізаціями Set та розбору внутрішньої реалізації через TreeMap. Кожна стаття містить повні приклади Java-коду з вхідними даними та очікуваним виводом. Ресурс також має розділ з практичними задачами та їх розв'язками. Матеріал регулярно оновлюється відповідно до нових версій Java та актуальних підходів до вирішення задач.

2.2 Позитивні аспекти оглянутих програмних розробок

Кожен із розглянутих інструментів має певні переваги, які є сенс врахувати при розробці власного тренажеру.

Головною перевагою VisuAlgo є висока якість анімацій та гнучкий інтерфейс управління візуалізацією. Користувач може самостійно обирати швидкість анімації, переходити між кроками вперед і назад, а також вводити власні вхідні дані. Для кожної операції доступний текстовий опис того, що відбувається на поточному кроці. Ресурс підтримує кілька мов інтерфейсу та є повністю безкоштовним. Покрокова навігація між кроками алгоритму є функцією, яку варто реалізувати у власному тренажері.

LeetCode демонструє ефективну модель практичного навчання через задачі з автоматичною перевіркою. Платформа підтримує виконання коду у браузері без встановлення середовища розробки, що значно знижує поріг входу для студентів. Система рівнів складності (easy/medium/hard) та теги задач дозволяють студенту самостійно обирати відповідний рівень навантаження. Позитивним є також наявність обговорень задач у спільноті. Студент може порівняти своє рішення з рішеннями інших. Модель рівнів складності та автоматична перевірка є елементами, які необхідно реалізувати у банку задач власного тренажеру.

Data Structure Visualizations демонструє вдалий підхід до покрокового відображення операцій балансування – при вставці або видаленні елемента видно не лише кінцевий результат, а й проміжні стани дерева під час поворотів та

перефарбування вузлів. Це допомагає студенту зрозуміти що саме відбувається при балансуванні, а не лише що відбулось.

Coursera та edX демонструють ефективну організацію навчального контенту: матеріал структурований у модулі та тижні, що допомагає студенту планувати навчання. Автоматично перевірені завдання з детальним зворотним зв'язком та форуми підтримки спільноти є позитивними аспектами, що сприяють ефективному навчанню.

Prometheus демонструє важливість україномовного контенту для підвищення доступності матеріалу для вітчизняної аудиторії. Студенти, яким важко сприймати технічну інформацію іноземною мовою, отримують можливість вивчати матеріал рідною мовою без мовного бар'єру. Ця перевага є визначальною для цільової аудиторії розроблюваного тренажеру.

Geeks for Geeks демонструє ефективну подачу технічного матеріалу з прикладами реального коду. Кожне теоретичне твердження ілюструється робочим Java-кодом, що можна безпосередньо запустити. Наявність вихідних даних та очікуваного результату для кожного прикладу допомагає студенту самостійно перевірити своє розуміння. Підхід «теорія + приклад коду» є основою для теоретичного модуля власного тренажеру.

2.3 Вади оглянутих програмних розробок

Попри наявні переваги, кожен із розглянутих інструментів має суттєві недоліки, які роблять їх неповноцінними для вирішення поставленої задачі.

VisuAlgo має кілька принципових недоліків з точки зору навчання темі TreeSet у Java. По-перше, ресурс демонструє абстрактне червоно-чорне дерево без будь-якого зв'язку з Java API. Тобто студент бачить як дерево балансується, але не розуміє що за цим стоїть виклик методу add() класу TreeSet. По-друге, відсутня методична частина – немає теоретичних пояснень, задач чи тестів. Інтерфейс повністю англійською мовою, що є бар'єром для частини студентів. Для входу в

режим «training» деяких структур необхідна авторизація, що ускладнює доступ. Ресурс не відстежує прогрес користувача – кожен сеанс починається з нуля.

LeetCode орієнтований на досвідчених розробників, а не на студентів, що вивчають тему вперше. Платформа не містить теоретичного матеріалу – передбачається, що користувач вже знає тему. Відсутність візуалізації внутрішньої структури означає, що студент не розуміє чому TreeSet повертає саме такий результат, а не інший. Повний функціонал платформи є платним – частина задач та пояснень доступна лише за підпискою, що є неприйнятним для навчального закладу. Крім того, LeetCode не локалізований українською мовою.

Data Structure Visualizations має морально застарілий інтерфейс та не оновлювався протягом тривалого часу. Відсутня прив'язка до Java API, немає навчального контенту поза межами самої візуалізації, задач чи тестування. Лише англійська мова. На відміну від VisuAlgo, навігація між кроками алгоритму менш зручна – немає можливості повернутися на попередній крок.

На Coursera курси не прив'язані до конкретних Java-класів Collections Framework. Студент дізнається про червоно-чорне дерево як абстрактну структуру, але не бачить зв'язку з методами TreeSet. Більшість якісних курсів є платними або доступні лише в рамках підписки. Курси розраховані на тижні навчання, тоді як для студента потрібен компактний інструмент для вивчення конкретної теми в рамках семестру. Мала кількість україномовного контенту є додатковим обмеженням.

Prometheus не має курсів, що детально охоплюють тему Collections Framework та TreeSet. Наявний контент з Java охоплює здебільшого базовий синтаксис та введення в ООП. Відсутність інтерактивної візуалізації структур даних є принциповим недоліком. Платформа не надає викладачу інструментів для відстеження прогресу конкретних студентів у реальному часі.

Geeks for Geeks є довідником, а не навчальним тренажером – відсутня система задач з підказками, автоматична перевірка відповідей та відстеження прогресу. Ресурс є виключно англійською мовою. Відсутність інтерактивної візуалізації означає що студент читає про балансування, але не бачить його. Оскільки ресурс є

публічним довідником, він не може слугувати основним навчальним інструментом у рамках курсу.

2.4 Необхідність та актуальність розробки тренажеру з теми «Collections. TreeSet»

Узагальнення результатів інформаційного огляду дозволяє зробити висновок. На ринку навчального програмного забезпечення існує явна прогалина – відсутній спеціалізований україномовний інтерактивний тренажер, який поєднував би у собі теоретичний матеріал з прив'язкою до Java API, покрокову анімацію балансування червоно-чорного дерева, систему задач з підказками, тестування з адаптивною вибіркою питань та відстеження прогресу студента в єдиному середовищі.

Необхідність розробки такого тренажеру обґрунтовується кількома факторами. З педагогічної точки зору, дослідження показують що активні методи навчання – де студент діє, а не лише спостерігає – підвищують ефективність засвоєння матеріалу в 1.5-2 рази порівняно з пасивним читанням [10]. Тренажер, що вимагає від студента самостійно виконувати операції над деревом та розв'язувати задачі, реалізує саме такий активний підхід. З методичної точки зору, поєднання теорії, візуалізації та практики в одному інструменті відповідає принципу навчання від загального до конкретного – студент спочатку читає про TreeSet, потім бачить як він працює, потім самостійно розв'язує задачі.

Актуальність розробки підтверджується також тенденціями розвитку дистанційної освіти в Україні. Після 2020 року більшість університетів перейшли на гібридний або дистанційний формат, що суттєво збільшило потребу в якісних інтерактивних навчальних інструментах [11]. Традиційні підручники та PDF-конспекти не можуть повністю замінити очний контакт студента з викладачем – натомість це може зробити добре спроєктований тренажер з інтелектуальним зворотним зв'язком [12].

Розроблюваний тренажер відрізняється від існуючих рішень за чотирма ключовими параметрами:

– україномовний інтерфейс усуває мовний бар'єр для цільової аудиторії;

– пряма прив'язка до Java API – кожна операція у модулі візуалізації відповідає конкретному виклику методу класу TreeSet, що дозволяє студенту побачити зв'язок між кодом і поведінкою структури даних;

– покрокова анімація балансування дозволяє студенту побачити не лише кінцевий результат операції, а й проміжні стани дерева під час поворотів та перефарбування;

– відстеження прогресу та адміністративна панель для викладача перетворюють тренажер на повноцінний навчальний інструмент, що може бути інтегрований у навчальний процес кафедри.

Порівняльний аналіз функціональних можливостей найбільш релевантних розглянутих інструментів наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз функціональних можливостей найбільш релевантних розглянутих інструментів

Критерій	VisuAlgo	LeetCode	Data Structure	Geeks for Geeks
Теоретичний матеріал	Ні	Ні	Ні	Так
Прив'язка до Java API	Ні	Частково	Ні	Так
Візуалізація TreeSet	Так	Ні	Так	Ні
Покрокова анімація	Так	Ні	Частково	Ні
Задачі з підказками	Ні	Так	Ні	Ні
Тестування	Ні	Так	Ні	Ні
Прогрес студента	Ні	Так	Ні	Ні
Панель викладача	Ні	Ні	Ні	Ні
Українська мова	Ні	Ні	Ні	Ні
Локальна робота	Ні	Ні	Ні	Ні
Безкоштовно	Так	Частково	Так	Частково

За результатами огляду визначено такі ключові вимоги до розроблюваного тренажеру, що відрізняють його від існуючих рішень: реалізація на українській мові, пряма прив'язка до Java API та методів класу TreeSet, покрокова анімація балансування червоно-чорного дерева, система задач з ієрархічними підказками та автоматичною перевіркою, відстеження прогресу студента між сеансами, адміністративна панель для викладача, запуск без встановлення додаткового програмного забезпечення.

3 ТЕОРЕТИЧНА ЧАСТИНА

3.1 Проєктування архітектури навчального тренажеру

Проєктування архітектури тренажеру розпочинається з визначення загальної структури системи. З огляду на вимоги до доступності – можливість запустити застосунок на різних операційних системах без додаткового налаштування та відкрити у будь-якому браузері – обрано трирівневу клієнт-серверну архітектуру.

Клієнт-серверна архітектура передбачає чіткий розподіл системи на два незалежні рівні: клієнт, що відповідає за відображення даних та взаємодію з користувачем, та сервер, що відповідає за бізнес-логіку та зберігання даних. Такий підхід забезпечує незалежність від платформи – клієнтська частина виконується у браузері на будь-якому пристрої, тоді як серверна частина виконується на одному комп'ютері [13].

Архітектура тренажеру складається з трьох рівнів. Рівень подання реалізований засобами HTML5, CSS3 та JavaScript без використання зовнішніх фреймворків. Це забезпечує максимальну сумісність з браузерами та відсутність залежностей від сторонніх пакетів на стороні клієнта. Рівень бізнес-логіки реалізований на Java 17 з використанням Spring Boot 3.2. Він відповідає за обробку HTTP-запитів, виконання операцій над деревом, перевірку відповідей та формування відповідей у форматі JSON. Рівень даних реалізований на основі бази даних SQLite, що зберігає інформацію про користувачів, їх прогрес та результати тестування. Взаємодія між рівнем подання та рівнем бізнес-логіки здійснюється через REST API у форматі JSON. Схема загальної архітектури наведена на рисунку 3.1.

Серверна частина організована за шаблоном проєктування MVC, адаптованим для REST-архітектури, і складається з трьох шарів [14]. Шар контролерів (controller) приймає HTTP-запити та делегує обробку відповідним сервісам. У тренажері реалізовано шість контролерів: VisualizerController для роботи з деревом, TaskController для задач, TestController для тестування, AuthController для

авторизації, ProgressController для прогресу студента та AdminController для адміністративної панелі.

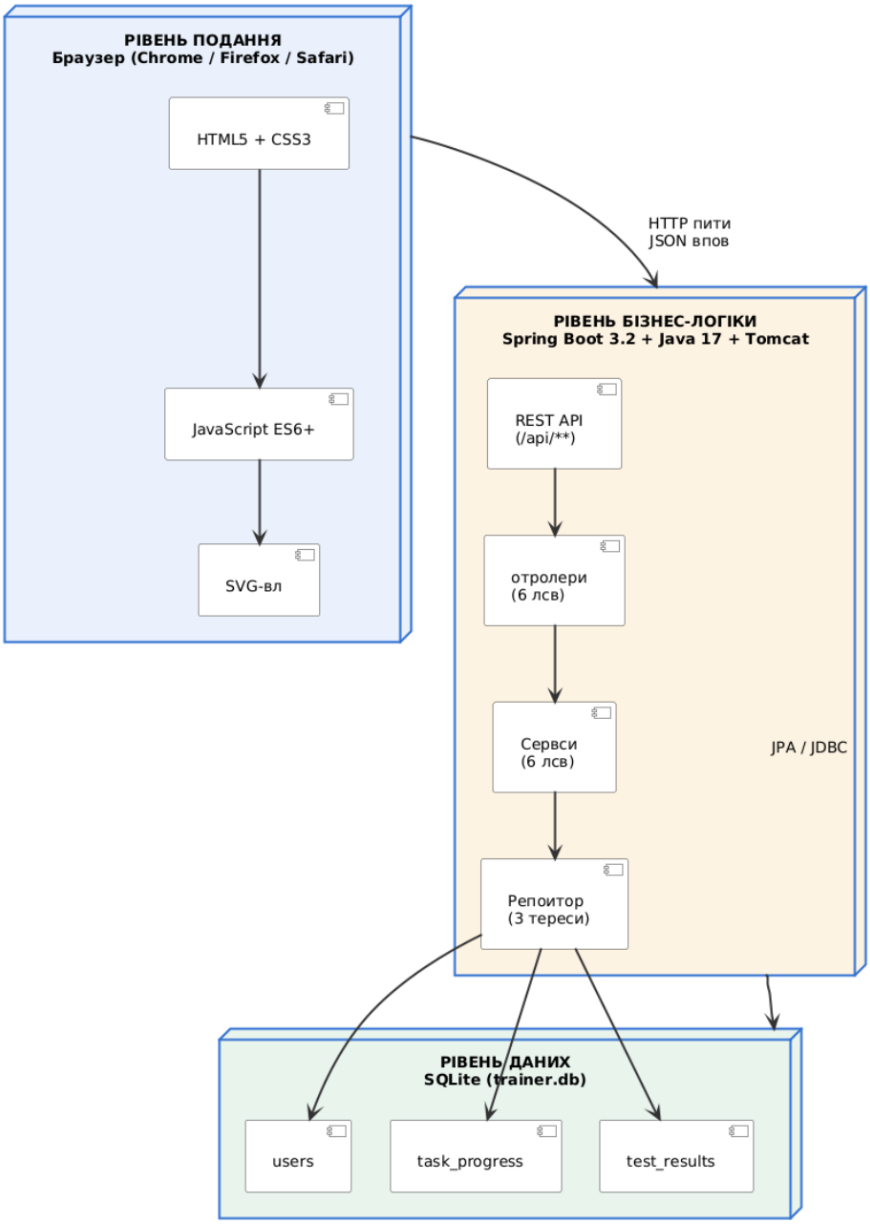

Рисунок 3.1 – Схема загальної архітектури

Шар сервісів (service) містить всю бізнес-логіку: TreeVisualizerService реалізує власне червоно-чорне дерево з покроковою анімацією, TaskService та TestService містять навчальний контент, UserService та ProgressService управляють даними користувачів, CompareService реалізує порівняльну візуалізацію. Шар моделей (model) містить класи-сутності бази даних: User, TaskProgress, TestResult, а також

допоміжні класи TreeNode, Task, TestQuestion для передачі даних між шарами. Деталізацію серверного рівня наведено на рисунку 3.2.

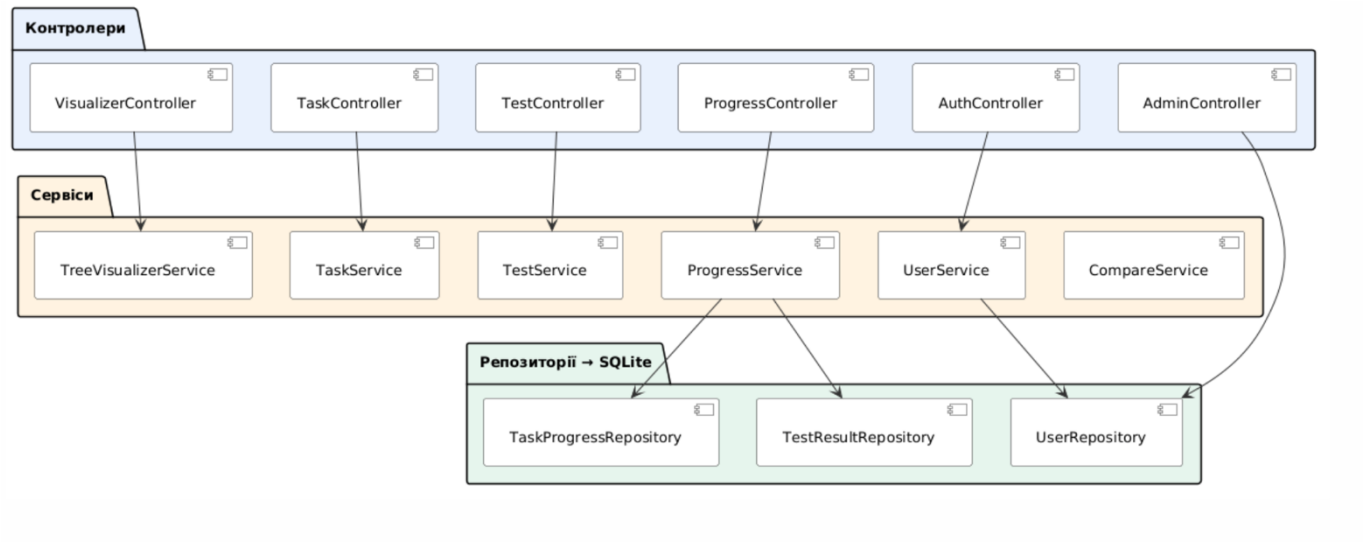

Рисунок 3.2 – Деталізація серверного рівня

Клієнтська частина організована у сім JavaScript-модулів відповідно до функціональних модулів тренажеру. Модуль app.js відповідає за навігацію між вкладками та містить спільні функції для HTTP-запитів. Модулі theory.js, visualizer.js, tasks.js, tests.js реалізують відповідні навчальні модулі. Модулі auth.js, progress.js, admin.js, compare.js реалізують функціонал авторизації, відстеження прогресу, адміністрування та порівняння.

Структура проєкту організована наступним чином. Серверна частина розміщена у директорії backend/src/main/java/ua/puet/treeset/ і поділена на підпакети controller, service, model, repository та config. Клієнтська частина розміщена у директорії backend/src/main/resources/static/ і містить файли index.html, директорії css/ та js/. База даних SQLite зберігається у файлі trainer.db у кореневій директорії backend/.

Архітектурною особливістю є те що серверна частина роздає статичні файли клієнтської частини безпосередньо через Spring Boot – окремого веб-сервера не потрібно. При запуску команди java -jar вмикається вбудований сервер Tomcat, який

одночасно обробляє як API-запити за адресами /api/*, так і роздає статичні файли за всіма іншими адресами. Це забезпечує запуск всього застосунку однією командою.

Безпека забезпечується через Spring Security. Авторизація реалізована на основі сесій – при успішному вході ідентифікатор та роль користувача зберігаються у серверній сесії. Ендпоінти адміністративної панелі /api/admin/** доступні лише для користувачів з роллю TEACHER, ендпоінти прогресу /api/progress/** – для авторизованих користувачів будь-якої ролі. Публічні ендпоінти (теорія, візуалізація, задачі, тест) доступні без авторизації.

3.2 Графічне представлення архітектури (UML-діаграми)

Для детального графічного представлення архітектури тренажеру розроблено п’ять UML-діаграми: діаграму класів, діаграму варіантів використання, діаграму послідовності та діаграму розгортання [15].

Діаграма класів відображає повну структуру класів серверної частини з атрибутами, методами та зв'язками між ними. Класи організовані у чотири пакети: model, repository, service та controller.

Пакет model містить шість класів. Три з них є сутностями бази даних – User, TaskProgress та TestResult. Вони відображаються на відповідні таблиці бази даних через анотації JPA. Клас User зберігає дані профілю студента або викладача. Клас TaskProgress фіксує результат розв'язання конкретної задачі – чи розв'язана вона, скільки підказок використано та скільки спроб зроблено. Клас TestResult зберігає результат одного проходження тесту з відсотком правильних відповідей та оцінкою. Між TaskProgress та User і між TestResult та User встановлено асоціацію @ManyToOne – один студент може мати багато результатів задач і тестів. Три допоміжні класи TreeNode, Task та TestQuestion не зберігаються у базі даних і використовуються лише для передачі даних між шарами. Пакет repository містить три інтерфейси що розширюють JpaRepository – вони забезпечують доступ до відповідних таблиць бази даних.

На рисунку 3.3 наведено класи пакетів model та repository.

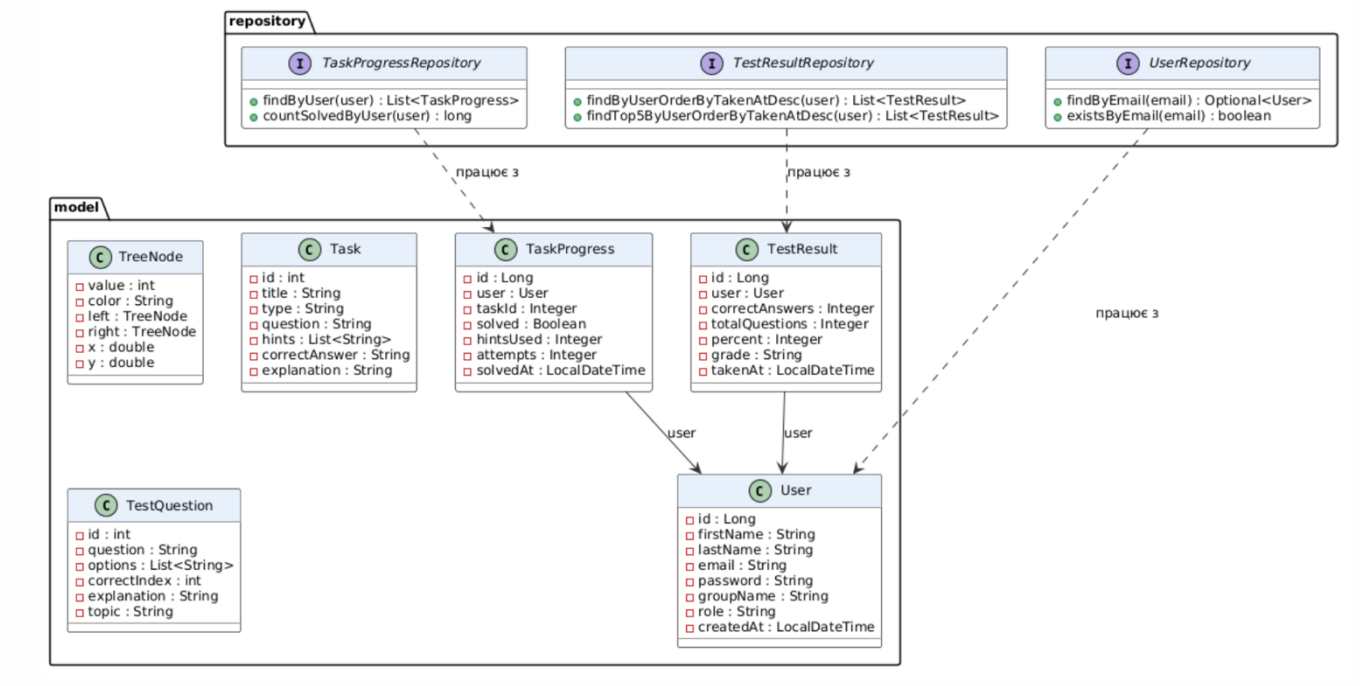

Рисунок 3.3 – Класи моделей та репозиторіїв

На рисунку 3.4 наведено класи пакетів service та controller. Пакет service містить шість класів з бізнес-логікою системи. TreeVisualizerService є центральним класом тренажеру – він реалізує власне червоно-чорне дерево та покрокову анімацію. TaskService та TestService є статичними сховищами навчального контенту. UserService та ProgressService забезпечують роботу з даними користувачів. CompareService реалізує паралельне порівняння трьох реалізацій Set. Пакет controller містить шість контролерів, кожен з яких обробляє HTTP-запити відповідної функціональної області та делегує виконання відповідному сервісу.

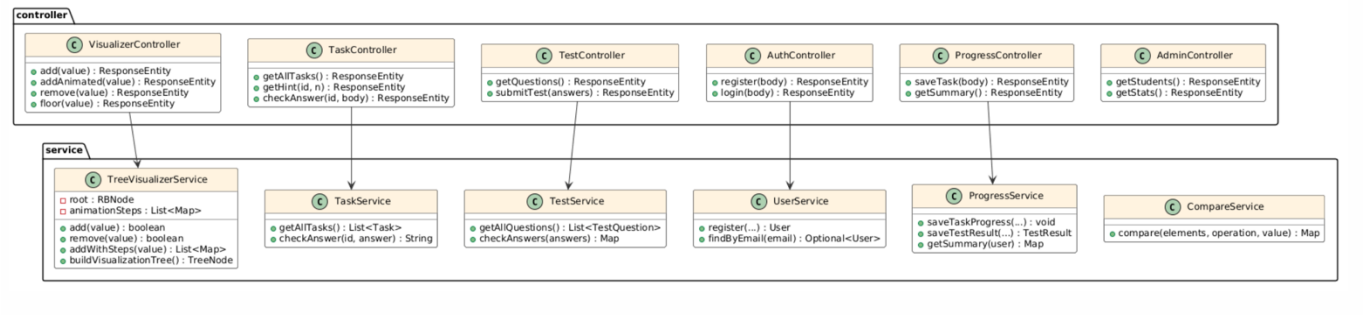

Рисунок 3.4 – Класи сервісів та контролерів

Діаграма варіантів використання відображає взаємодію двох акторів – Студента та Викладача – з функціоналом системи.

Студент без авторизації може переглядати теоретичний матеріал, виконувати операції над деревом, переглядати анімацію балансування, порівнювати реалізації Set, розв'язувати задачі та проходити тест. Авторизований студент додатково отримує доступ до модуля прогресу, де бачить статистику своїх результатів.

Викладач після авторизації отримує доступ до адміністративної панелі, що включає перегляд списку всіх студентів з їх результатами та загальну статистику по групах. Відношення extend між варіантом використання «Увійти в систему» та «Переглядати прогрес» і «Переглядати результати студентів» означає що ці функції розширюють базовий сценарій авторизації.

Діаграму варіантів використання наведено на рисунку 3.5.

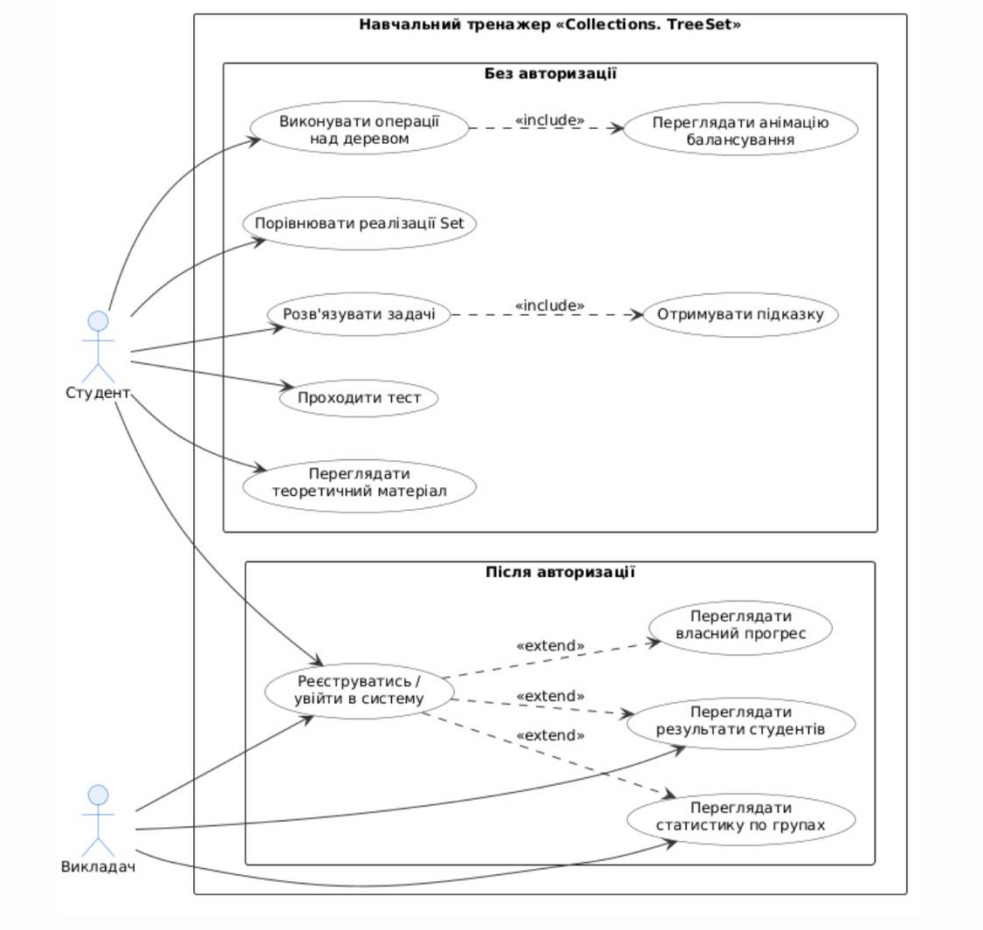

Рисунок 3.5 – Діаграма варіантів використання

Діаграма послідовності детально відображає взаємодію між компонентами системи під час виконання операції add() з покроковою анімацією – найбільш складного сценарію у тренажері (рис. 3.6).

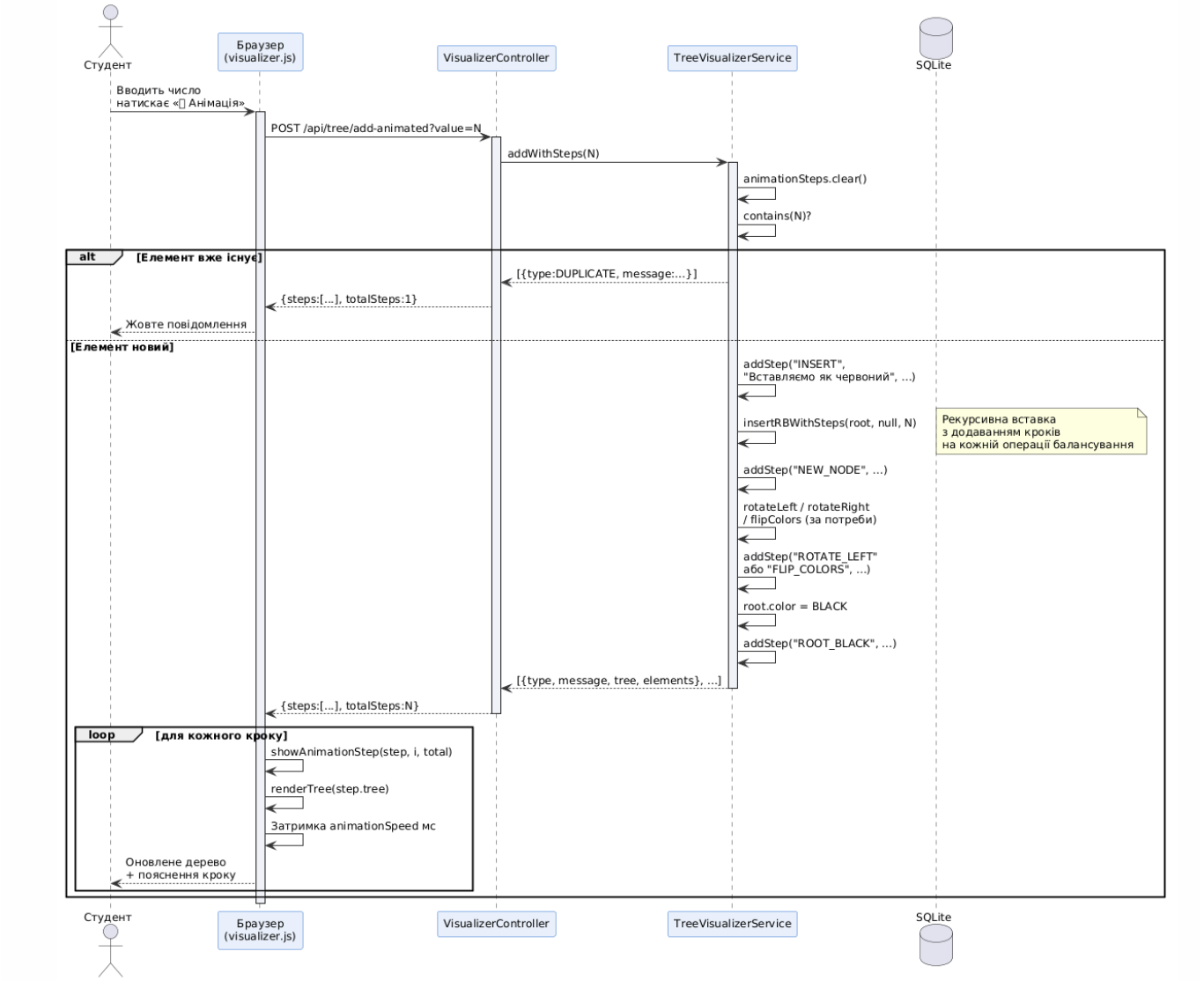

Рисунок 3.6 – Діаграма послідовності операції add() з анімацією

Сценарій починається з того, що студент вводить числове значення та натискає кнопку «Анімація». Браузер надсилає POST-запит до /api/tree/add-animated?value=N. VisualizerController делегує обробку методу addWithSteps() сервісу TreeVisualizerService.

Сервіс виконує такі дії: очищає список кроків анімації, перевіряє чи не є елемент дублікатом. Якщо елемент новий – додає крок «INSERT», рекурсивно вставляє елемент у дерево через insertRBWithSteps(), під час якого додає кроки для кожної операції балансування (ROTATE_LEFT, ROTATE_RIGHT, FLIP_COLORS за потреби), перефарбовує корінь у чорний та додає фінальний крок ROOT_BLACK.

Сервіс повертає список кроків у форматі JSON, де кожен крок містить тип операції, текстове пояснення, поточний стан дерева з координатами вузлів та поточний список елементів. Браузер отримує відповідь та відтворює анімацію – для кожного кроку відмальовує дерево на SVG-полотні, відображає пояснення та чекає animationSpeed мілісекунд перед наступним кроком.

Діаграма також відображає альтернативний сценарій – якщо елемент вже існує, сервіс повертає один крок з повідомленням про дублікат без зміни дерева.

Діаграма розгортання відображає фізичне розміщення компонентів системи (рис. 3.7).

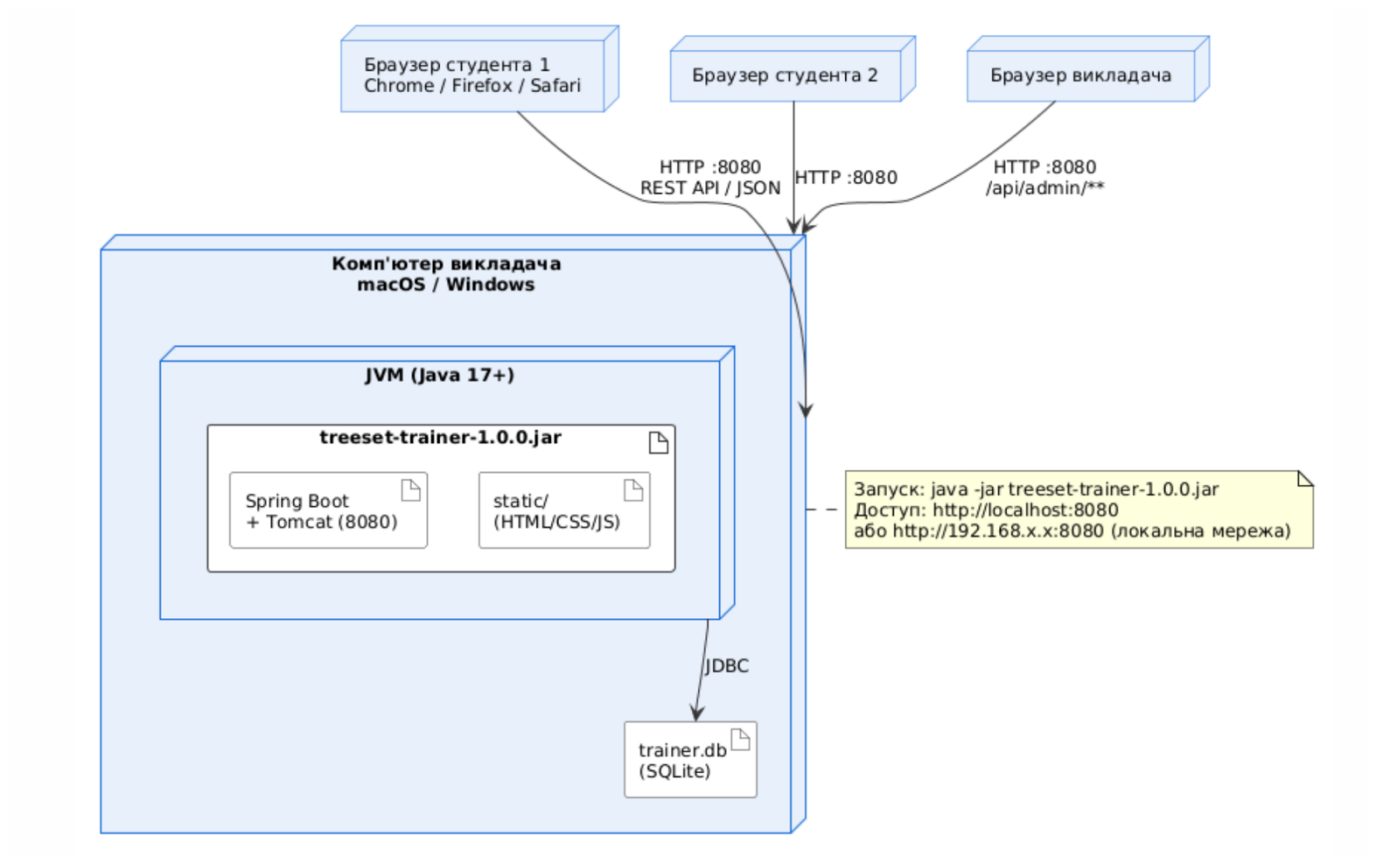

Рисунок 3.7 – Діаграма розгортання

Весь застосунок розгортається на одному комп'ютері – сервері. На сервері запускається JVM з виконуваним jar-файлом treeset-trainer-1.0.0.jar, який містить Spring Boot, вбудований Tomcat та статичні файли клієнтської частини. База даних SQLite зберігається у файлі trainer.db у тій самій директорії.

Браузери студентів та викладача підключаються до сервера через HTTP на порту 8080. При роботі в локальній мережі університету адреса має вигляд http://192.168.x.x:8080, при локальному запуску – http://localhost:8080. Адміністративна панель доступна лише для користувачів з роллю TEACHER, що забезпечується Spring Security на рівні серверних ендпоінтів /api/admin/**.

3.3 Обґрунтування вибору програмних засобів для реалізації тренажеру

Вибір технологічного стеку для розробки тренажеру здійснювався на основі чотирьох критеріїв: відповідність темі розробки, крос-платформеність, простота розгортання та наявність необхідних інструментів для реалізації всіх функціональних вимог. Для кожної складової системи розглядалось кілька альтернатив, після чого обиралось найбільш відповідне рішення.

Для реалізації серверної частини розглядались три мови: Java, Python та Node.js [16]. Python має простий синтаксис та великий вибір бібліотек для веброзробки, зокрема Flask та Django. Node.js забезпечує єдину мову для серверної та клієнтської частини і підходить для побудови асинхронних застосунків. Проте обидва варіанти були відхилені на користь Java.

Java обрана з трьох причин. По-перше, тренажер розробляється для навчання саме мові Java та її бібліотеці Collections Framework – тому використання тієї ж мови для реалізації є методично виправданим. Студент може вивчати вихідний код тренажеру як додатковий приклад об'єктно-орієнтованого програмування з реальним застосуванням тих самих класів які він вивчає. По-друге, Java забезпечує справжню крос-платформеність через принцип «write once, run anywhere» – один і той самий jar-файл запускається на macOS, Windows та Linux без жодних змін у коді

[17]. По-третє, Java 17 є версією з довгостроковою підтримкою (LTS), що гарантує стабільність та безпеку застосунку протягом тривалого часу.

Для побудови серверної частини розглядались три фреймворки: Spring Boot, Jakarta EE та Micronaut [18]. Jakarta EE є потужним стандартом корпоративної розробки, але вимагає окремого сервера додатків (WildFly, GlassFish) що суперечить вимозі НФ6 про запуск без додаткового налаштування. Micronaut є сучасним легковісним фреймворком з швидким стартом, але має меншу спільноту та документацію.

Spring Boot 3.2 обраний з двох ключових причин. По-перше, він включає вбудований сервер Tomcat – весь застосунок запускається однією командою java -jar без необхідності окремо встановлювати та налаштовувати веб-сервер. По-друге, Spring Boot суттєво спрощує розробку REST API через анотації @RestController, @GetMapping, @PostMapping, що скорочує обсяг шаблонного коду та підвищує читабельність [19]. Spring Security інтегровано для авторизації – воно надає готові механізми захисту ендпоінтів, хешування паролів через BCrypt та управління сесіями без написання власного коду безпеки.

Spring Data JPA використовується для роботи з базою даних [20]. Цей підхід дозволяє описувати запити до бази декларативно через інтерфейси-репозиторії без написання SQL-коду вручну. Наприклад, метод findByUserOrderByTakenAtDesc автоматично генерує відповідний SQL-запит на основі назви методу, що значно прискорює розробку та зменшує ймовірність помилок.

Для зберігання даних розглядались три варіанти: PostgreSQL, H2 та SQLite [21]. PostgreSQL є потужною реляційною СУБД з широкими можливостями, але потребує окремого серверного процесу та налаштування – це суперечить вимозі про простоту розгортання. H2 є вбудованою базою даних для Java з режимом роботи у пам'яті, але дані зникають при перезапуску застосунку якщо не налаштовано збереження у файл.

SQLite обрана як оптимальний варіант для навчального тренажеру. На відміну від серверних СУБД, SQLite не потребує окремого процесу – вся база зберігається в одному файлі trainer.db у директорії застосунку. Це повністю відповідає вимозі НФ6

– запуск без додаткового налаштування. Для навчального тренажеру з невеликою кількістю одночасних користувачів продуктивності SQLite цілком достатньо. Підключення реалізується через бібліотеку sqlite-jdbc та діалект hibernate-community-dialects для Hibernate.

Для реалізації клієнтської частини розглядались JavaScript-фреймворки React, Vue та Angular, а також варіант з чистим JavaScript без фреймворків [22]. React є найпопулярнішим вибором для веб-застосунків, Vue відзначається низьким порогом входу, Angular надає повноцінний фреймворк з усіма інструментами. Проте від усіх трьох варіантів свідомо відмовились на користь чистого JavaScript.

Для навчального тренажеру достатньо нативного функціоналу браузера ES6+, а відсутність npm-залежностей суттєво спрощує збірку та підтримку. Використання фреймворку вимагало б встановлення Node.js та збірки фронтенду через webpack або vite, що ускладнює процес розробки без суттєвих переваг для проєкту такого масштабу. Клієнтська частина організована у сім модулів через патерн IIFE (Immediately Invoked Function Expression), що забезпечує інкапсуляцію та уникнення конфліктів імен у глобальному просторі.

SVG (Scalable Vector Graphics) обрано для відмальовування дерева [23]. Векторний формат масштабується без втрати якості при будь-якому розмірі екрана, підтримується всіма сучасними браузерами без додаткових бібліотек та дозволяє динамічно змінювати вузли через JavaScript DOM API.

Apache Maven 3.9 обрано як систему збірки бо це стандарт для Java-проєктів з підтримкою в усіх популярних IDE та зрозумілою структурою залежностей у файлі pom.xml [24]. Як середовище розробки використовується Visual Studio Code з розширеннями для Java – Extension Pack for Java та Spring Boot Extension Pack. VS Code є безкоштовним редактором з відкритим вихідним кодом, що підтримує Java, JavaScript, HTML та CSS в єдиному середовищі. Це особливо зручно для розробки даного тренажеру оскільки серверна та клієнтська частини написані різними мовами – VS Code дозволяє працювати з усіма файлами проєкту без перемикання між інструментами.

Повний перелік використаних технологій та їх версій наведено у таблиці 3.1.

Таблиця 3.3 – Перелік використаних технологій та їх версій

№	Технологія	Версія	Призначення
1	Java	17 LTS	Мова програмування серверної частини
2	Spring Boot	3.2.0	Серверний фреймворк з вбудованим Tomcat
3	Spring Security	6.2	Авторизація та захист ендпоінтів
4	Spring Data JPA	3.2	Робота з базою даних через репозиторії
5	Hibernate	6.4	ORM для маппінгу Java-об'єктів на таблиці
6	SQLite JDBC	3.45.1.0	Драйвер бази даних SQLite
7	Apache Maven	3.9	Система збірки та управління залежностями
8	HTML5	-	Розмітка клієнтської частини
9	CSS3	-	Стилізація інтерфейсу тренажеру
10	JavaScript	ES6+	Логіка клієнтської частини
11	SVG	1.1	Векторна візуалізація дерева
12	IntelliJ IDEA	Community	Середовище розробки

Усі обрані технології є відкритими та безкоштовними, що відповідає принципу доступності навчального тренажеру. Обраний стек забезпечує виконання всіх нефункціональних вимог – крос-платформеність (НФ4), запуск без додаткового налаштування (НФ6), підтримку сучасних браузерів (НФ5) та модульну структуру коду (НФ12).

4 ПРАКТИЧНА ЧАСТИНА

4.1 Опис процесу програмної реалізації елементів тренажеру

Програмна реалізація тренажеру здійснювалась поетапно відповідно до архітектури описаної у розділі 3. На першому етапі було реалізовано базовий функціонал – теоретичний модуль, модуль візуалізації, модуль задач та базовий модуль тестування. На другому етапі базовий функціонал було розширено: додано авторизацію, збереження прогресу у базі даних SQLite, покрокову анімацію балансування, адміністративну панель та модуль порівняння реалізацій Set.

Реалізація серверної частини розпочиналась зі створення структури проєкту через Maven. У файлі pom.xml визначено всі залежності:

– spring-boot-starter-web для REST API;

– spring-boot-starter-security для авторизації;

– spring-boot-starter-data-jpa для роботи з базою даних;

– sqlite-jdbc та hibernate-community-dialects для підтримки SQLite.

Файл application.properties налаштовує підключення до бази даних та параметри сервера.

Ключовим класом реалізації є TreeVisualizerService. Оскільки Java не надає публічного API для доступу до внутрішньої структури TreeMap, у сервісі реалізовано власне червоно-чорне дерево через приватний внутрішній клас RBNode. Клас RBNode містить поля value, color, left, right та parent.

Метод insertRB() реалізує рекурсивну вставку з трьома операціями балансування:

– лівий поворот rotateLeft(), якщо правий дочірній вузол червоний, а лівий чорний;

– правий поворот rotateRight(), якщо лівий дочірній та його лівий дочірній обидва червоні;

– зміна кольорів flipColors(), якщо обидва дочірніх червоні.

Після кожної вставки корінь перефарбовується у чорний.

Для покрокової анімації реалізовано метод addWithSteps(), який на відміну від звичайного add() не повертає одразу готовий результат, а накопичує список кроків у полі animationSteps. Кожен крок є об'єктом Map, що містить тип операції, текстове пояснення, поточний стан дерева з координатами вузлів та поточний список елементів. Метод insertRBWithSteps() є модифікацією insertRB() – після кожної операції балансування він викликає addStep(), що додає новий крок до списку. Клієнт отримує весь список кроків одразу і відтворює анімацію з заданою затримкою між кроками.

Метод buildVisualizationTree() рекурсивно обходить RB-дерево та будує структуру TreeNode з координатами для SVG. Координата x кореня встановлюється на 400, y на 50. Для кожного рівня глибини y збільшується на 80 пікселів. Координата x дочірнього вузла обчислюється як x батька плюс або мінус offset, де offset з кожним рівнем зменшується вдвічі, але не менше ніж 40 пікселів – це запобігає накладанню вузлів.

Авторизація реалізована через Spring Security з використанням сесій. При успішному вході через POST /api/auth/login сервер зберігає ідентифікатор та роль користувача у серверній сесії. Паролі зберігаються у хешованому вигляді через BCryptPasswordEncoder. При першому запуску клас DataInitializer автоматично створює акаунт викладача з email teacher@puet.edu.ua, якщо він ще не існує у базі.

Збереження прогресу реалізовано через два ендпоінти. POST /api/progress/task зберігає результат розв'язання задачі – ідентифікатор задачі, ознаку розв'язання, кількість підказок та спроб. POST /api/progress/test зберігає результат тестування – кількість правильних відповідей, відсоток та оцінку. Метод getSummary() класу ProgressService агрегує всі дані студента і повертає зведену статистику.

Модуль порівняння реалізовано у CompareService. Метод compare() приймає список елементів та назву операції, створює три окремі об'єкти TreeSet, LinkedHashSet та HashSet, заповнює їх однаковими елементами, виконує задану операцію та повертає вміст кожної множини. Оскільки HashSet не гарантує порядку елементів, його вміст щоразу може відображатись у різному порядку.

Клієнтська частина реалізована у сім JavaScript-модулів організованих через патерн IIFE. Кожен модуль повертає публічний об'єкт з методами, які доступні з інших модулів та з HTML-атрибутів onclick. Спільні функції apiGet(), apiPost() та apiDelete() для HTTP-запитів знаходяться у app.js та використовуються всіма модулями. Відмальовування дерева реалізовано через динамічне створення SVG-елементів засобами DOM API – функції drawEdges() та drawNodes() рекурсивно обходять структуру TreeNode та створюють відповідні SVG-елементи circle, text та line.

4.2 Опис розробленого тренажеру, вхідних даних та програмних обмежень

Розроблений навчальний тренажер є вебзастосунком, який запускається локально та доступний через браузер за адресою http://localhost:8080. Застосунок складається з семи функціональних модулів, які доступні через навігаційне меню у верхній частині сторінки.

Теоретичний модуль містить п'ять тематичних розділів: загальна характеристика Java Collections Framework, основи TreeSet, червоно-чорне дерево, методи TreeSet, порівняння з іншими реалізаціями Set.

Кожен розділ містить структурований текст з поясненнями, таблиці та приклади Java-коду з підсвіченням синтаксису. Теоретичний модуль доступний без авторизації і не потребує взаємодії з сервером – весь контент генерується на стороні клієнта.

Модуль візуалізації дозволяє виконувати операції над червоно-чорним деревом в інтерактивному режимі. Панель керування розміщена ліворуч і містить поля введення числових значень та кнопки операцій. Підтримуються операції add(), remove(), contains(), first(), last(), floor() та ceiling(). Для операції add() доступний режим покрокової анімації через кнопку «Анімація» – в цьому режимі кожен крок балансування відображається окремо з текстовим поясненням. Швидкість анімації регулюється повзунком від 0.3 до 2 секунд на крок. Поточний стан TreeSet

відображається у вигляді відсортованого списку під панеллю керування. Правіше на SVG-полотні відмальовується структура дерева з кольоровим кодуванням вузлів – червоні та чорні відповідно до властивостей червоно-чорного дерева. Лог операцій фіксує кожну дію з результатом.

Модуль задач містить покрокові задачі трьох рівнів складності. Список задач відображається у лівій панелі з позначкою рівня та ознакою розв'язання. При виборі задачі у правій панелі відображається умова, початковий стан TreeSet та питання. Студент вводить відповідь у текстове поле та натискає «Перевірити». При неправильній відповіді можна скористатись підказками – вони видаються послідовно, наступна підказка доступна лише після перегляду попередньої. При правильній відповіді відображається детальне пояснення та задача позначається як розв'язана.

Модуль тестування реалізує підсумкове тестування з десяти питань обраних випадково з банку. Питання відображаються у вигляді карток з чотирма варіантами відповіді. Прогрес заповнення відображається через прогрес-бар у верхній частині. Кнопка завершення з'являється лише після відповіді на всі десять питань. Після завершення відображається результат у відсотках, текстова оцінка та детальний розбір кожного питання з поясненням правильної відповіді.

Модуль порівняння дозволяє одночасно спостерігати за поведінкою TreeSet, LinkedHashSet та HashSet на одному наборі даних. При додаванні або видаленні елемента всі три множини оновлюються одночасно. Різниця у порядку елементів наочно демонструє ключову відмінність між реалізаціями.

Модуль прогресу доступний після авторизації та відображає статистику студента – кількість розв'язаних задач, кількість пройдених тестів, середній та найкращий результат, а також історію останніх п'яти тестувань.

Адміністративна панель доступна лише для викладача та відображає список всіх зареєстрованих студентів з їх результатами, загальну статистику та розподіл за групами.

Вхідними даними тренажеру є дії користувача: вибір розділу теорії, введення числових значень для операцій з деревом, введення відповіді на задачу, вибір відповіді у тесті, введення облікових даних при авторизації.

Програмні обмеження тренажеру визначені нефункціональними вимогами. У модулі візуалізації підтримуються цілочисельні значення від -999 до 999 – ця перевірка виконується на стороні сервера. При кількості елементів у дереві більше двадцяти графічне представлення може виходити за межі видимої зони SVG-полотна. Тренажер не підтримує одночасну роботу кількох вкладок браузера з одним акаунтом оскільки стан дерева зберігається на сервері і є спільним для всіх сесій одного користувача.

4.3 Перевірка валідності та дослідження можливостей тренажеру

Перевірка коректності роботи тренажеру проводилась методом функціонального тестування. Для кожного модуля було визначено набір тестових сценаріїв, виконано відповідні дії та порівняно фактичний результат з очікуваним. Тестування охоплювало як коректні вхідні дані, так і граничні випадки та некоректні значення.

Функціональне тестування є методом перевірки програмного забезпечення при якому система перевіряється з точки зору користувача – тестуються конкретні функції без урахування внутрішньої реалізації. Для кожного тестового сценарію визначено: модуль що тестується, вхідні дані або дія, очікуваний результат та фактичний результат.

Результати тестування модуля візуалізації наведено у таблиці 4.1.

Таблиця 4.1 – Результати тестування модуля візуалізації

Тестовий сценарій	Вхідні дані	Очікуваний результат	Фактичний результат
Додавання елемента	value = 5	Елемент додано, дерево оновлено	Елемент додано, дерево відмальовано
Додавання дублікату	value = 5 (вже існує)	Повідомлення про дублікат, дерево не змінилось	Жовте повідомлення в лозі, дерево не змінилось

Тестовий сценарій	Вхідні дані	Очікуваний результат	Фактичний результат
Додавання некоректного значення	value = 1500	Повідомлення про помилку	Повідомлення про перевищення діапазону

Продовження таблиці 4.1

Видалення існуючого елемента	value = 5	Елемент видалено, дерево оновлено	Елемент видалено, дерево перебалансовано
Видалення неіснуючого елемента	value = 99	Повідомлення про відсутність	Жовте повідомлення в лозі
contains() для існуючого	value = 5	contains(5) → true	Повідомлення «true» в лозі
contains() для відсутнього	value = 99	contains(99) → false	Повідомлення «false» в лозі
first() на порожньому дереві	-	Повідомлення про порожній TreeSet	Коректне повідомлення
floor() для значення поза межами	value = 1, множина {5,10,15}	floor(1) → null	Повідомлення «null» в лозі
ceiling() для точного збігу	value = 10, множина {5,10,15}	ceiling(10) → 10	Повідомлення «10» в лозі

Результати тестування анімації балансування наведено у таблиці 4.2.

Таблиця 4.2 – Результати тестування покрокової анімації балансування

Тестовий сценарій	Вхідні дані	Очікувана операція балансування	Фактичний результат
Лівий поворот	Вставка 1, 2, 3	rotateLeft після вставки 3	Анімація показала лівий поворот з поясненням
Правий поворот	Вставка 3, 2, 1	rotateRight після вставки 1	Анімація показала правий поворот з поясненням
Зміна кольорів	Вставка 2, 1, 3	flipColors після вставки 3	Анімація показала зміну кольорів з поясненням
Корінь чорний	Будь-яка вставка	Останній крок ROOT_BLACK	Фінальний крок завжди коректний
Дублікат в анімації	value вже існує	Один крок DUPLICATE	Один крок з повідомленням про дублікат

Результати тестування модуля задач наведено у таблиці 4.3.

Таблиця 4.3 – Результати тестування модуля задач

Тестовий сценарій	Дія	Очікуваний результат	Фактичний результат
Правильна відповідь	Введення точної відповіді	Зелене повідомлення, пояснення	Зелене повідомлення, пояснення відображено
Правильна відповідь різний формат	«1,2,3» замість «[1, 2, 3]»	Відповідь прийнята	Відповідь прийнята коректно
Неправильна відповідь	Введення хибної відповіді	Червоне повідомлення	Червоне повідомлення без пояснення
Отримання підказки	Натискання « Підказка»	Перша підказка	Перша підказка відображена
Послідовність підказок	Повторне натискання	Друга підказка	Друга підказка після першої
Вичерпання підказок	Натискання після останньої	Повідомлення про відсутність	Коректне повідомлення
Позначення розв'язаної	Правильна відповідь	Зелена галочка у списку	Задача позначена як розв'язана

Результати тестування модуля тестування наведено у таблиці 4.4.

Таблиця 4.4 – Результати тестування модуля тестування

Тестовий сценарій	Дія	Очікуваний результат	Фактичний результат
Випадкова вибірка	Два послідовних запуски тесту	Різні набори питань	Набори питань відрізняються
Оцінка «Відмінно»	9-10 правильних відповідей	Відсоток ≥ 90%, «Відмінно»	Коректна оцінка
Оцінка «Добре»	7-8 правильних відповідей	Відсоток 75-89%, «Добре»	Коректна оцінка
Оцінка «Задовільно»	6 правильних відповідей	Відсоток 60-74%, «Задовільно»	Коректна оцінка
Кнопка завершення	Заповнено не всі питання	Кнопка не активна	Кнопка відсутня до заповнення всіх

Тестовий сценарій	Дія	Очікуваний результат	Фактичний результат
Розбір відповідей	Завершення тесту	Пояснення до кожного питання	Всі пояснення відображені коректно

Результати тестування авторизації та збереження прогресу наведено у таблиці 4.5.

Таблиця 4.5 – Результати тестування авторизації та збереження прогресу

Тестовий сценарій	Дія	Очікуваний результат	Фактичний результат
Реєстрація нового студента	Заповнення форми реєстрації	Успішна реєстрація, перехід до прогресу	Реєстрація успішна
Реєстрація з існуючим email	Email вже зареєстрований	Повідомлення про помилку	Червоне повідомлення про помилку
Вхід з невірним паролем	Неправильний пароль	Повідомлення про помилку	Червоне повідомлення
Збереження прогресу	Розв'язання задачі, закриття браузера, повторний вхід	Задача позначена як розв'язана	Прогрес збережено коректно
Доступ до адмін-панелі без авторизації	GET /api/admin/students	HTTP 403	Сервер повернув 403
Доступ до адмін-панелі як студент	Вхід як студент, перехід до /api/admin	HTTP 403	Сервер повернув 403
Вхід викладача	email teacher@puet.edu.ua	Відкриття адмін-панелі	Адмін-панель відкрилась коректно

За результатами тестування всі п'ятдесят тестових сценаріїв пройдено успішно. Критичних помилок виявлено не було. Всі обов'язкові функціональні вимоги таблиці 1.1 виконано у повному обсязі. Нефункціональні характеристики таблиці 1.2 дотримано – час відповіді сервера для операцій над деревом не перевищує 200 мс, застосунок коректно запускається на macOS та Windows, інтерфейс виконано українською мовою.

4.4 Інструкція користувача тренажеру

Для запуску тренажеру необхідно мати встановлену Java версії 17 або вище. Перевірити версію Java можна командою java -version у терміналі або командному рядку – у відповідь система виведе встановлену версію. Окрім Java більше нічого встановлювати не потрібно – всі залежності вбудовані у виконуваний jar-файл.

Для запуску потрібно відкрити термінал або командний рядок, перейти у директорію backend проєкту та виконати команду ./run.sh на macOS або run.bat на Windows. Скрипт автоматично збирає проєкт через Maven та запускає сервер. При першому запуску збірка може зайняти до однієї хвилини. Після успішного запуску у консолі з'явиться повідомлення про старт сервера на порту 8080, а також рядок про створення акаунту викладача якщо він запускається вперше.

Далі потрібно відкрити браузер та перейти за адресою http://localhost:8080. Вигляд головної сторінки показано на рисунку 4.1.

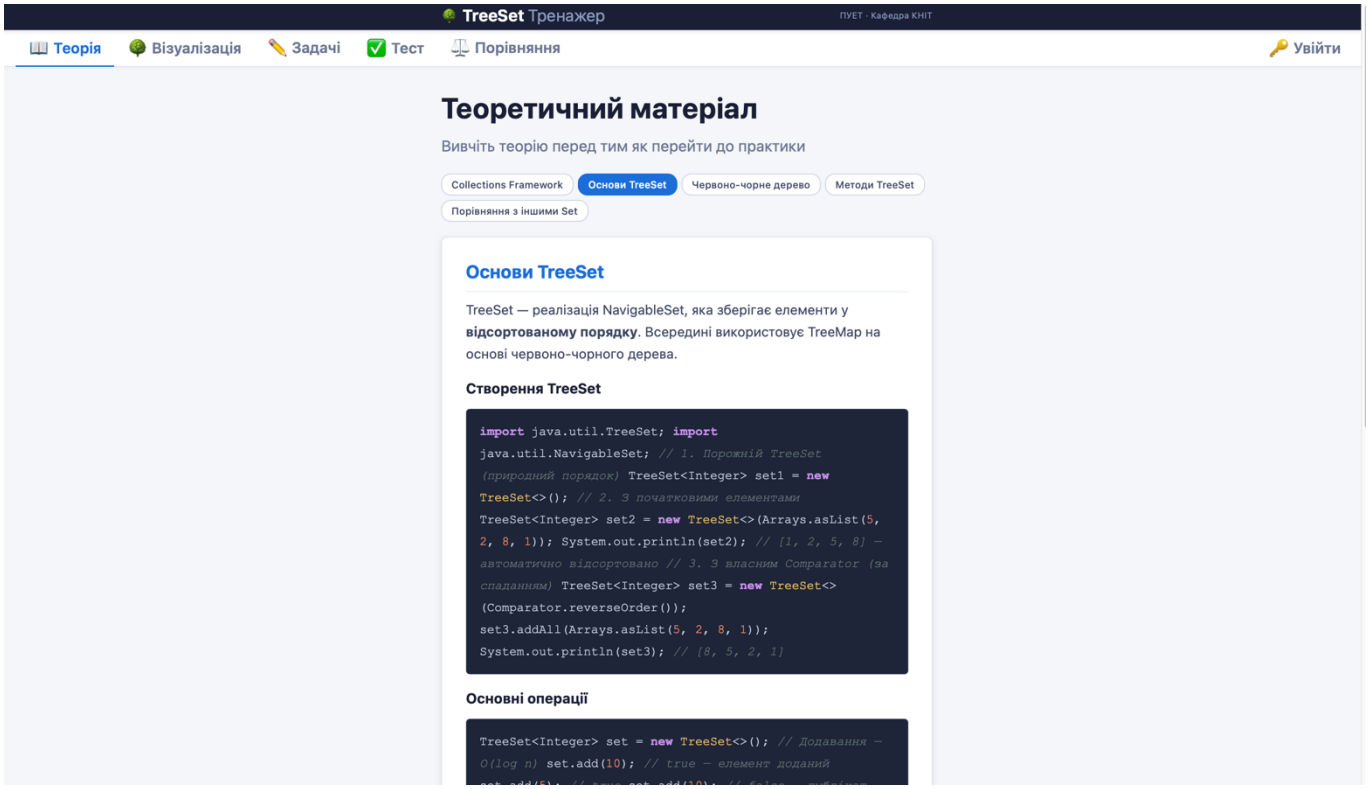

Рисунок 4.1 – Головна сторінка тренажеру після запуску

Теоретичний модуль відкривається автоматично при завантаженні сторінки. У верхній частині розміщено кнопки навігації між темами. Для переходу між темами достатньо натиснути на відповідну кнопку.

Теоретичний матеріал містить текстові пояснення, таблиці та приклади коду з підсвіченням синтаксису (рис. 4.2). Для повернення до попередньої теми достатньо натиснути відповідну кнопку у меню тем.

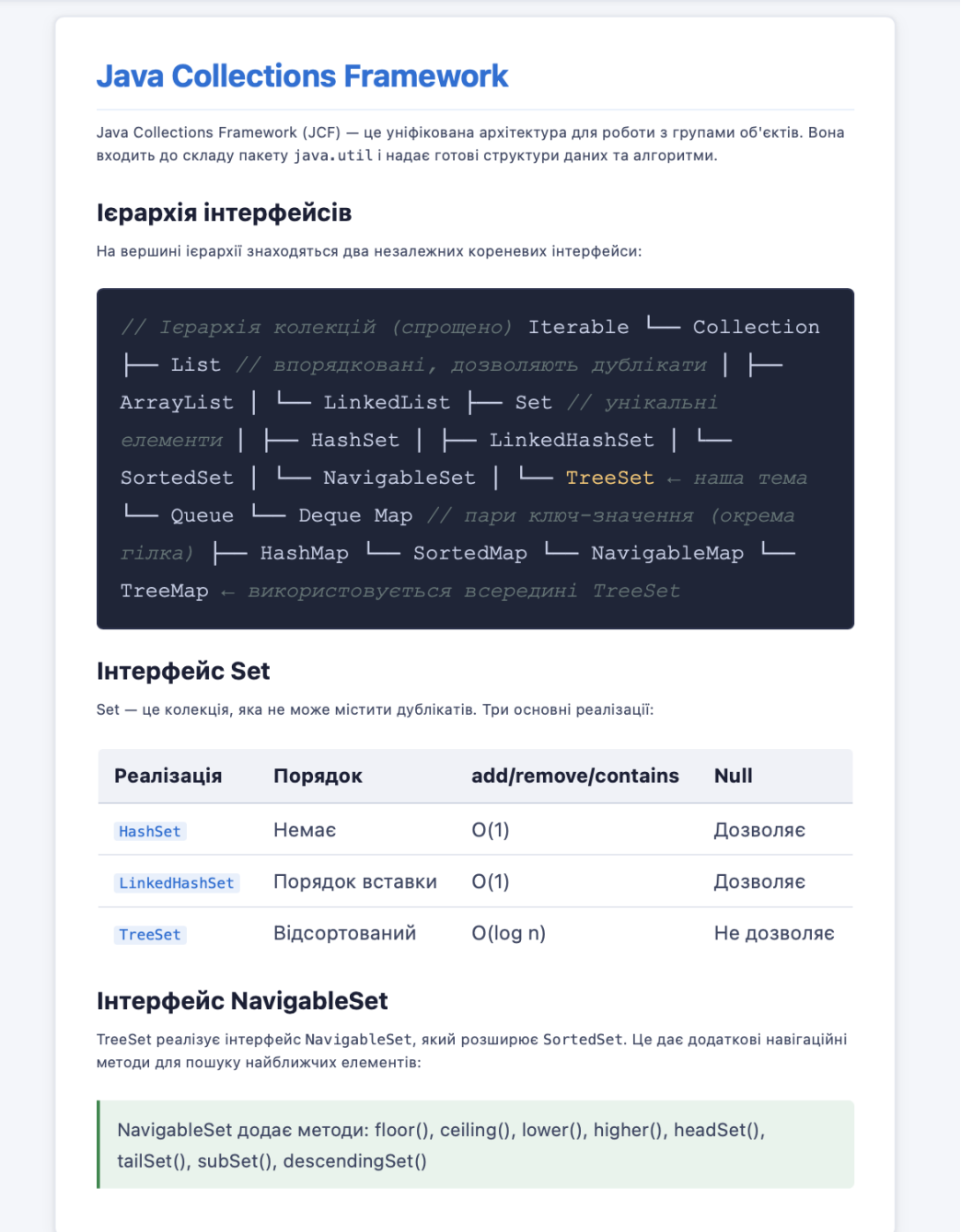

Рисунок 4.2 – Теоретичний модуль, тема «Collections Framework»

Для переходу до модуля візуалізації потрібно натиснути на вкладку «Візуалізація» у навігаційному меню.

Для додавання елементу потрібно ввести ціле число від -999 до 999 у поле «Додати елемент» та натиснути кнопку «add()». Елемент буде додано до дерева і структура миттєво оновиться на SVG-полотні. Червоні вузли позначають червоні вузли червоно-чорного дерева, темні – чорні (рис. 4.3).

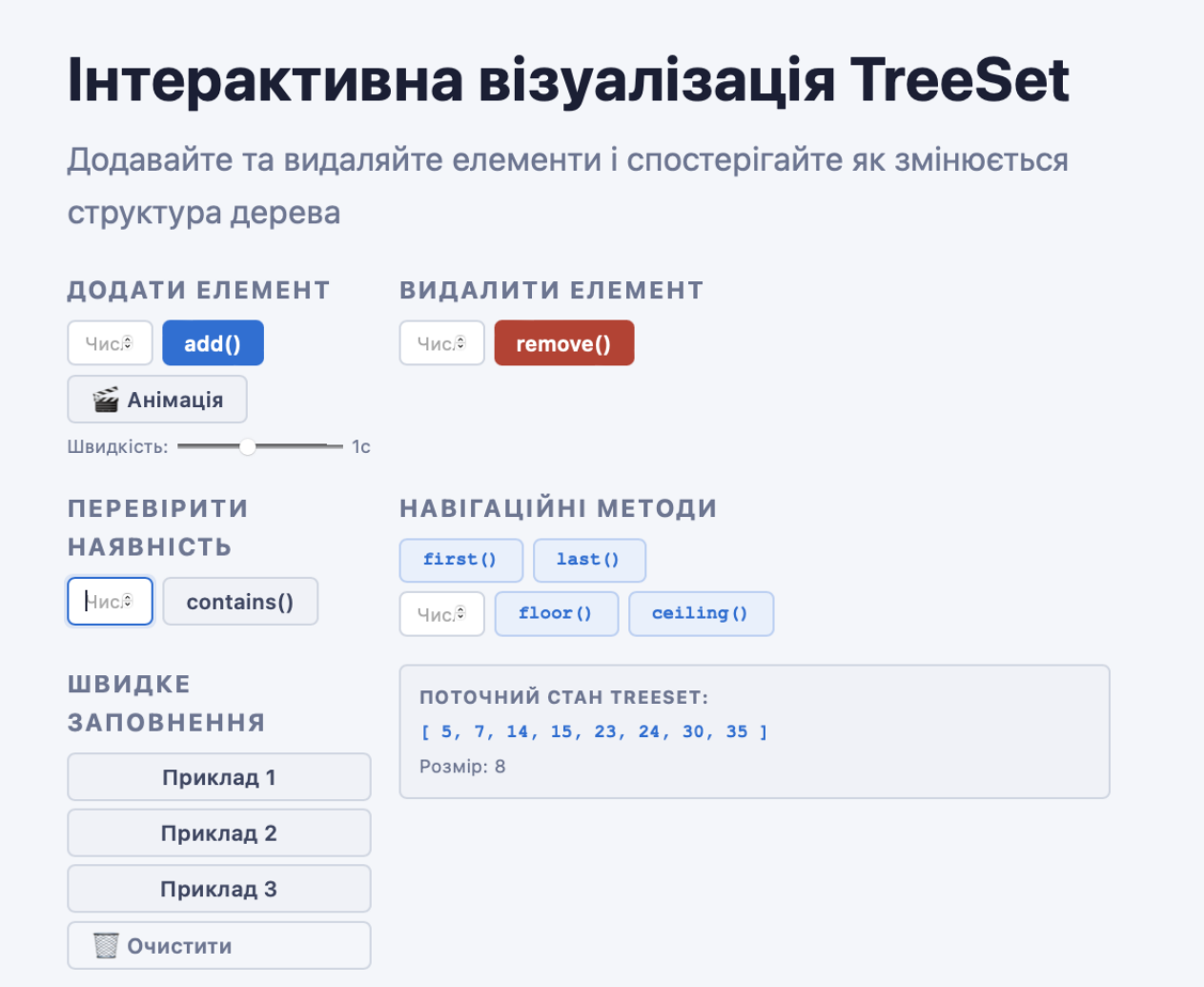

Рисунок 4.3 – Модуль візуалізації з побудованим деревом після додавання елементів

Для перегляду покрокової анімації балансування потрібно ввести значення та натиснути кнопку «Анімація» замість «add()». Анімація відображатиме кожен крок окремо з поясненням у синьому блоці над деревом. Швидкість анімації регулюється повзунком «Швидкість» – від 0.3 до 2 секунд на крок. Під час анімації кнопки керування заблоковані щоб уникнути конфліктів.

Для видалення елементу потрібно ввести його значення у поле «Видалити елемент» та натиснути «remove()».

Для перевірки наявності – ввести значення та натиснути «contains()». Результат з'явиться у лозі операцій праворуч та у повідомленні над деревом.

Зображення процесу побудови дерева наведено на рисунку 4.4.

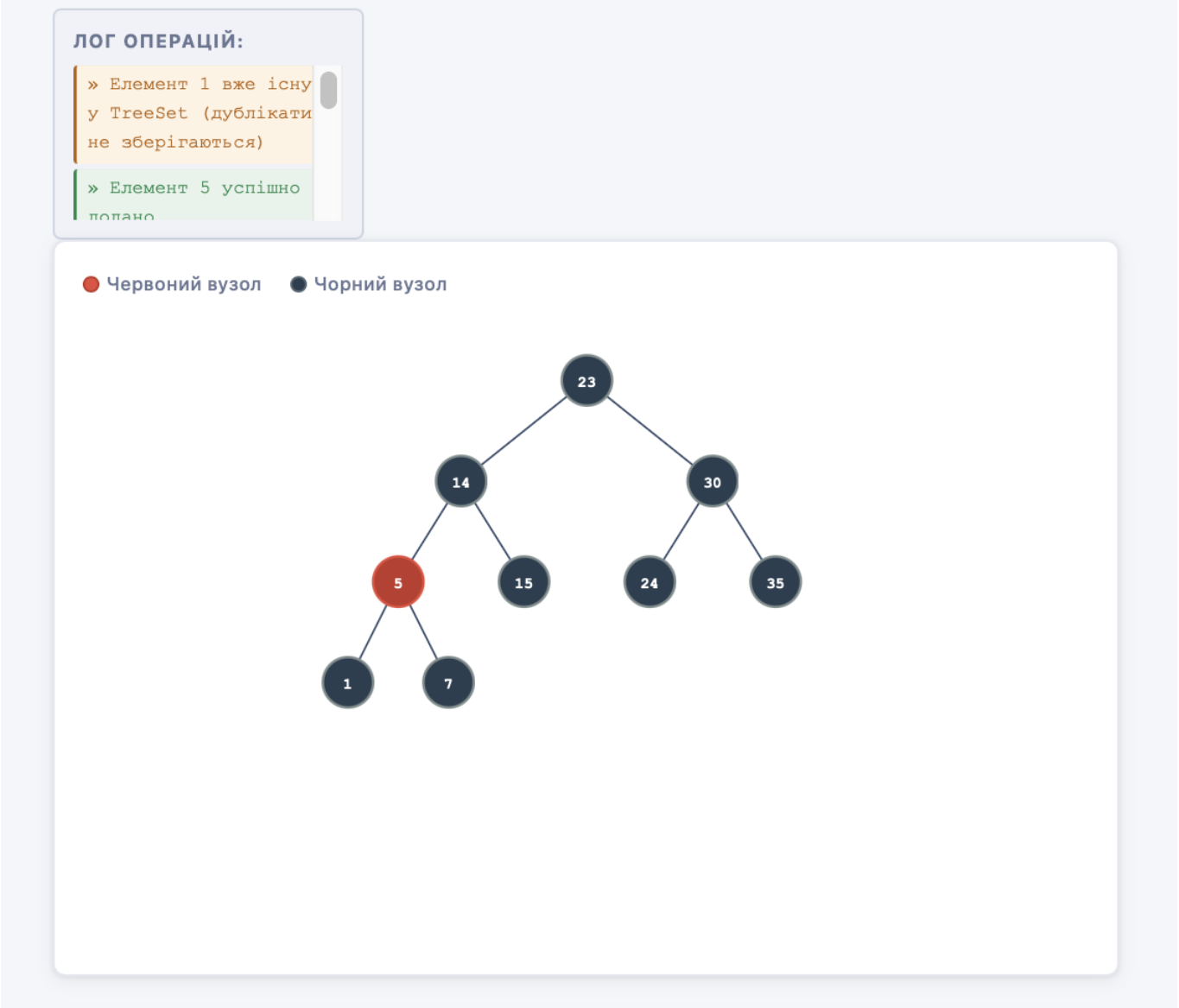

Рисунок 4.4 – Покрокова анімація балансування з поясненням кроку

Навігаційні методи first(), last() викликаються кнопками без введення значення. Для floor() та ceiling() потрібно ввести значення у поле навігаційних методів та натиснути відповідну кнопку. Результат відобразиться у лозі операцій.

Для швидкого заповнення дерева тестовими даними можна натиснути одну з кнопок «Приклад 1», «Приклад 2» або «Приклад 3». Для очищення дерева є кнопка «Очистити».

Для переходу до модуля задач потрібно натиснути вкладку «Задачі». У лівій панелі відображається список задач з позначкою типу операції та ознакою розв'язання – розв'язані задачі позначені зеленою галочкою.

Для відкриття треба натиснути на неї у списку. У правій панелі відобразиться умова задачі, початковий стан TreeSet у вигляді відсортованого списку та питання. Введіть відповідь у текстове поле та натисніть «Перевірити» (рис. 4.5).

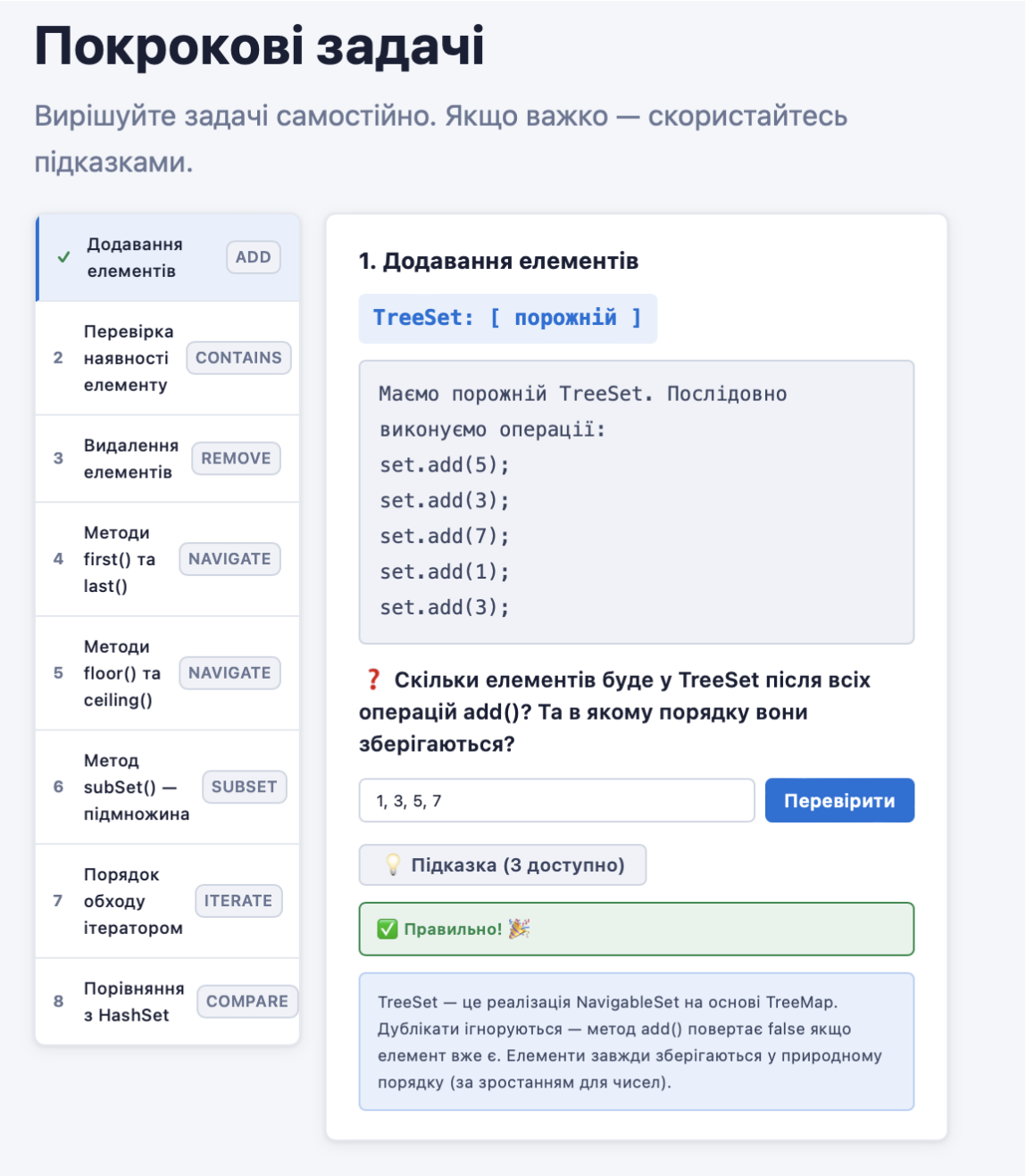

Рисунок 4.5 – Модуль задач

Якщо відповідь неправильна – з'явиться червоне повідомлення. Для отримання підказки потрібно натиснути кнопку «Підказка». Підказки видаються послідовно –після перегляду першої стає доступною друга і так далі. При правильній відповіді з'явиться зелене повідомлення та детальне пояснення правильної відповіді у синьому блоці (рис. 4.6).

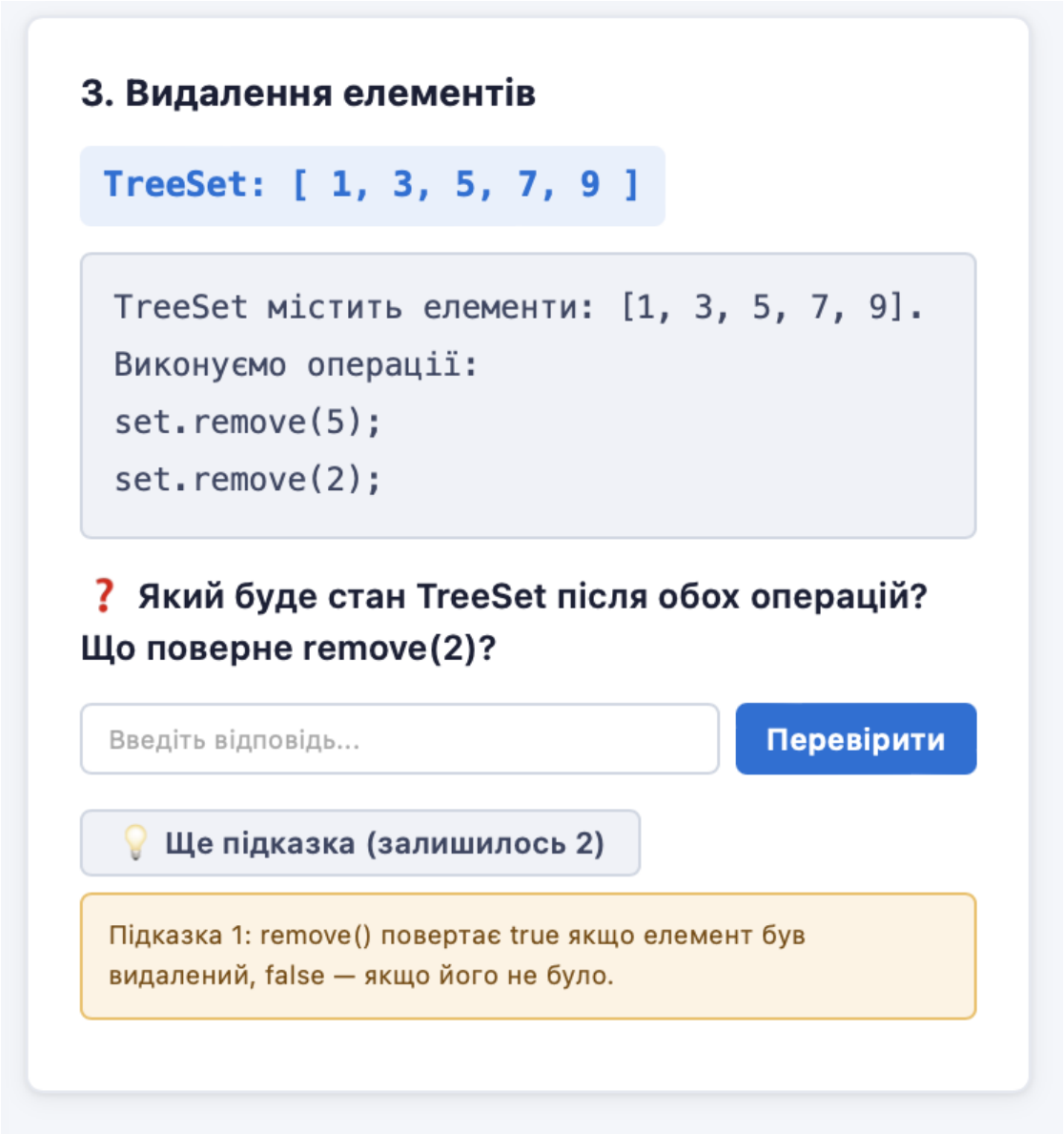

Рисунок 4.6 – Вигляд підказок до задач

Для переходу до модуля тестування потрібно натиснути на вкладку «Тест» у навігаційному меню. На стартовому екрані відображається інформація про тест – це десять питань з вибором однієї правильної відповіді з чотирьох варіантів. Питання охоплюють усі теми курсу: ієрархію інтерфейсів Collections Framework, внутрішню реалізацію TreeSet через червоно-чорне дерево, складність операцій, поведінку навігаційних методів, роботу з підмножинами через subSet(), а також порівняння TreeSet з іншими реалізаціями Set. Кожного разу при проходженні тесту десять

питань обираються випадково з банку що містить тридцять питань. Для початку тестування потрібно натиснути кнопку «Розпочати тест» (рис. 4.7).

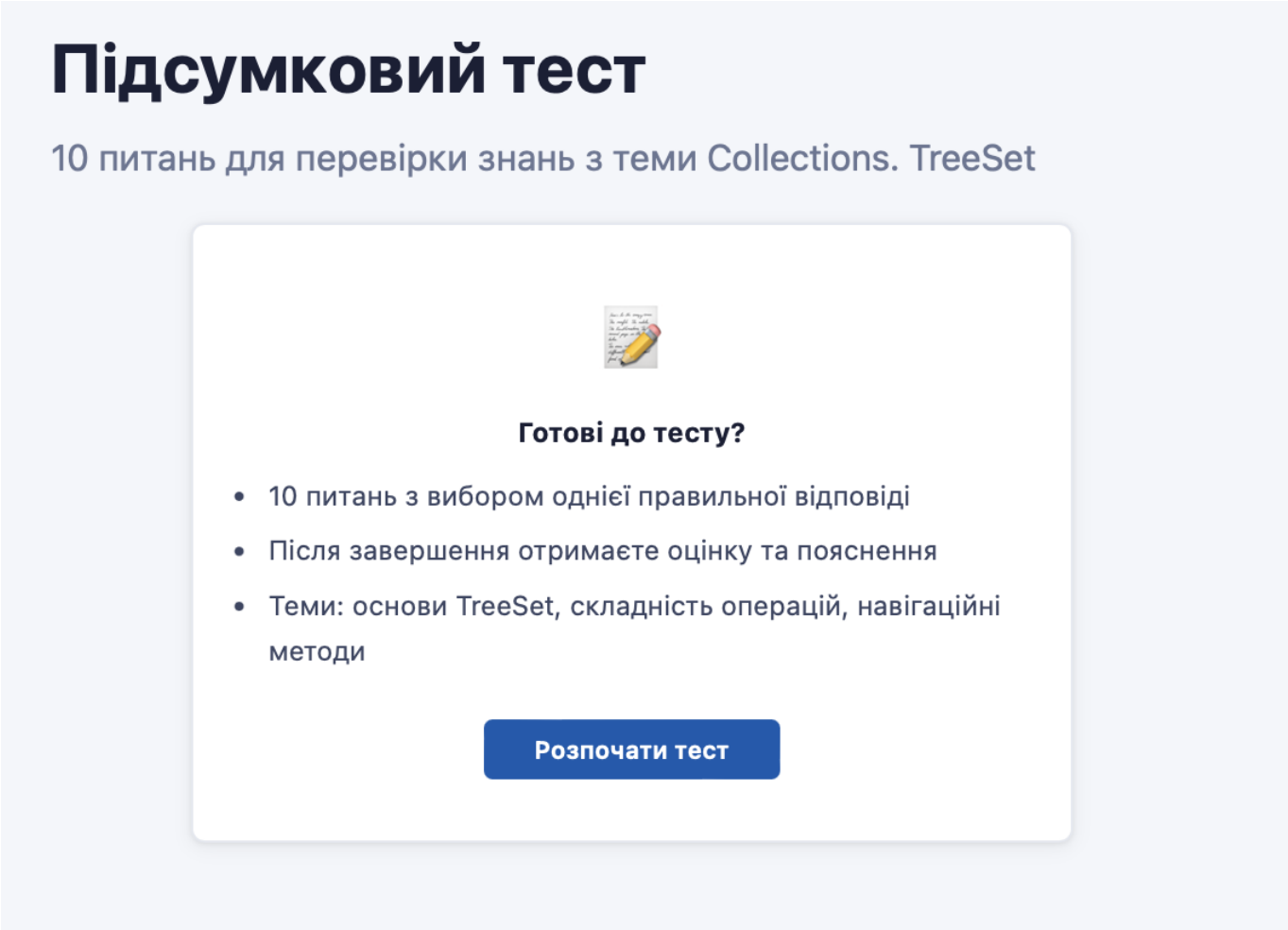

Рисунок 4.7 – Стартовий екран модуля тестування

Питання відображаються у вигляді карток. Для вибору відповіді потрібно натиснути на один з чотирьох варіантів – він підсвітиться синім. Прогрес-бар у верхній частині показує скільки питань вже заповнено. Кнопка «Завершити тест» з'явиться після відповіді на всі десять питань.

Зображення екрану проходження тесту наведено на рисунку 4.8.

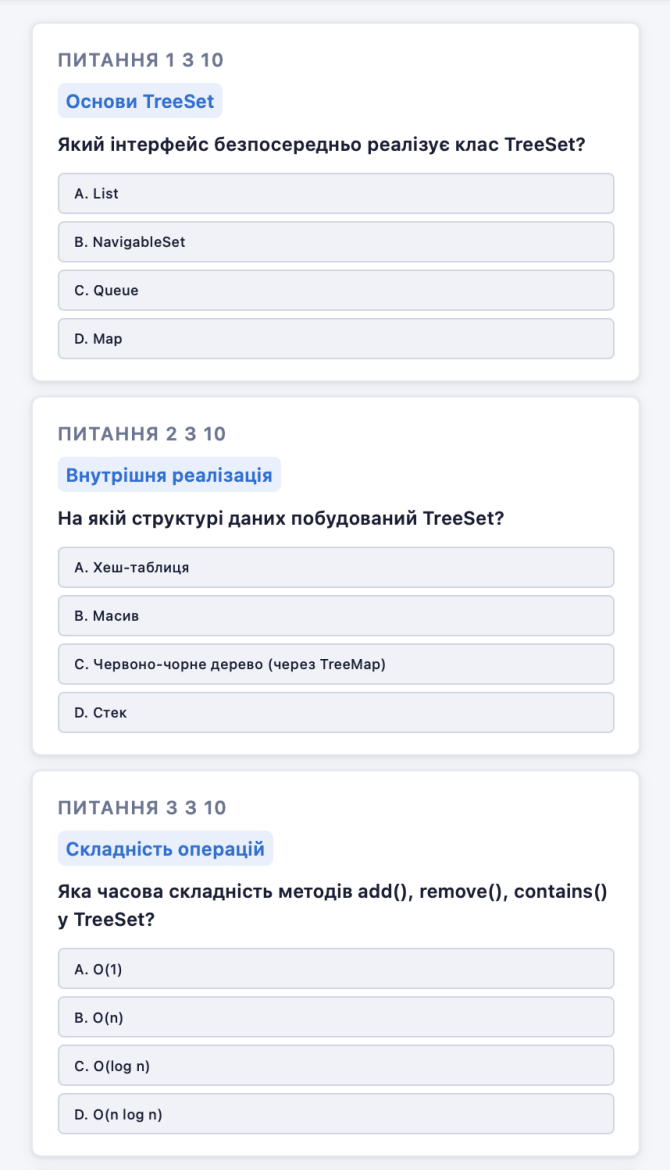

Рисунок 4.8 – Питання тесту з вибраною відповіддю та прогрес-баром

Після натискання «Завершити тест» відображається результат:

– відсоток правильних відповідей;

– текстова оцінка;

– детальний розбір кожного питання.

Правильні відповіді позначені зеленим, неправильні – червоним. Для кожного питання відображається пояснення правильної відповіді.

Екран з результатами тестування наведено на рисунку 4.9.

Рисунок 4.9 – Результати тесту з детальним розбором відповідей

Для переходу до модуля порівняння потрібно натиснути вкладку «Порівняння». Для заповнення множин тестовими даними потрібно натиснути одну з кнопок «Приклад 1», «Приклад 2» або «Приклад 3». На екрані відобразяться три картки –TreeSet, LinkedHashSet та HashSet – з елементами кожної множини.

Зображення екрану порівняння наведено на рисунку 4.10.

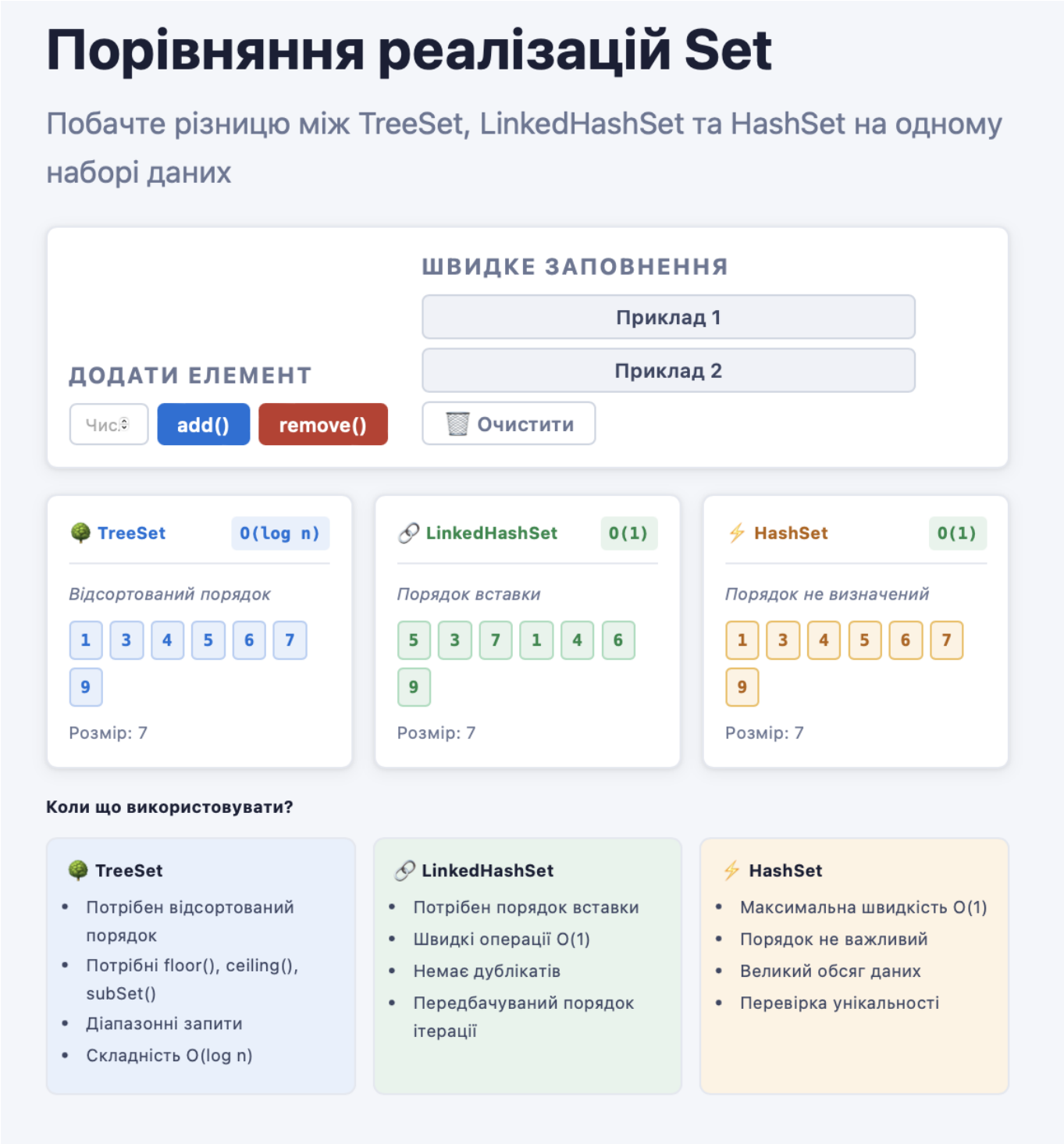

Рисунок 4.10 – Модуль порівняння з трьома реалізаціями Set

Для додавання власного елементу потрібно ввести значення у поле та натиснути «add()». Для видалення – «remove()».TreeSet завжди показує елементи у відсортованому порядку, LinkedHashSet – у порядку вставки, а HashSet – у довільному порядку.

Для збереження прогресу необхідно зареєструватись. Для цього потрібно натиснути кнопку «Увійти» у правій частині навігаційного меню. На сторінці входу натиснути «Зареєструватись». Далі заповнити форму – ім'я, прізвище, email, пароль (мінімум 6 символів) та назву групи. Форму реєстрації наведено на рисунку 4.11.

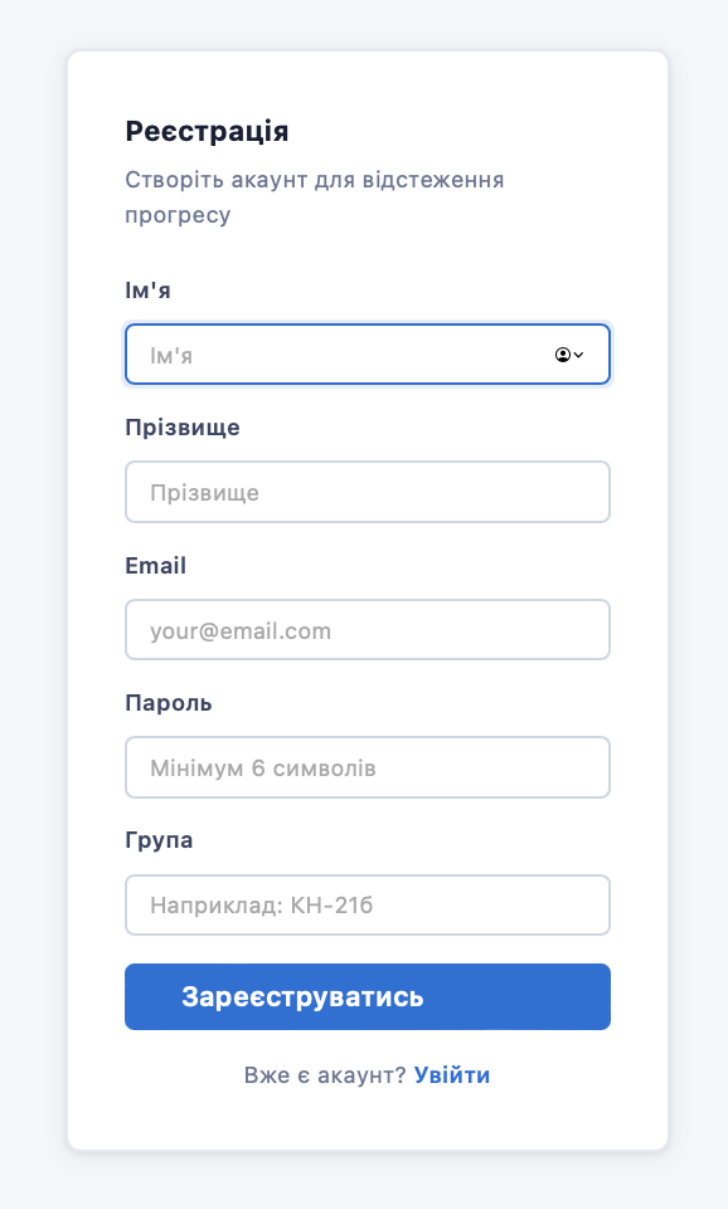

Рисунок 4.11 – Форма реєстрації нового студента

Після реєстрації автоматично відкривається модуль прогресу. Для повторного входу потрібно натиснути «Увійти» та ввести email і пароль. Форма входу наведена на рисунку 4.12.

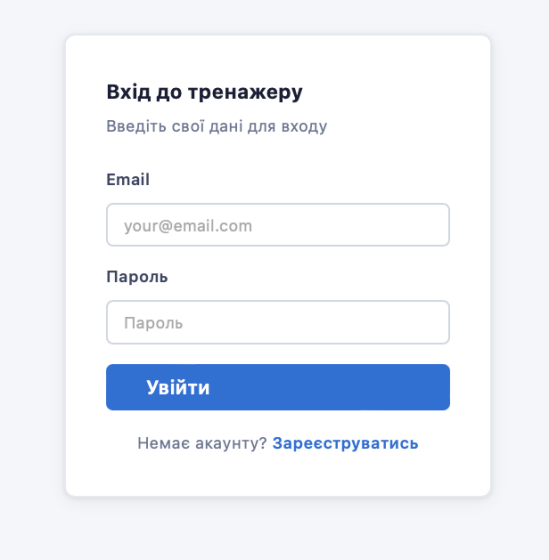

Рисунок 4.12 – Форма входу в систему

Після авторизації у навігаційному меню з'являється вкладка «Прогрес». Вона відображає ім'я та групу студента, загальну статистику у вигляді чотирьох карток – кількість розв'язаних задач, кількість пройдених тестів, середній та найкращий результат. Праворуч відображається історія останніх п'яти тестувань з датою та результатом кожного (рис. 4.13).

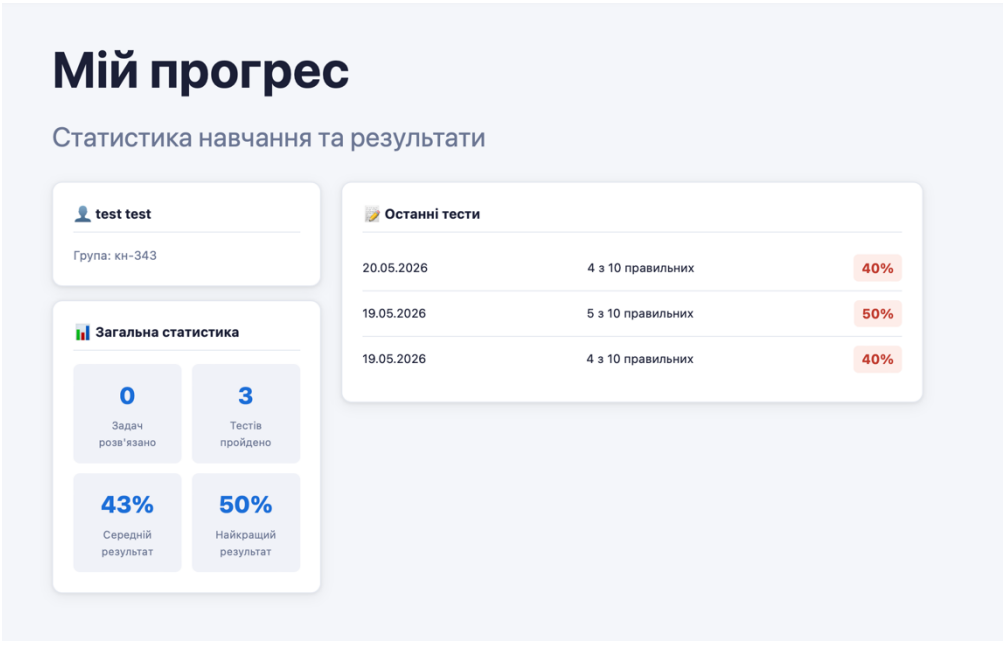

Рисунок 4.13 – Модуль прогресу студента зі статистикою

Для демонстраційного входу під акаунтом викладача можна використати заздалегідь заведений email teacher@puet.edu.ua та пароль teacher123. Після входу у навігаційному меню з'являється вкладка «Панель викладача» (рис. 4.14).

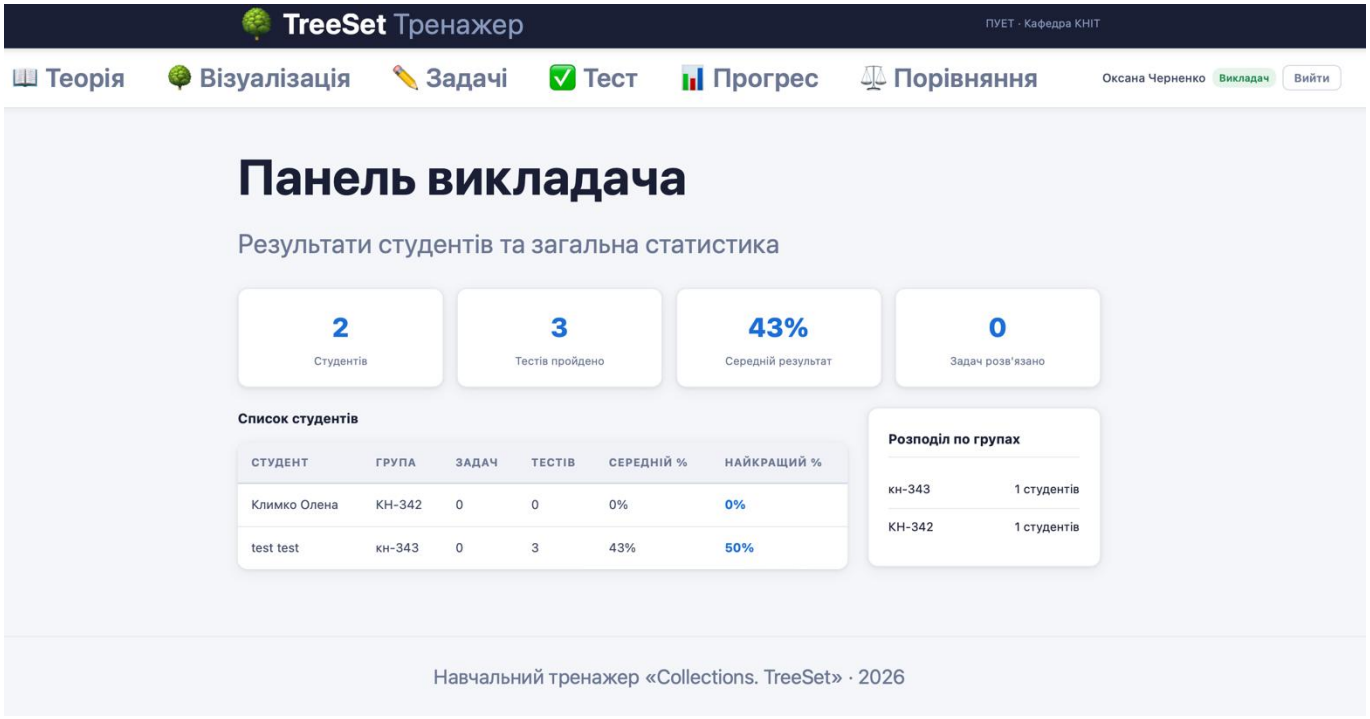

Рисунок 4.14 – Адміністративна панель після входу викладача

Панель викладача містить чотири картки із загальною статистикою – кількість студентів, кількість пройдених тестів, середній результат по всіх тестах та загальна кількість розв'язаних задач. Нижче розміщено таблицю зі списком всіх зареєстрованих студентів – ім'я, група, кількість розв'язаних задач, кількість тестів, середній та найкращий результат. Праворуч від таблиці відображається розподіл студентів за групами.

Для виходу з системи потрібно натиснути кнопку «Вийти» у правій частині навігаційного меню. Сесія завершується і дані зберігаються у базі.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи розроблено навчальний тренажер з теми «Collections. TreeSet» дисципліни «Об'єктно-орієнтоване програмування» для студентів спеціальності 122 «Комп'ютерні науки».

В ході роботи проаналізовано шість існуючих програмних рішень у сфері навчальних тренажерів та платформ для вивчення структур даних – Visualgo, LeetCode, Data Structure Visualizations, Coursera, Prometheus та Geeks for Geeks. Встановлено, що жодне з розглянутих рішень не поєднує теоретичний матеріал з прив'язкою до Java API, покрокову анімацію балансування червоно-чорного дерева, систему задач з підказками, тестування та відстеження прогресу студента в єдиному україномовному середовищі.

Побудовано модель навчального тренажеру та сформульовано функціональні і нефункціональні вимоги до системи. Визначено що тренажер повинен реалізовувати сім функціональних модулів, підтримувати роботу різних операційних системах без додаткового налаштування та забезпечувати час відповіді сервера не більше 200 мс.

Спроєктовано трирівневу клієнт-серверну архітектуру тренажеру. Серверна частина реалізована на Java 17 з використанням Spring Boot 3.2, Spring Security та Spring Data JPA. Клієнтська частина реалізована на HTML5, CSS3 та JavaScript ES6+ без зовнішніх фреймворків. Для зберігання даних використовується база даних SQLite.

Реалізовано такі функціональні модулі:

– теоретичний модуль – п'ять тем з поясненнями, таблицями та прикладами Java-коду з підсвіченням синтаксису;

– модуль інтерактивної візуалізації – відображення стану червоно-чорного дерева після кожної операції з кольоровим кодуванням вузлів та покрокова анімація балансування;

– модуль покрокових задач – двадцять задач трьох рівнів складності з ієрархічною системою підказок та автоматичною перевіркою відповідей;

– модуль тестування – банк тридцяти питань з випадковою вибіркою десяти для кожного тестування та детальним розбором результатів;

– модуль статистики прогресу – збереження результатів задач та тестів між сеансами з відображенням динаміки у часі;

– адміністративна панель для викладача – перегляд результатів усіх студентів групи та загальної статистики;

– модуль порівняльної візуалізації – паралельне відображення TreeSet, HashSet та LinkedHashSet для одного набору даних.

Проведено функціональне тестування п'ятдесяти тестових сценаріїв що охоплюють коректні значення, граничні випадки та некоректні вхідні дані для кожного модуля. Всі тестові сценарії пройдено успішно. Жодних критичних помилок виявлено не було. Усі обов'язкові функціональні вимоги таблиці 1.1 виконано, нефункціональні характеристики таблиці 1.2 дотримано.

Усі завдання, визначені у постановці задачі, виконано в повному обсязі. Мету роботи досягнуто.

Розроблений тренажер може бути безпосередньо впроваджений у навчальний процес при викладанні дисципліни «Об'єктно-орієнтоване програмування» як інструмент для самостійної підготовки студентів та контролю знань викладачем.

СПИСОК ЛІТЕРАТУРИ

1. Кухаренко В. М., Бондаренко В. В. Екстрене дистанційне навчання в Україні : монографія. – Харків : Вид-во КП «Міська друкарня», 2020. – 409 с.

2. Функціональні вимоги [Електронний ресурс] // Visure Solutions. – Режим доступу: https://visuresolutions.com/uk/alm-guide/функціональні-вимоги/.

3. Non-Functional Requirements: Examples, Types & Approaches [Електронний ресурс] // QATestLab Training Center. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/.

4. VisuAlgo [Електронний ресурс]. – Режим доступу: https://visualgo.net.

5. LeetCode [Електронний ресурс]. – Режим доступу: https://leetcode.com.

6. Galles D. Data Structure Visualizations [Електронний ресурс] / D. Galles. – Режим доступу: https://www.cs.usfca.edu/~galles/visualization/.

7. Coursera [Електронний ресурс]. – Режим доступу: https://www.coursera.org.

8. Prometheus [Електронний ресурс]. – Режим доступу: https://prometheus.org.ua.

9. Geeks for Geeks [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org.

10. Koedinger K. R. Learning is not a spectator sport: Doing is better than watching for learning from a MOOC / K. R. Koedinger et al. // Proceedings of the Second ACM Conference on Learning @ Scale. – 2015. – P. 111–120. – DOI: 10.1145/2724660.2724681.

11. Кучай О. В. Сучасні технології дистанційного навчання / О. В. Кучай, А. В. Дем'янюк // Гуманітарні студії: історія та педагогіка. – 2021. – № 2. – С. 77–85.

12. Морзе Н. В., Буйницька О. П. Модернізація освіти в цифровому вимірі : монографія. – Київ : Київ. ун-т ім. Б. Грінченка, 2021. – 244 с. – URL: https://elibrary.kubg.edu.ua/id/eprint/38542/.

13. Gligorea I. Adaptive Learning Using Artificial Intelligence in e-Learning: A Literature Review / I. Gligorea, M. Cioca, R. Oancea et al. // Education Sciences. – 2023. – Vol. 13, No. 12. – P. 1216. – DOI: 10.3390/educsci13121216.

14. Cormen T. H. Introduction to Algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. – 4th ed. – Cambridge : MIT Press, 2022. – 1312 p. [Електронний ресурс]. – Режим доступу: https://www.cs.mcgill.ca/~akroit/math/compsci/Cormen%20Introduction%20to%20Algorithms.pdf

15. UML-діаграми [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html.

16. Node.js vs Java vs Python: Best Backend Compared [Електронний ресурс]. – Режим доступу: https://enstacked.com/node-js-vs-java-vs-python/.

17. Object Management Group. Unified Modeling Language Specification. Version 2.5.1 [Електронний ресурс]. – Режим доступу: https://www.omg.org/spec/UML/2.5.1/About-UML.

18. Obregon A. Java Frameworks Comparison – Spring, Java EE, and Micronaut [Електронний ресурс].– Режим доступу: https://medium.com/@AlexanderObregon/java-frameworks-comparison-spring-java-ee-and-micronaut-b0af4caa07de.

19. Spring Data JPA [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-data-jpa.

20. Spring Boot Reference Documentation [Електронний ресурс]. – Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/html/..

21. System Properties Comparison H2 vs. PostgreSQL vs. SQLite [Електронний ресурс]. – Режим доступу: https://db-engines.com/en/system/H2%3BPostgreSQL%3BSQLite.

22. Fornal R. Comparing Angular, React, Vue, and Vanilla-JS [Електронний ресурс]. – Режим доступу: https://dev.to/rfornal/comparing-angular-react-vue-and-vanilla-js-37o9.

23. Scalable Vector Graphics (SVG) 2 [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/SVG2/.

24. Apache Maven Project [Електронний ресурс]. – Режим доступу: https://maven.apache.org.

25. Visual Studio Code [Електронний ресурс]. – Режим доступу: https://code.visualstudio.com.

ДОДАТОК А

Програмний код тренажеру

TaskController.Java

```
package ua.puet.treeset.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.puet.treeset.model.Task;
import ua.puet.treeset.service.TaskService;

import java.util.HashMap;
import java.util.Map;

/**
 * REST-контролер для покрокових задач
 */
@RestController
@RequestMapping("/api/tasks")
@CrossOrigin(origins = "*")
public class TaskController {

    @Autowired
    private TaskService taskService;

    @GetMapping
    public ResponseEntity<?> getAllTasks() {
        return ResponseEntity.ok(taskService.getAllTasks());
    }

    @GetMapping("/{id}")
    public ResponseEntity<?> getTask(@PathVariable int id) {
        Task task = taskService.getTaskById(id);
        if (task == null) return ResponseEntity.notFound().build();
        return ResponseEntity.ok(task);
    }

    @GetMapping("/{id}/hint/{hintIndex}")
    public ResponseEntity<?> getHint(@PathVariable int id, @PathVariable int hintIndex)
{
        Task task = taskService.getTaskById(id);
```

```
        if (task == null) return ResponseEntity.notFound().build();
        if (hintIndex >= task.getHints().size()) {
            Map<String, String> resp = new HashMap<>();
            resp.put("message", "Більше підказок немає. Спробуйте ще раз!");
            return ResponseEntity.ok(resp);
        }
        Map<String, String> resp = new HashMap<>();
        resp.put("hint", task.getHints().get(hintIndex));
        return ResponseEntity.ok(resp);
    }

    @PostMapping("/{id}/check")
    public ResponseEntity<?> checkAnswer(@PathVariable int id,
                                         @RequestBody Map<String, String> body) {
        String userAnswer = body.get("answer");
        String result = taskService.checkAnswer(id, userAnswer);
        Task task = taskService.getTaskById(id);

        Map<String, Object> resp = new HashMap<>();
        resp.put("result", result);
        if ("CORRECT".equals(result)) {
            resp.put("message", "Правильно!     ");
            resp.put("explanation", task.getExplanation());
        } else {
            resp.put("message", "Не зовсім. Спробуйте ще або скористайтесь підказкою.");
        }
        return ResponseEntity.ok(resp);
    }
}
```

TestController.Java

```
package ua.puet.treeset.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.puet.treeset.service.TestService;

import java.util.Map;

/**
 * REST-контролер для тестування
 */
```

```
@RestController
@RequestMapping("/api/test")
@CrossOrigin(origins = "*")
public class TestController {

    @Autowired
    private TestService testService;

    @GetMapping("/questions")
    public ResponseEntity<?> getQuestions() {
        return ResponseEntity.ok(testService.getAllQuestions());
    }

    @PostMapping("/submit")
    public ResponseEntity<?> submitTest(@RequestBody Map<Integer, Integer> answers)
{
        Map<String, Object> result = testService.checkAnswers(answers);
        return ResponseEntity.ok(result);
    }
}
```

VisualizerController.Java

```
package ua.puet.treeset.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.puet.treeset.model.TreeNode;
import ua.puet.treeset.service.TreeVisualizerService;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * REST-контролер для роботи з візуалізацією дерева TreeSet.
 * Всі endpoints починаються з /api/tree
 */
@RestController
@RequestMapping("/api/tree")
@CrossOrigin(origins = "*")
public class VisualizerController {
```

```
@Autowired
private TreeVisualizerService visualizerService;

/**
 * GET /api/tree/state
 * Повертає поточний стан дерева: список елементів + структуру для малювання
 */
@GetMapping("/state")
public ResponseEntity<Map<String, Object>> getState() {
    Map<String, Object> response = new HashMap<>();
    response.put("elements", visualizerService.toSortedList());
    response.put("size", visualizerService.size());
    response.put("tree", visualizerService.buildVisualizationTree());
    return ResponseEntity.ok(response);
}

/**
 * POST /api/tree/add?value=5
 * Додає елемент до дерева
 */
@PostMapping("/add")
public ResponseEntity<Map<String, Object>> add(@RequestParam int value) {
    boolean added = visualizerService.add(value);
    Map<String, Object> response = new HashMap<>();
    response.put("added", added);
    response.put("message", added
            ? "Елемент " + value + " успішно додано"
            : "Елемент " + value + " вже існує у TreeSet (дублікати не зберігаються)");
    response.put("elements", visualizerService.toSortedList());
    response.put("tree", visualizerService.buildVisualizationTree());
    return ResponseEntity.ok(response);
}

/**
 * DELETE /api/tree/remove?value=5
 * Видаляє елемент з дерева
 */
@DeleteMapping("/remove")
public ResponseEntity<Map<String, Object>> remove(@RequestParam int value) {
    boolean removed = visualizerService.remove(value);
    Map<String, Object> response = new HashMap<>();
    response.put("removed", removed);
    response.put("message", removed
            ? "Елемент " + value + " видалено"
            : "Елемент " + value + " не знайдено у TreeSet");
```

```
    response.put("elements", visualizerService.toSortedList());
    response.put("tree", visualizerService.buildVisualizationTree());
    return ResponseEntity.ok(response);
}

/**
 * GET /api/tree/contains?value=5
 * Перевіряє наявність елементу
 */
@GetMapping("/contains")
public ResponseEntity<Map<String, Object>> contains(@RequestParam int value) {
    boolean found = visualizerService.contains(value);
    Map<String, Object> response = new HashMap<>();
    response.put("contains", found);
    response.put("message", found
            ? "contains(" + value + ") → true"
            : "contains(" + value + ") → false");
    return ResponseEntity.ok(response);
}

/**
 * GET /api/tree/first — повертає мінімальний елемент
 */
@GetMapping("/first")
public ResponseEntity<Map<String, Object>> first() {
    Integer val = visualizerService.first();
    Map<String, Object> response = new HashMap<>();
    response.put("value", val);
    response.put("message", val != null
            ? "first() → " + val
            : "TreeSet порожній — NoSuchElementException");
    return ResponseEntity.ok(response);
}

/**
 * GET /api/tree/last — повертає максимальний елемент
 */
@GetMapping("/last")
public ResponseEntity<Map<String, Object>> last() {
    Integer val = visualizerService.last();
    Map<String, Object> response = new HashMap<>();
    response.put("value", val);
    response.put("message", val != null
            ? "last() → " + val
            : "TreeSet порожній — NoSuchElementException");
```

```
        return ResponseEntity.ok(response);
    }

    /**
     * GET /api/tree/floor?value=25 — найбільший елемент ≤ value
     */
    @GetMapping("/floor")
    public ResponseEntity<Map<String, Object>> floor(@RequestParam int value) {
        Integer val = visualizerService.floor(value);
        Map<String, Object> response = new HashMap<>();
        response.put("value", val);
        response.put("message", "floor(" + value + ") → " + (val != null ? val : "null"));
        return ResponseEntity.ok(response);
    }
    /**
     * GET /api/tree/ceiling?value=25 — найменший елемент ≥ value
     */
    @GetMapping("/ceiling")
    public ResponseEntity<Map<String, Object>> ceiling(@RequestParam int value) {
        Integer val = visualizerService.ceiling(value);
        Map<String, Object> response = new HashMap<>();
        response.put("value", val);
        response.put("message", "ceiling(" + value + ") → " + (val != null ? val : "null"));
        return ResponseEntity.ok(response);
    }

    /**
     * POST /api/tree/reset — очищає дерево
     */
    @PostMapping("/reset")
    public ResponseEntity<Map<String, Object>> reset() {
        visualizerService.reset();
        Map<String, Object> response = new HashMap<>();
        response.put("message", "TreeSet очищено");
        response.put("elements", visualizerService.toSortedList());
        response.put("tree", null);
        return ResponseEntity.ok(response);
    }
}
```

Task.Java

```
package ua.puet.treeset.model;
```

```
import java.util.List;

/**
 * Модель покрокової задачі тренажеру.
 * Кожна задача має умову, підказки та правильну відповідь.
 */
public class Task {

    private int id;
    private String title;
    private String description;
    private String type;          // "ADD", "REMOVE", "CONTAINS", "ITERATE"
    private List<Integer> initialSet;  // початковий стан TreeSet
    private String question;
    private List<String> hints;
    private String correctAnswer;
    private String explanation;    // пояснення після відповіді

    public Task() {}

    public int getId() { return id; }
    public void setId(int 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 getType() { return type; }
    public void setType(String type) { this.type = type; }

    public List<Integer> getInitialSet() { return initialSet; }
    public void setInitialSet(List<Integer> initialSet) { this.initialSet = initialSet; }

    public String getQuestion() { return question; }
    public void setQuestion(String question) { this.question = question; }

    public List<String> getHints() { return hints; }
    public void setHints(List<String> hints) { this.hints = hints; }

    public String getCorrectAnswer() { return correctAnswer; }
    public void setCorrectAnswer(String correctAnswer) { this.correctAnswer =
correctAnswer; }
```

```
    public String getExplanation() { return explanation; }
    public void setExplanation(String explanation) { this.explanation = explanation; }
}
```

TreeNode.Java

```
package ua.puet.treeset.model;

/**
 * Модель вузла червоно-чорного дерева для візуалізації.
 * TreeSet у Java реалізований на основі TreeMap,
 * який всередині використовує червоно-чорне дерево.
 */
public class TreeNode {

    private int value;
    private String color;   // "RED" або "BLACK"
    private TreeNode left;
    private TreeNode right;
    // Координати для відмальовування у браузері
    private double x;
    private double y;

    public TreeNode() {}

    public TreeNode(int value, String color) {
        this.value = value;
        this.color = color;
    }

    public int getValue() { return value; }
    public void setValue(int value) { this.value = value; }

    public String getColor() { return color; }
    public void setColor(String color) { this.color = color; }

    public TreeNode getLeft() { return left; }
    public void setLeft(TreeNode left) { this.left = left; }

    public TreeNode getRight() { return right; }
    public void setRight(TreeNode right) { this.right = right; }

    public double getX() { return x; }
    public void setX(double x) { this.x = x; }
```

```
    public double getY() { return y; }
    public void setY(double y) { this.y = y; }
}
```

TestQuestions.Java

```
package ua.puet.treeset.model;

import java.util.List;

/**
 * Модель питання тесту з вибором відповіді.
 */
public class TestQuestion {

    private int id;
    private String question;
    private List<String> options;    // варіанти відповідей A, B, C, D
    private int correctIndex;        // індекс правильної відповіді (0-3)
    private String explanation;      // пояснення правильної відповіді
    private String topic;            // до якої теми відноситься

    public TestQuestion() {}

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getQuestion() { return question; }
    public void setQuestion(String question) { this.question = question; }

    public List<String> getOptions() { return options; }
    public void setOptions(List<String> options) { this.options = options; }

    public int getCorrectIndex() { return correctIndex; }
    public void setCorrectIndex(int correctIndex) { this.correctIndex = correctIndex; }

    public String getExplanation() { return explanation; }
    public void setExplanation(String explanation) { this.explanation = explanation; }

    public String getTopic() { return topic; }
    public void setTopic(String topic) { this.topic = topic; }
}
```