Міністерство освіти і науки України
Тернопільський національний педагогічний університет
імені Володимира Гнатюка

Фізико-математичний факультет
Кафедра інформатики та методики її навчання

Кваліфікаційна робота

МЕТОДИЧНІ АСПЕКТИ НАВЧАННЯ УЧНІВ ОСНОВАМ ВЕБРОЗРОБКИ ІЗ ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ REACT

Спеціальність 014 Середня освіта

Освітня програма

«Середня освіта (Інформатика, математика, STEM-освіта)»

Здобувача другого (магістерського)
рівня вищої освіти
Твердохліба Юрія Петровича

НАУКОВИЙ КЕРІВНИК:
кандидат біологічних наук, доцент
Шмигер Галина Петрівна

РЕЦЕНЗЕНТ:
професор кафедри комп'ютерних
технологій Тернопільського
національного педагогічного
університету ім. В. Гнатюка,
доктор педагогічних наук
Потапчук Ольга Ігорівна

Тернопіль – 2025

АНОТАЦІЯ

Твердохліб Ю. П. Методичні аспекти навчання учнів основам веброзробки із використанням технології React. Кваліфікаційна робота на здобуття освітнього ступеня «магістр» зі спеціальності Середня освіта (Інформатика). ТНПУ ім. В. Гнатюка. Тернопіль, 2025. 74 с.

У кваліфікаційній роботі розглянуто проблему підвищення ефективності навчання інформатики засобами сучасних вебтехнологій та обґрунтовано доцільність використання інтерактивних вебзастосунків у навчальному процесі. У роботі проаналізовано психолого-педагогічні засади використання інтерактивних технологій та потенціал технології React для створення освітніх рішень. На основі цього аналізу розроблено методику навчання, що базується на інтеграції проєктного підходу, методу «живого кодування», використанні хмарних середовищ розробки та інструментів штучного інтелекту.

Ефективність розробленої методики була перевірена шляхом педагогічного експерименту за участю учнів 10 - 11 класів. Результати дослідження засвідчили підвищення рівня навчальних досягнень, інтересу та мотивації учнів, які навчалися за розробленою методикою.

Розроблений підхід може бути інтегрований у вивчення теми «Основи веброзробки» та інших розділів інформатики.

Ключові слова: вебзастосунок, React, MERN-стек, інтерактивне навчання, інформатика, гейміфікація, педагогічний експеримент.

ABSTRACT

Tverdokhlib Y. P. Methodological aspects of teaching students the basics of web development using React technology. Master’s Qualification Thesis for the degree of Master Specialty Secondary Education (Computer Science). Ternopil Volodymyr Hnatyuk National Pedagogical University Ternopil. 2025, 74 p.

This thesis examines the problem of improving the effectiveness of computer science education using modern web technologies and justifies the use of interactive web applications in the educational process. The work analyses the psychological and pedagogical foundations of using interactive technologies and the potential of React technology for creating educational solutions. Based on this analysis, a teaching methodology was developed, which is based on the integration of the project-based approach, the "Live Coding" method, the use of cloud development environments, and artificial intelligence tools.

The effectiveness of the proposed software tool was tested through a pedagogical experiment involving 10th and 11th grade students. The results of the study demonstrated an increase in the level of academic achievement, interest, and motivation of students who studied using the developed methodology.

The developed approach can be integrated into the study of the topic "Fundamentals of Web Development" and other sections of computer science.

Keywords: web application, React, MERN stack, interactive learning, computer science, gamification, pedagogical experiment.

ЗМІСТ

ВСТУП

З швидким розвитком сфери інформаційних технологій та цифровізацією суспільства на систему освіти накладаються нові вимоги. Сучасний ринок праці гостро потребує веб-розробників з навичками в сучасних парадигмах JavaScript, таких як React.

Відсутність порівняння між технологіями, що вивчаються в школі (зазвичай простий HTML/CSS або застарілі мови програмування), та інструментами індустрії є однією з основних проблем сучасної шкільної IT-освіти [2]. Раніше учні вивчали основи алгоритмізації на абстрактних прикладах. Учні, які є «цифровим поколінням», краще мотивовані, коли отримують можливість створити реальний, інтерактивний та естетично привабливий продукт (вебсайт або вебдодаток).

Хоча в Інтернеті є більше освітніх ресурсів, ніж будь-коли раніше (курси, підручники), виникла нова проблема, а саме відсутність наукової основи та адаптованого способу впровадження складних технологій у шкільний процес. Вчителі інформатики зазвичай не мають доступних рекомендацій для викладання складних концепцій React (компонентний підхід, керування станом компоненти, хуки) своїм учням у легкозрозумілій та послідовній формі.

Саме у цьому контексті набуває актуальності розробка якісних методичних підходів. Така стратегія базується на усвідомленні, що освітній процес має бути практично спрямованим та відповідати інтересами учнів.

Отже, **актуальність дослідження** полягає в необхідності теоретичного обґрунтування та розробки методичних підходів до навчання учнів основам веброзробки з використанням технології React для покращення якості їхньої IT-освіти.

Метою дослідження є теоретичне обґрунтування, розроблення та експериментальна перевірка ефективності моделі інтерактивного освітнього вебзастосунку на основі React для підвищення якості засвоєння знань учнями з інформатики.

Для досягнення поставленої мети були визначені такі **завдання**:

1) дослідити психологічні та педагогічні аспекти щодо використання інтерактивних технологій в процесі навчання учнів основам веброзробки;

2) здійснити аналіз сучасних освітніх платформ та технологічних особливостей технології React як інструменту для розробки освітніх вебзастосунків;

3) розробити методику навчання, що базується на інтеграції проєктного підходу, методу «живого кодування», хмарних середовищ та інструментів штучного інтелекту;

4) експериментально перевірити ефективність розробленої методики та підготувати практичні рекомендації для її інтеграції у освітній процес.

Об’єкт дослідження – процес навчання інформатики з використанням інтерактивних вебзастосунків у сучасному освітньому середовищі.

Предмет дослідження – методика створення та використання інтерактивних навчальних вебзастосунків на основі технології React для підвищення ефективності засвоєння знань учнями.

Практичне значення отриманих результатів полягає у розробці набору навчально-методичних матеріалів (тематичний план, конспекти, система практичних завдань та критерії оцінювання), які можуть бути безпосередньо використані вчителями інформатики для впровадження навчального курсу «Основи веброзробки на React», що ефективно модернізує зміст шкільної ІТ-освіти та значно вплине на мотивацію учнів.

Кваліфікаційна робота створює системний методичний підхід, заснований на доказах, який сприяє покращенню та прискоренню здатності учнів вивчати складні концепції сучасної веброзробки. Дослідження зосереджено на навчанні інформатики (технології веброзробки) учням загальної середньої освіти.

Запропонований підхід може бути використаний вчителями, які бажають надати своїм учням корисні та затребувані знання, які можуть вимагати ринок праці, та учнями, які хочуть зробити кар'єру в інформаційних технологіях.

Апробація та впровадження результатів дослідження. Результати дослідження обговорювалися на двох міжнародних науково-практичних

конференціях, зокрема на XV Міжнародній науково-практичній інтернет-конференції[12] (м. Тернопіль, 10 квітня, 2025 р.) та XVI Міжнародній науково-практичній інтернет-конференції[13] (м. Тернопіль, 6-7 листопада, 2025 р.).

РОЗДІЛ 1. ТЕОРЕТИКО-ПЕДАГОГІЧНІ ЗАСАДИ НАВЧАННЯ ОСНОВАМ ВЕБРОЗРОБКИ

1.1 Веброзробка як ключова складова у структурі шкільної ІТ-освіти

Вхід до інформаційного суспільства та цифрової економіки ставить нові проблеми перед системою загальної середньої освіти. Тому, коли ми зосереджуємося на освіті учнів у закладах загальної середньої освіти [4], ми дивимося за межі навчальної програми і в соціальну систему. Сьогодні інформатика вийшла за межі вузької спеціалізації. Вона стала універсальною навичкою (наскрізною компетенцією), яка є необхідною умовою для професійного та особистого розвитку кожної людини [8].

Цей розділ має на меті дослідити статус-кво навчання інформатики та підкреслити тенденції, які мотивують еволюцію змісту та викладання. Ми стикаємося з викликами та проблемами. Поточний стан ІТ-освіти в українських школах зазнав значних змін, але все ще має багато серйозних труднощів. Від «грамотності» до «компетентності».

Шкільна інформатика довгий час зосереджувалася на розвитку цифрової грамотності – навчанні використанню готових програмних продуктів (текстові та табличні процесори, графічні редактори тощо). Сучасний виклик, який був впроваджений у Новій українській школі (НУШ), полягає в переході до цифрової компетентності [7]. Це означає створення нових цифрових продуктів, розуміння функціонування систем та вирішення проблем через програмування.

Розвиток ІТ-сектору значно випереджає оновлення державних освітніх стандартів і навчальних програм. Учні часто вивчають технології (наприклад, Pascal, традиційні підходи до HTML/CSS), які вже не відповідають вимогам ринку праці. Ця невідповідність призводить до низької мотивації між тим, що очікують учні (сучасні вебсайти та мобільні додатки), і тим, що фактично викладають на уроках [9].

Існує велика різниця в матеріально-технічному забезпеченні шкіл. Порівняно з ліцеями у великих містах з сучасним обладнанням та високошвидкісним доступом до інтернету, школи в сільській місцевості можуть

бути обмежені в ресурсах для онлайн-середовищ, що ускладнює впровадження сучасних підходів, які акцентують ці практики.

Впровадження нових знань для учнів у молодому віці (наприклад, веброзробка) часто не є ефективним, якщо вчитель не підготовлений до цього. Виклик «подвійного навантаження» для вчителів інформатики вимагає подвійних навичок експертів-педагогів-методистів та вміння йти в ногу з постійно змінюваними тенденціями в технологіях [1].

І все ж, кілька ключових, але стабільних тенденцій з'являються, які будуть очевидними в найближчі роки шкільної ІТ-освіти. Впровадження НУШ (компетентнісний підхід). Нова українська школа зосереджується не на знаннях у їхній сумарній цінності, а на можливості застосування їх на практиці. Це відкриває шлях до того, щоб учні, замість просто теоретизувати, будували власні теорії та проєкти [5].

Мови програмування стають реальністю. Дійсно, відбувається перехід від вивчення «академічних» мов (зокрема Pascal) до мов, конкурентоспроможних на ринку. Python став фактичним стандартом у початковому курсі програмування завдяки своїй дуже прямій та зрозумілій синтаксису. Водночас JavaScript набирає популярності як веб-мова, що є необхідною перед будь-якою сучасною веброзробкою [11].

Проєктне навчання – це реакція на проблему мотивації. Коли учні бачать «кінцеву» мету – наприклад, створення власного блогу або інтерактивного додатку – вони набагато охочіше досліджують складні теми! Це змушує вчителя переходити від ролі вчителя до ролі фасилітатора або наставника.

Все більше інформатика сприймається не як окрема дисципліна, а як крос-дисциплінарний спосіб вирішення проблем в інших галузях (природничі науки, математика, інженерія).

Аналіз, проведений у контексті поточного стану, також вказує на гостру суперечність. Існує чітка соціальна та освітня потреба в практичному навчанні деяких технологій, пов'язаних з ними (переважно використання веброзробки). З іншого боку, існує методологічний вакуум – відсутність наукових методів,

заснованих на фактах і перевірених як способи навчання учнів (наприклад, сучасні JavaScript-технології), які є складними та затребуваними. Більшість існуючих програм починаються з HTML, CSS та трохи JavaScript. Перехід до компонентного підходу, декларативного UI та управління динамічними даними, що є основними концепціями React, є унікальною методологічною підготовкою, адаптованою до старшокласників та їхніх психологічних і педагогічних особливостей.

1.2. Дидактичний потенціал JavaScript-технологій (React, Angular, Vue) у формуванні цифрових компетентностей учнів

Там, де веброзробка в шкільній IT-освіті раніше обмежувалася статичними технологіями (HTML, CSS), сьогоднішня індустрія вимагає динамічних, інтерактивних веб-додатків [2]. Використання JavaScript технологій, які є промисловим стандартом для створення складних інтерфейсів користувача, робить цей перехід нездійсненним.

Цінність впровадження технологій в освіту полягає в тому, що вони мають велику дидактичну цінність, оскільки дозволяють учням здобувати специфічні навички та важливі цифрові компетенції:

1. Алгоритмічне та логічне мислення: обробка даних, стан додатку та взаємодія з користувачем.
2. Системне мислення та декомпозиція: розбиває складне завдання, таке як додаток, на невеликі, окремі та керовані частини – компоненти.
3. Навички вирішення проблем (налагодження): це процес пошуку проблем та їх виправлення, коли вони з'являються у вашій програмі.
4. Проєктна діяльність: здатність планувати та виконувати цілий програмний продукт [6].

Для обґрунтованого вибору інструментарію нашої методики, проаналізуємо три найпопулярніші технології (таблиця 1.1) з точки зору їх дидактичного потенціалу.

Таблиця 1.1

Порівняльний аналіз технологій веброзробки

в освітньому контексті

Критерій	**Angular (від Google)**	**Vue.js**	**React (від Meta)**
Поріг входження	Дуже високий	Низький	Середній
Філософія	Комплексний фреймворк («все з коробки»)	Прогресивна технологія (легко почати, можна ускладнювати)	Бібліотека для UI (фокус лише на відображенні)
Вимоги (Перешкоди)	Вимагає знання TypeScript та концепцій ООП. Складна архітектура.	Майже відсутні. Можна почати з підключення скрипта.	Потрібне добре розуміння «чистого» JavaScript (ES6+).
Дидактичний аспект	Недолік: Надмірна складність для шкільного курсу. Забагато абстракцій	Перевага: Найкраща крива навчання, чудова документація.	Перевага: Найпопулярніший в індустрії (висока мотивація). Фокусує учня на ключових концепціях (компоненти, стан).

Дидактичні переваги методології React. Як видно з таблиці 1.1, хоча Vue пропонує найнижчий поріг входження, React є набагато професійнішим у здатності досягти оптимального середовища між легкістю використання та професійною актуальністю. Для цієї роботи React був обраний як базова технологія через свої особливі дидактичні переваги:

Перевага 1. Архітектура на основі компонентів. Суть полягає у тому, що React спонукає вас думати про інтерфейс як про колекцію незалежних, багаторазових «блоків» (компонентів). З дидактичної точки зору це ідеально тренує декомпозицію, одну з основних навичок програмування. Користувач знаходить способи декомпозувати складний інтерфейс користувача на прості компоненти (<Avatar>, <UserInfo>).

Перевага 2. Декларативний інтерфейс користувача. Учень каже, що він хоче бачити на екрані – а не як це змінити (на відміну від імперативного в «чистому» JavaScript, де інструкція може бути написана з дієсловом) на увазі. Це допомагає значно зменшити когнітивне навантаження. Для цього учня акцент робиться на логіці програми, тобто на зв'язку даних та інтерфейсу користувача, а не на технічних деталях маніпуляції DOM.

Перевага 3. Концепція «стану» є основною. Суть у тому, що React представляє пряму та просту концепцію «стану» – даних, які можуть змінюватися і на яких, здається, базується зовнішній вигляд будь-якого компонента. Використовуючи приклад Hook – useState, учні легко розуміють патерн «якщо дані змінилися – то інтерфейс автоматично оновився», що формує основу для взаємодії.

Перевага 4. Висока мотивація. React – це технологія №1 в індустрії. Це випливає з результатів опитування, зображеного на рис. 1.1. Дидактична цінність у тому, що учні знають, що вони беруть на себе роль використання інструменту, який є загальним для Facebook, Instagram, Netflix. Це дуже важливо для підтримки та мотивації старших класів [6].

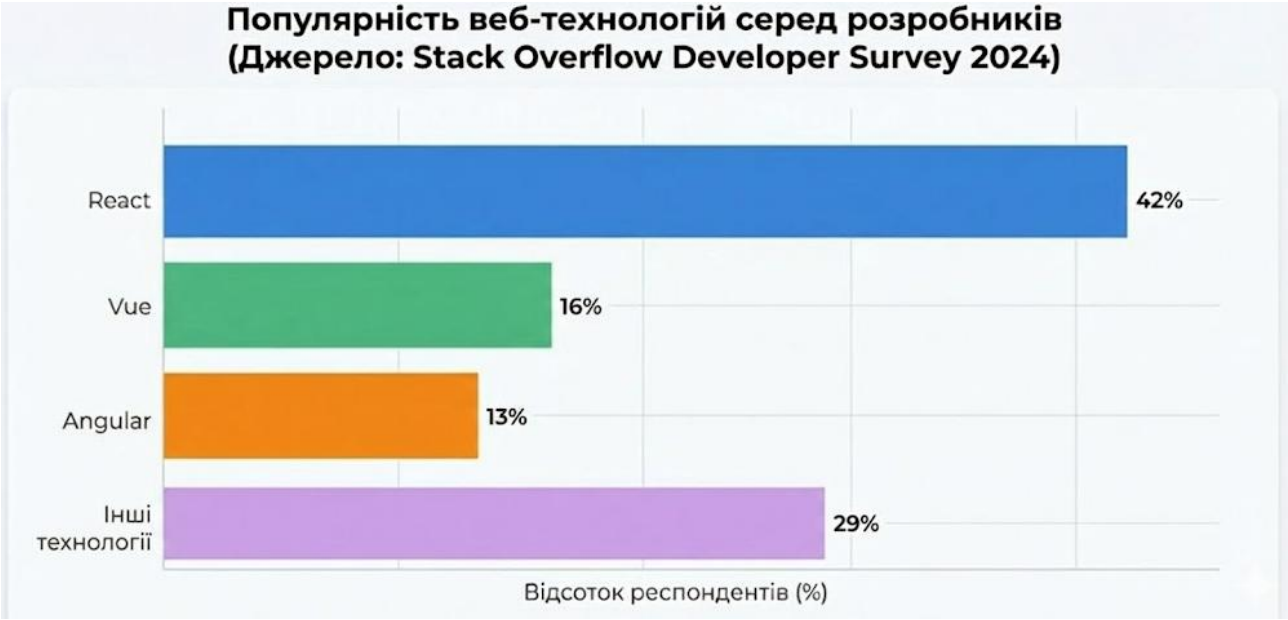

Рис. 1.1. Діаграма популярності веб-технологій

Представлені дані на стовпчиковій діаграмі відповідають глобальному опитуванню розробників Stack Overflow 2024 року і надають чітке розуміння розподілу сил. Аналіз ринкової ситуації, представлений у візуалізації, демонструє, що React є беззаперечним лідером, оскільки його обрали 42% респондентів. Це не лише ставить його на перше місце, але й робить його лідером серед інших конкурентів за часткою ринку, оскільки React має більше ніж удвічі більшу частку, ніж Vue (16%), і втричі більшу, ніж Angular (13%). Такий величезний розрив показує, що React тепер є фактичним стандартом у фронтенд-розробці.

На основі цього аналізу, React є найкращою технологією для застосування у розробці методології для учнів старшої школи, оскільки вона має професійну актуальність, викликає сильні мотиваційні ефекти та містить набір дидактично цінних концепцій.

1.3. Психолого-педагогічні особливості навчання програмування учнів старших класів

Предмет програмування є цілковито відмінним від решти навчальних дисциплін, що викладаються в школі. Він вимагає від учнів не просто механічно вивчати синтаксис, а створювати спосіб мислення – алгоритмічний та абстрактно-логічний. Неможливо визначити ефективність методології викладання React у старших класах (10-11 класи) без оцінки значних психологічних та педагогічних особливостей цієї вікової групи.

По-перше, старшокласники знаходяться на стадії формальних операцій з когнітивної точки зору. Це означає, що вони вже здатні мислити гіпотетично та застосовувати абстрактні ідеї без фізичного втілення безпосередньо [10]. У той час як учні середньої школи потребують візуальних блокових середовищ, старшокласники емоційно готові до роботи з текстовими мовами та складними ідеями. Такі абстракції в React, як «компонент» (наприклад, «шаблон» для UI), «стан» та «властивості» (як контракт для передачі даних) повністю відповідають їх когнітивним можливостям.

По-друге, фундаментальною ознакою успіху є мотивація. Для старшокласника на професійному етапі самовизначення абстрактні «академічні» завдання можуть бути деморалізуючими. Вони хочуть відчутних і значущих результатів своїх зусиль. Знайомство з сучасними веб-технологіями [2] та проєктним методом [6] є рушійним впливом. Коли учні бачать, що вони роблять реальну річ (веб-сторінка, блог), де використовується інструмент провідних IT-компаній – React, це підвищує залученість.

По-третє, слід враховувати питання високого когнітивного навантаження. Теорія Дж. Свеллера [38] говорить, що програмування є діяльністю з високою внутрішньою складністю. Учень повинен одночасно утримувати в робочій пам’яті синтаксис, логіку програми, структуру даних та потік їх передачі. Таким чином, корисні методи повинні мінімізувати непотрібне когнітивне навантаження [3]. Наприклад, впровадження хмарних технологій [1], через онлайн-середовища («пісочниці», як у CodeSandbox) дозволяє уникнути складного завдання налаштування локальної інфраструктури (node.js, npm) і зосередити увагу учня саме на елементах програмування.

По-четверте, навчання – це не «сума знань», а формування компетенцій. Навички програмування є «крихкими» і втрачають свою гостроту на практиці швидше, ніж були вивчені для «праці думки» , якщо не застосовуються постійно. Тому підхід повинен спиратися не на академічні експерименти, а на підкріплення практичними, поступово складнішими завданнями, а потім фінальним проєктом, де учень використовує накопичені компетенції [6].

Нарешті, психологічною перешкодою для навчання є прийняття помилок. Повідомлення про помилки в консолі не є зворотним зв'язком для учня для покращення, а скоріше як невдача особистого характеру. Це викликає стрес і «вивчену безпорадність», що є основною перешкодою у вивченні програмування. Викладання методології має на меті зробити помилки нормативними [42]. Вчитель повинен пройти процес налагодження («живе кодування»), пояснюючи, що знаходження та виправлення помилок є нормальною, а не винятковою частиною роботи розробника.

Таким чином, ефективний підхід до викладання React у старших класах включає навички, орієнтовані на компетентнісний підхід [8], а також проєктний підхід [6], відповідно до рівня готовності учнів до абстрактного мислення [10], одночасно мінімізуючи когнітивне навантаження, яке надмірно виникає [3], і підтримуючи мотивацію через актуальність технології.

1.4. Огляд існуючих освітніх ресурсів та засобів навчання основам React

На основі вибору React (пункт 1.2) та психологічних і педагогічних характеристик учнів (пункт 1.3) необхідно проаналізувати існуючі освітні ресурси. Цей аналіз має на меті визначити ефективність існуючих інструментів та курсів у виконанні освітньої мети шкіл та виявити «методологічний вакуум», який це дослідження має на меті заповнити. Сучасний ринок освітніх IT-сервісів пропонує численні ресурси для вивчення React. Для аналізу ми можемо класифікувати їх за форматом та дидактичними характеристиками.

Як показано в Таблиці 1.2, жоден з існуючих типів ресурсів не може самостійно забезпечити систематичне та методологічно обґрунтоване навчання основам React у середній освітній установі.

Таблиця 1.2

Сучасні освітні ресурси для вивчення React

Категорія ресурсу	Приклади	Переваги	Недоліки (для учнів ЗЗСО)
Офіційна документація	react.dev	Актуальність та повнота. Це першоджерело. Інтерактивні приклади (нові React Docs).	Високе когнітивне навантаження [3]. Написана для розробників, а не для школярів. Складна професійна лексика, високі вимоги до знання англійської

Продовження табл. 1.2

Масові онлайн-курси (MOOC)	Prometheus, Coursera, Codecademy, FreeCodeCamp	Структурованість. Матеріал подається послідовно. Інтерактивність (вбудовані редактори).	«Один розмір для всіх» [42]. Відсутня гнучкість та адаптація до темпу класу. Орієнтовані на самостійну, вмотивовану дорослу аудиторію. Часто формують навички, а не компетентності [8].
Відео-контент	YouTube, авторські відео-курси	Наочність. Добре підходить для візуального сприйняття. Демонстрація «живого» процесу кодування.	Пасивне споживання. Учень часто дивиться, але не осмислює. Фрагментарність та різна якість контенту. Відсутній зворотний зв'язок та педагогічний супровід.
«Хмарні» середовища (IDE)	CodeSandbox, Replit, StackBlitz	Усунення бар'єру входу. Дозволяють почати кодувати одразу в браузері, без складного налаштування [1].	Це інструмент, а не методика [15]. Вони не містять навчального контенту, теорії чи системи завдань. Вчитель має створювати методику «з нуля».

Як видно з рис. 1.2, існують проблеми впровадження React у шкільну освіту.

Проблема №1. Відсутність педагогічної адаптивності. Масові онлайн-курси та офіційна документація розроблені для дорослих, мотивованих студентів, а не для учнів 10-11 класів. Вони не враховують надмірне когнітивне навантаження [3] та психологічні особливості підлітків.

Проблема №2. Фрагментація. Один вчитель може використовувати деякі компоненти: CodeSandbox для практики [1], YouTube для візуалізації. Проте немає системи цілісних методологій, які б об'єднували ці інструменти в єдиний логічний навчальний процес, заснований на проєктному методі [6].

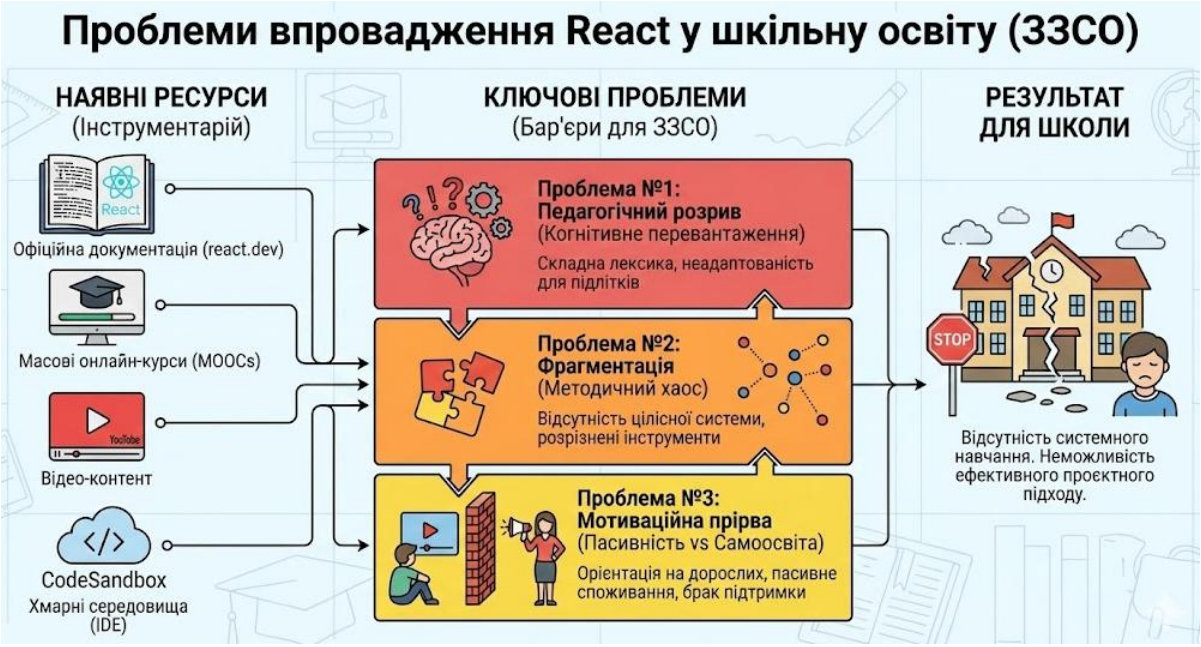

Рис. 1.2. Ілюстрація проблем впровадження технології React у шкільну освіту

Проблема №3. Орієнтація на самоосвіту. Висока внутрішня мотивація учня є найбільш покладеною на ресурси. У шкільному середовищі нам потрібна стратегія, яка розглядає вчителя у ролі наставника та використовує перевірені навчальні тактики.

Висновки до розділу 1

Обговорення в Розділі I (стан IT-освіти, дидактичний потенціал React, психологічні характеристики учнів та існуючі матеріали) вказує на існування серйозної наукової та методологічної проблеми. Відсутній систематичний, науково обґрунтований спосіб вивчення React для учнів, який би:

– враховував вік та когнітивні здібності учнів;
– був доставлений через сучасні пристрої (хмарні інтегровані середовища розробки, AI-асистенти);
– був розроблений для залучення компетентнісного [8] та проєктного [6] підходу;
– забезпечував вчителя готовими матеріалами для керівництва.

Такий методологічний підхід, що розробляється в цій роботі, є центральним для розділу 2, цієї роботи.

РОЗДІЛ 2. МЕТОДИЧНІ ПІДХОДИ ДО НАВЧАННЯ УЧНІВ ОСНОВАМ REACT

2.1. Методичні аспекти відбору змісту: обґрунтування «ядра» навчального матеріалу (компоненти, JSX, props, state, hooks)

Вибір та організація навчальної інформації є першим і головним методологічним завданням. Етап дидактичного проєктування змісту навчального курсу є основоположним етапом для ефективності курсу навчання [38]. Хоча навчання React у загальноосвітніх навчальних закладах, на відміну від професійної підготовки розробників, не має на меті знати всі можливості технології, а навчитися основним навичкам [8] та сучасній парадигмі веброзробки [2].

Враховуючи психологічні та педагогічні особливості учнів (пункт 1.3), особливо необхідність контролю та управління когнітивним навантаженням [3] та заохочення мотивації як наслідок проєктного навчання [6], ми створили «мінімально життєздатне ядро» для навчального матеріалу для учнів. Це ядро, яке зображене на рис. 2.1, є мінімальним набором елементів, необхідних для того, щоб учень міг розробити реальний інтерактивний проєкт.

Рис. 2.1. Схема «Навчальне ядро React»

Принцип виключення являє собою те, що ми не вчимо. Методологічний вибір є свідомим, категоричним кроком у відмові від непотрібних і надто складних концепцій, які є просто «занадто складними» або «занадто малими чи зайвими» для початкового рівня. Введення цих тем у ваш вступ до курсу значно підвищує когнітивне навантаження [3] і заважає учням стати компетентними для освоєння основ.

1. Компоненти класу повністю виключені. Сучасний React (з 2018 року) розроблений навколо функціональних елементів і хуків [34]. Вивчення класів безпідставно ускладнює систему (концепція this, конструктор, render), що дидактично безпідставно або просто занадто заплутано для учня [30].
2. Складне управління станом (Redux, MobX, Context API). Для шкільних освітніх проєктів локальний стан (useState) є абсолютно адекватним. Глобальне управління станом є відповіддю для масивних додатків, воно вводить велику кількість «boilerplate» коду, тому важко навчити учнів заздалегідь.
3. Використання розширених хуків (useCallback, useMemo, useReducer, useEffect) для оптимізації, ні учень, ні я не можемо інтерпретувати (вони не зустрічаються в освітніх проєктах). useEffect є найскладнішим хуком для обробки «побічних ефектів» (наприклад: запити API). Ця концепція (життєвий цикл, залежності) занадто складна для початківців. Ми замінюємо на імітацію і зберігаємо дані в статичному масиві.
4. TypeScript – це стандарт індустрії, але він вводить додатковий шар абстракції (система типів), зайвий для простого веброзробки і подвоює когнітивне навантаження [23].

Дидактична послідовність нашого курсу - це мінімальна кількість концепцій, необхідних учню для самостійного створення повноцінного інтерактивного мікропроєкту. Це називається «ядром» [6]. Дидактична модель важлива для процесу, де зміст нової теми логічно випливає з іншої і вирішує нову

проблему. Діаграма 2.2 показує цю послідовність (див. Діаграма 2.2 − Логічна структура «ядра» навчального матеріалу).

Рис. 2.2. Структура "ядра" навчального матеріалу

Крок 1: Візуальний блок (JSX і компоненти).

На початковому етапі навчання виникає необхідність переходу від статичної розмітки HTML до динамічного формування інтерфейсів. Для вирішення цього завдання впроваджується синтаксис JSX (JavaScript XML), який дозволяє інтегрувати розмітку безпосередньо в JavaScript-код. Завдяки візуальній подібності до звичного HTML та можливості отримати миттєвий результат, цей підхід знижує поріг входження та суттєво підтримує мотивацію учнів. Паралельно вводиться поняття функціонального компонента, де через аналогію з конструктором (наприклад, LEGO) пояснюється принцип створення власних «будівельних блоків» або кастомних тегів. У результаті опанування

цього етапу здобувач освіти набуває здатності самостійно конструювати веб-сторінки, сформовані з власних ізольованих елементів.

Крок 2: Передача даних (Props).

Після опанування структури компонентів виникає проблема однотипності контенту: створений елемент, наприклад <UserProfileCard />, відображає однакову інформацію при кожному використанні. Для вирішення цього завдання вводиться концепція props (властивостей), які методично пояснюються як атрибути для кастомних тегів. Це дозволяє реалізувати механізм передачі даних за принципом «зверху вниз» (наприклад, <UserProfileCard username="Іван" />), забезпечуючи варіативність відображення. Результатом етапу є здатність учня створювати універсальні компоненти, придатні до багаторазового використання з різними вхідними даними, а також розуміння логіки потоку інформації в системі.

Крок 3: Інтерактивність та «Пам'ять» (State та useState).

Попри те, що використання props дозволяє відображати різний контент, компоненти залишаються статичними та не реагують на взаємодію з користувачем. Оскільки props передаються ззовні, вони не придатні для збереження динамічних змін, таких як кількість натискань кнопки чи статус активності елемента. Для вирішення цієї проблеми вводиться поняття стану (state) — фундаментальної концепції, яку доцільно пояснювати через аналогію з внутрішньою пам'яттю компонента. Практична реалізація здійснюється за допомогою хука useState. Цей інструмент надає компоненту здатність зберігати дані та автоматично оновлювати відображення при їх зміні. Варто зазначити, що методика навчання хукам є предметом сучасних педагогічних досліджень [9]. Результатом цього етапу є перехід до створення інтерактивних інтерфейсів, де учень чітко усвідомлює принцип реактивності: зміна внутрішнього стану автоматично ініціює оновлення інтерфейсу користувача.

Крок 4: Практичні рішення (Події та списки).

Навіть розуміючи концепцію стану, учень стикається з проблемою неможливості його зміни без налаштованої взаємодії. Для вирішення цього

завдання вводяться обробники подій, зокрема onClick, які виступають сполучною ланкою між діями користувача та функціями оновлення стану. Наступним викликом є необхідність масштабування інтерфейсу: ручне створення великої кількості компонентів (наприклад, 100 карток) є неефективним. Тому навчальний процес переходить до рендерингу списків, актуалізуючи знання «чистого» JavaScript, а саме методу .map(). Це дозволяє трансформувати масиви даних у масиви JSX-елементів. На цьому етапі також розглядається важливий технічний аспект оптимізації рендерингу за допомогою унікальних ключів (key). У результаті здобувач освіти отримує навички створення динамічних інтерактивних списків — основи таких сучасних систем, як стрічки соціальних мереж, електронні каталоги чи списки завдань, що є необхідною базою для розробки фінального проєкту.

Таким чином, обґрунтоване «ядро» навчального матеріалу є гарною лінійною послідовністю JSX/Компоненти (статика), Props (перевикористання), Стан/Події (інтерактивність), Списки (динамічні дані). Це особливо підходить для побудови складних абстракцій, дозволяючи учневі просуватися на зростаючих етапах з суворим контролем інтелектуального навантаження [3], пропонуючи лише відповідні та достатні навички для реалізації навчального проєкту [37]. Це також вказує на те, що методології, які використовують «виключення непотрібного», для проєктування навчального контенту успішні [14].

2.2 Методичні аспекти організації навчального процесу: застосування проєктного навчання, live coding та інтерактивних середовищ (CodeSandbox)

Ефективність оволодіння «основним» навчальним матеріалом, обраним раніше (компоненти, пропси, стан), залежить не лише від технічних компонентів, але й від форм та методів організації навчальної діяльності. Однак, теорія спочатку, потім практика – принцип традиційної лекційно-практичного підходу не був дуже ефективним у навчанні сучасних технологій веброзробки, оскільки

така цифрова трансформація вимагає динамічних ментальних моделей, які не можуть бути виражені у статичних слайдах або нотатках. Як наслідок, підхід до навчання базується на тріаді методів, які забезпечують діяльнісну методологію: навчання на основі проєктів, live coding (живе кодування) та хмарне інтерактивне середовище (Cloud IDE).

«Стратегія скаффолдингу» та навчання на основі проєктів. Суть методології полягає у відмові від ізольованої лабораторної роботи на користь цілісного навчального продукту, що позначає собою завершений проєкт. Проєктний підхід вважається значно підвищуючим внутрішню мотивацію, яка виникає в учнів завдяки усвідомленню та розумінню реального застосування всього, що вони вивчили [18]. Організація роботи в проєкті базується на принципі скаффолдингу педагогічної підтримки, яка поступово зменшується, коли учні набувають компетентності [43]. Проєкт має бути створений у відповідності до принципів React-розробки: спершу структура, потім компоненти, потім логіка, потім покращення:

1. Етап «Статичний прототип» (візуалізація). Учням пропонується макет, і вони застосовують макет (JSX, компоненти). Тепер увага переміщується на розбиття інтерфейсів, а не на програмну логіку. Це веде до видимого продукту (сторінки), встановлюючи загальний фактор успіху вже на першому уроці.
2. Етап «Параметризація» (абстракція). Вчитель починає з «проблемного» питання: «Як використовувати цей компонент для іншого користувача?». Ми повинні вивчити пропси в цьому сенсі. Пропси замінюють жорстко закодовані дані. Проєкт модифікується.
3. Етап «Анімація» (інтерактивність) – вимагає реакцій на дії користувача (кліки, введення тексту). Це забезпечує логіку використання useState. Учні можуть додавати лічильники, перемикачі тем або поля введення до проєкту.

4. Етап «Масштабування» (списки) – заключний етап, де компоненти перетворюються на активні списки (метод .map()), початкове створення мінімально життєздатного продукту.

Ця організація процесу змінює роль вчителя з передавача знань на наставника, який супроводжує учня у створенні власного продукту.

Метод «живе кодування» та його використання як інструменту когнітивного учнівства [35]. Ви більше не бачите існуючих рішень, представлених у вигляді слайдів, оскільки ваші нові матеріали розкриваються у вигляді живого кодування (рис. 2.3) – написання коду вчителем у реальному часі з коментарями до його дій. Цей підхід відтворює когнітивне учнівство, оскільки ментальні процеси експерта можуть спостерігатися новачками.

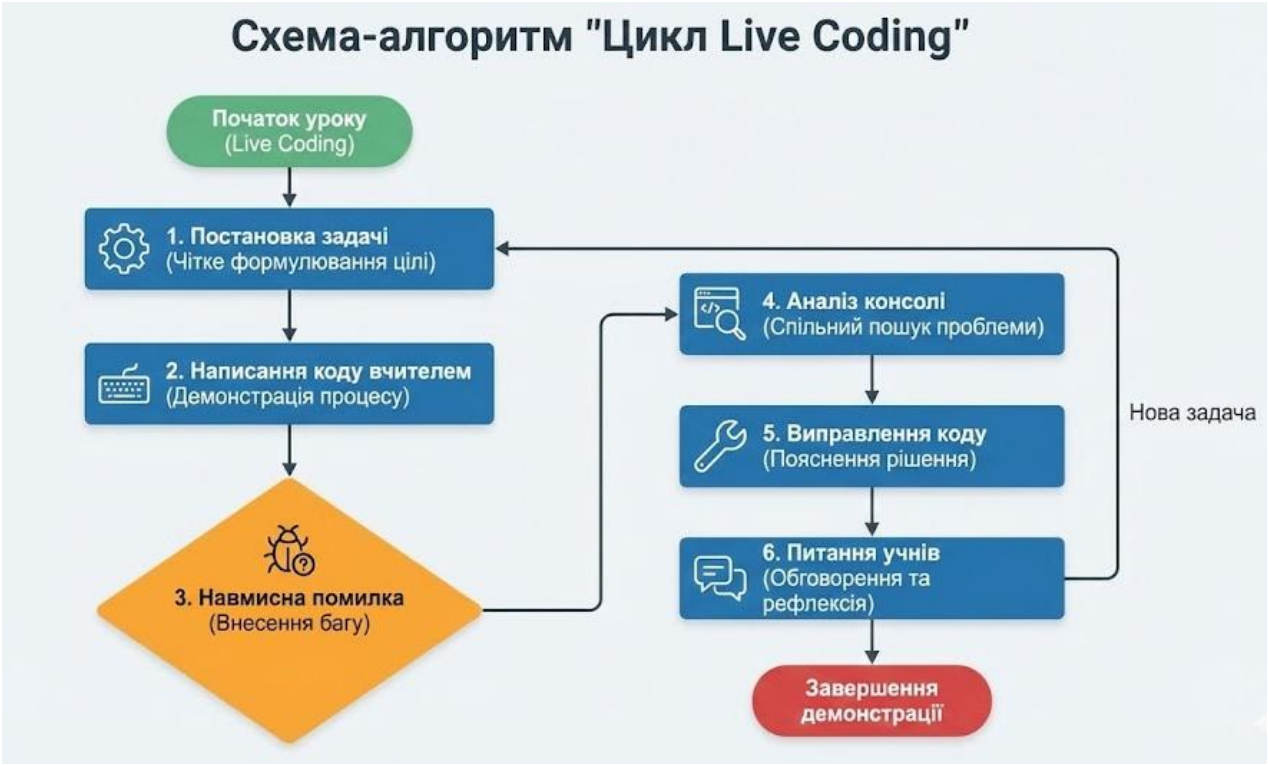

Рис. 2.3. Схема-алгоритм "Цикл Live Coding"

Які педагогічні переваги живого кодування в контексті вивчення React? Демонстрація процесу, а не результату. Учні бачать, що код не слідує лінійному тренду від першого рядка до останнього. Вони спостерігають за бібліотеками, скелетом функції, а потім поступово додається логіка. Порівняльний аналіз традиційного підходу та методу Live Coding наведено в Таблиці 2.1.

Таблиця 2.1

Порівняльний аналіз традиційного підходу та методу Live Coding

Критерій порівняння	Традиційна презентація (Слайди)	Метод Live Coding (Живе кодування)
Фокус уваги	Статичний результат (готовий код).	Процес написання та логіка побудови.
Когнітивне навантаження	Високе (одночасне сприйняття великого обсягу коду).	Помірне (поетапна поява рядків коду).
Реакція на помилки	Помилки приховані, демонструється «ідеальний» код.	Вчитель допускає та виправляє помилки в реальному часі (нормалізація помилок).
Роль учня	Пасивний спостерігач.	Активний учасник (може запропонувати рішення під час кодування).
Розвиток навичок	Запам'ятовування синтаксису.	Розвиток навичок відлагодження коду та алгоритмічного мислення.

Нормалізація помилок. Як мінімум, вчитель робить помилки у синтаксисі та логіці під час живого кодування. Насправді корисніше спостерігати, як інженер інтерпретує повідомлення про помилку в консолі, інтерпретує його та відлагоджує, ніж бачити ідеальний код.

Управління увагою. Код з'являється поетапно, що мінімізує надмірне когнітивне навантаження. Учень бачить лише той рядок, який пишеться і вводиться тут, і, отже, зосереджується лише на ньому (замість того, щоб бачити слайд з 20 рядками коду) одночасно, тоді як презентація з 20 рядками коду зайняла б годину або більше.

Хмарні середовища (CodeSandbox). Для усунення технічних перешкод Однією з найбільших перешкод, з якими стикаються учні при вивченні сучасних

JS-технологій у школі, є проблема створення локального середовища розробки (Node.js, NPM, Webpack). Обмежені права на шкільних комп'ютерах, застаріле обладнання та різноманітність домашніх пристроїв, на яких працюють учні, можуть перешкоджати навчанню ще до написання першого рядка коду.

Методологічний підхід для вирішення цієї проблеми полягає у наданні CodeSandbox (рис. 2.4) – хмарного середовища розробки (Cloud IDE), яке працює в браузері.

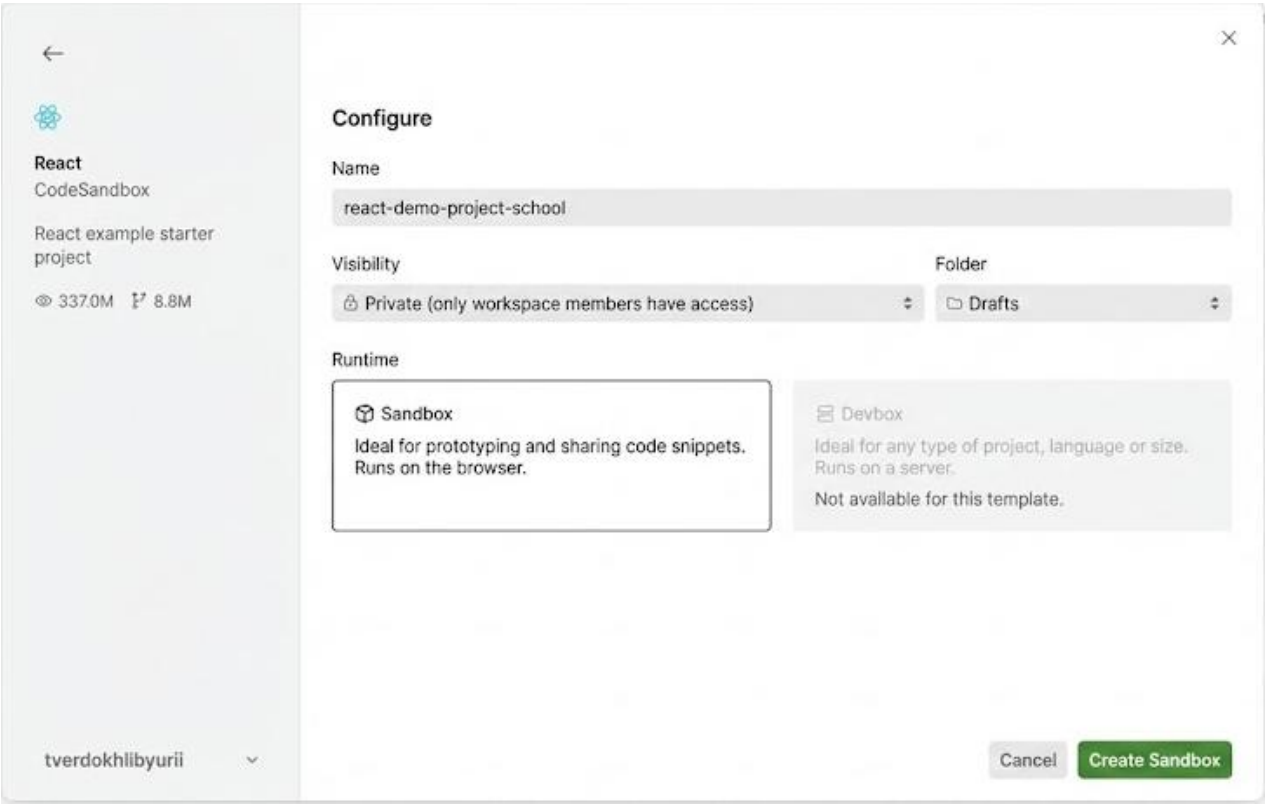

Рис.2.4. Вікно створення нового проєкту

Переваги методологічного використання CodeSandbox. Однією з переваг яку можна виділити є миттєвий старт, тобто нульова конфігурація. Учень отримує посилання на шаблон проєкту і починає роботу за лічені 5 секунд. Це може заощадити від 20 до 30% навчального часу, який зазвичай витрачається на вирішення проблем, таких як «npm install не вдалося» [27].

– середовище ідеальне своєю ізоляцією і безпекою, воно функціонує у «пісочниці», тому не можна пошкодити операційну систему на шкільному комп'ютері.

– середовище формує можливості для зворотного зв'язку. CodeSandbox дає вчителю можливість відкрити посилання на проєкт учня, переглянути код у реальному часі та результати виконання, редагувати або коментувати. Ідеально підходить для дистанційного або гібридного навчання.

– соціальний фактор також важливий аспект. Обмін проєктами через URL підвищує мотивацію учнів і дозволяє їм ділитися своїми проєктами з батьками, друзями або однокласниками.

Підхід, який ми можемо використовувати, це CodeSandbox (рис. 2.5), який змінює акцент навчання з адміністрування (налаштування інструментів) на творчість (кодування), і це важливо на першому етапі вивчення React.

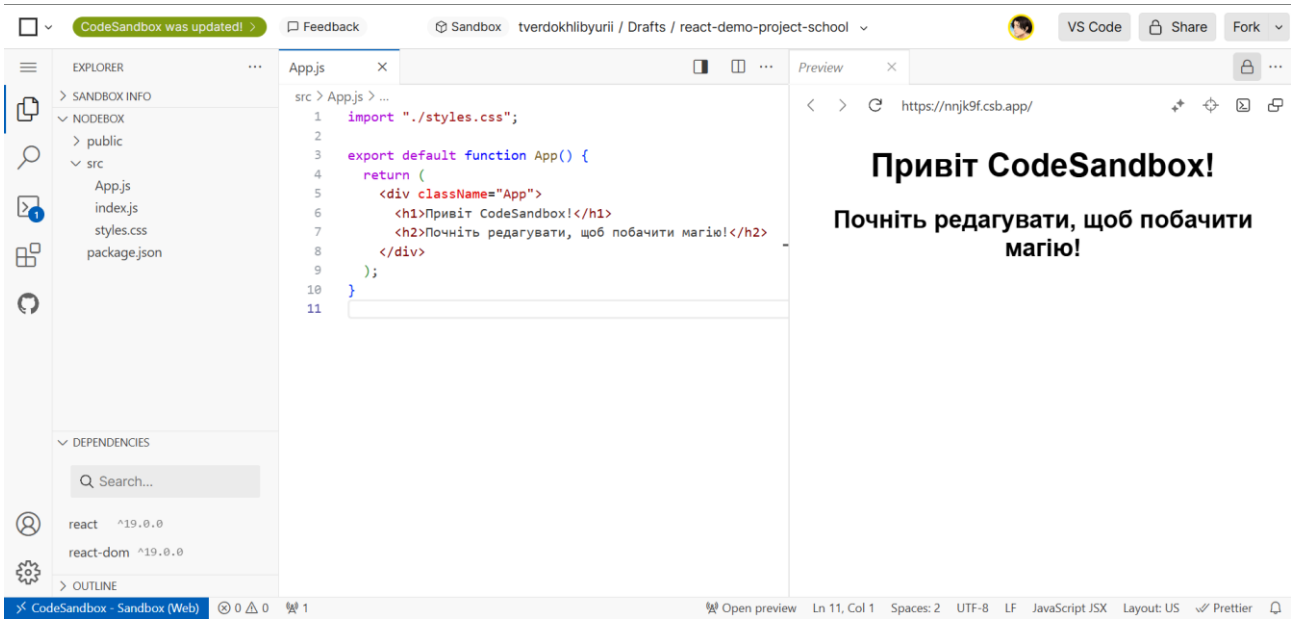

Рис.2.5. Робоче вікно сервісу CodeSandbox

2.3 Методичні аспекти застосування ШІ-асистентів: роль ChatGPT та GitHub Copilot у формуванні навичок дебагінгу та рефлексії коду

У сучасній дидактиці інтеграція інструментів генеративного штучного інтелекту (GenAI) у шкільну програму з інформатики є однією з найбільш обговорюваних тем нашого часу. Використання підказок у минулому вважалося порушенням академічної доброчесності. Але в культурі, де такі інструменти, як

GitHub Copilot і ChatGPT, стали невід'ємною частиною професійного середовища розробника, завдання школи полягає не в тому, щоб їх виключати, а в тому, щоб створити культуру, де їх можна використовувати належним чином [17].

Це підхід, який має велику значимість для навчання. Неконтрольоване використання ШІ призводить до феномену «зони відсутності розвитку», згідно з дослідженнями, коли учень отримує код, який він повинен створити негайно і без витрат когнітивних зусиль на розуміння коду. Таким чином, наш підхід полягає в концептуалізації ролі ШІ не як «генератора рішень», а як «інтелектуального спаринг-партнера» і наставника [31].

Розподіл ролей за дидактичними функціями: Copilot як «партнер», ChatGPT як «наставник». Під час вивчення React ми систематично обираємо застосування у двох різних типах асистентів (див. таблицю 2.2).

Таблиця 2.2

Дидактичний розподіл функцій ШІ-асистентів в навчальному процесі

Характеристика	**GitHub Copilot (Автодоповнення)**	**ChatGPT / Claude (Чат-бот)**
Режим роботи	Синхронний (real-time). Пропонує код під час набору.	Асинхронний. Діалог у чаті.
Методична роль	«Pair Programmer» (Парний програміст). Знімає рутинне навантаження (закриття тегів, шаблонний код).	«Socratic Tutor» (Сократівський тьютор). Пояснює концепції, шукає помилки, проводить рефлексію.

Продовження табл. 2.2

Ризик для учня	Пасивне прийняття коду без перевірки («Tab-completion bias»).	Отримання галюцинацій або занадто складних рішень.
Ключова навичка	Читання та верифікація коду (Code Review).	Формулювання запитів (Prompt Engineering) та дебагінг.

Формування навичок дебагінгу. Стратегія «поясни помилку». Повідомлення про помилки (наприклад, «Minified React error #31», «Too many re-renders») є найскладнішими для розуміння початківцем у React. Це одна з найпоширеніших проблем. Зазвичай це призводить до розчарування і демотивації.

Ми пропонуємо алгоритм використання ChatGPT (див на рис. 2.6) для навчання налагодженню, встановлюючи 3 рівні кроків:

1. Контекстуалізація. Учень переформульовує і не просто відтворює помилку, а надає контекст (частина, де сталася помилка).
2. Запит на пояснення. Учень не повинен просити «виправити код», а «пояснити, чому сталася ця помилка, і надати підказку для вирішення». Це вмикає процес мислення.
3. Аналіз рішення. Учень коментує новий рядок коду після виправлення.

Це робить помилку «навчальним випадком», перетвореним з «перешкоди», де ШІ виступає як перекладач з технічної мови на лексику, яку учень може зрозуміти [33].

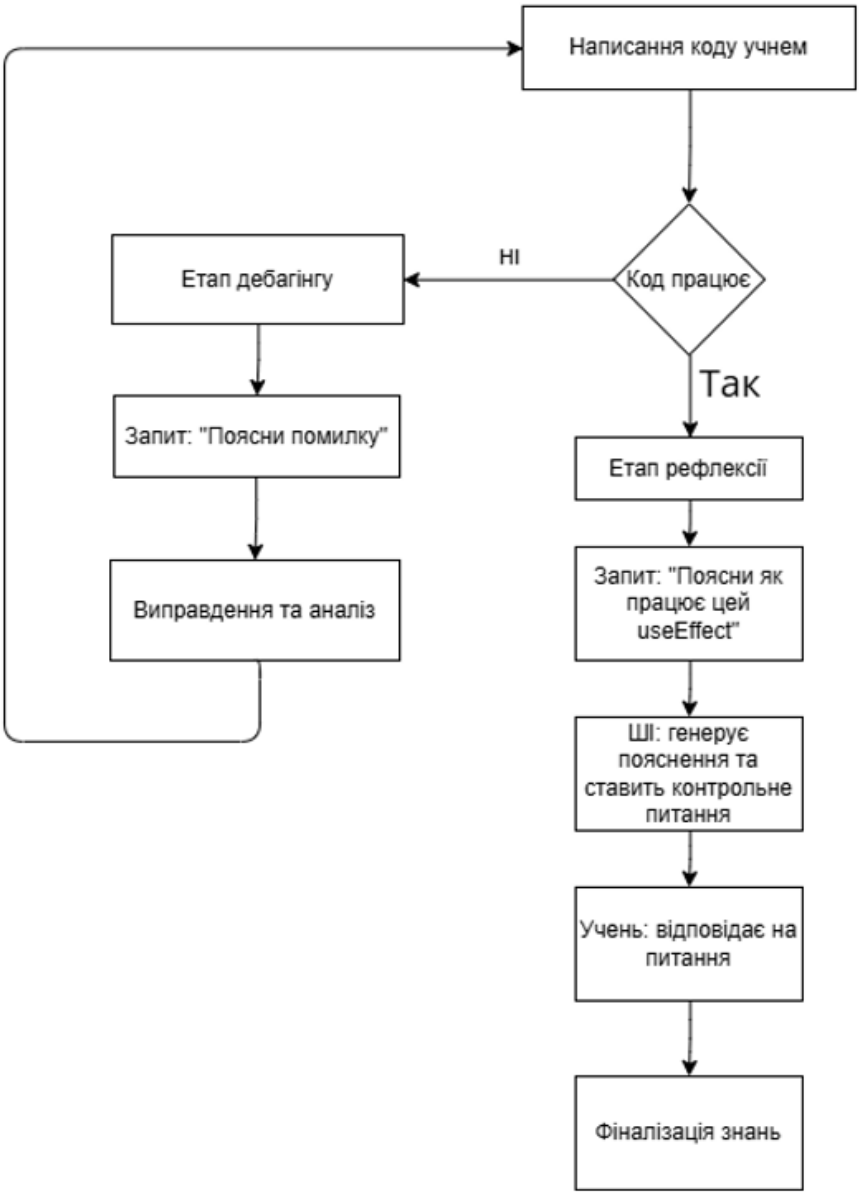

Рис. 2.6. Алгоритм ітеративного процесу навчання програмуванню з використанням штучного інтелекту

Рефлексія коду та подолання ілюзії компетентності. Найбільш ризикованим методологічним ризиком є те, коли код може працювати, а учень не може його зрозуміти (ілюзія компетентності). Щоб вирішити це, створено фазу «Рефлексія з підтримкою ШІ» (рис. 2.7). Після кожного етапу будь-якого проєкту (наприклад, створення компонента UserProfile), учень повинен взаємодіяти з ChatGPT відповідно до сценарію «Аналіз рядок за рядком».

Рис. 2.7. Схема ітеративного процесу навчання програмуванню з використанням штучного інтелекту

Схема на рис 2.7 відображає, що успішний запуск коду не є кінцевою точкою. Тому учень повинен сказати, ШІ: «Згенеруй коментарі для кожного рядка мого useEffect хука, опиши порожній масив залежностей». Це вимагає від учня оцінити свій власний код з точки зору експерта і шукати логічні прогалини, які компілятор ніколи не міг би побачити.

Етичні питання та інженерія підказок. Однією з важливих частин методології є навчання учнів інженерії підказок як професійної навички. Використовуючи модель «Роль-Завдання-Контекст-Формат», ми навчаємо учнів, як структурувати успішний запит (рис. 2.8) [44].

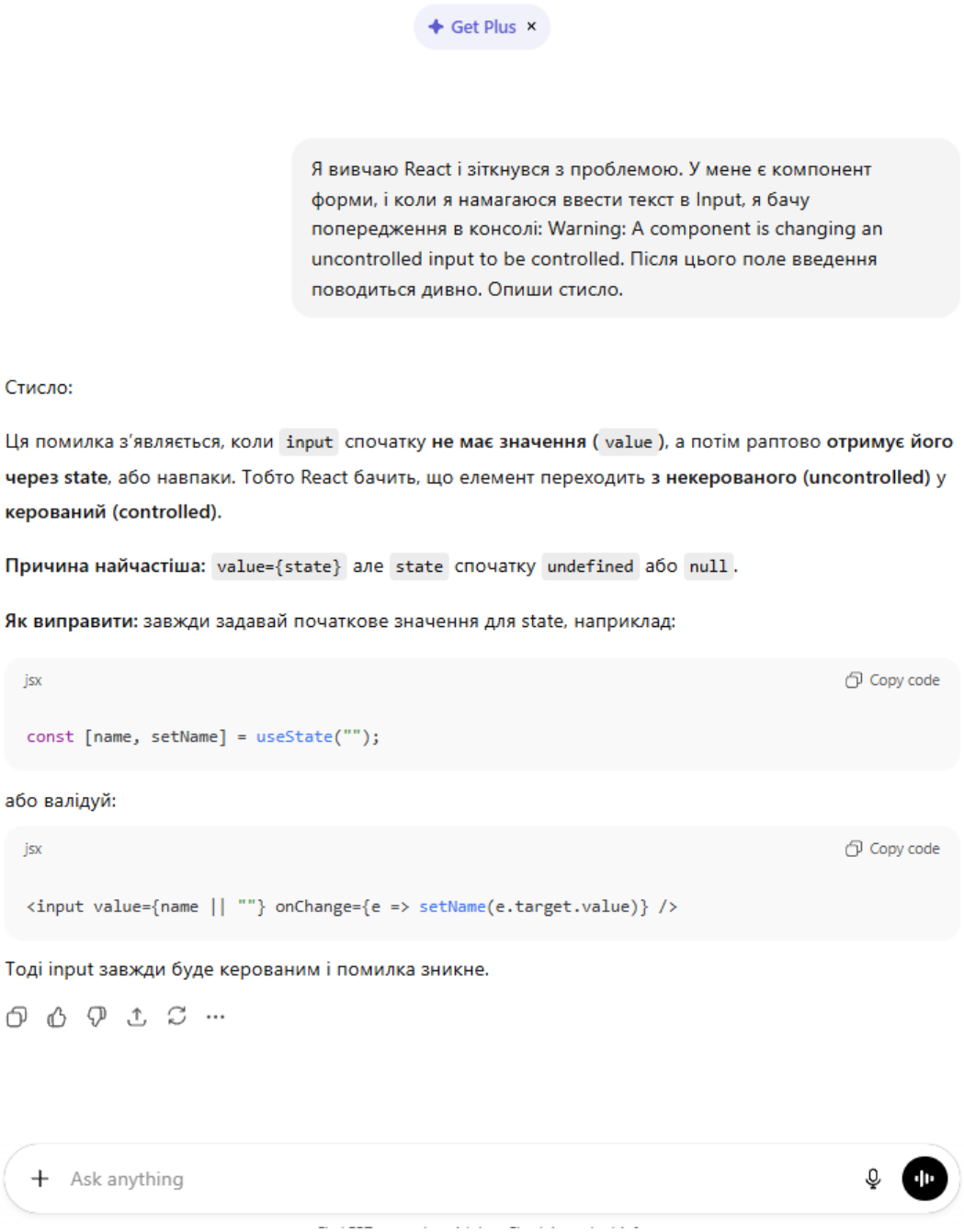

Рис. 2.8. Приклад практичного використання ШІ асистента для вирішення помилок

– поганий запит написаний учнем буде звучати так: «Зроби мені список справ».
– освітній, правильно сформований запит буде виглядати так: «Наставник з React. Я новачок. Спробуй пояснити логіку створення списку завдань з useState та методом .map(), але не потрібно писати весь код для мене, просто вкажи його структуру».

У результаті асистенти ШІ інтегруються в навчання не як доповнення, а як інструменти для його поглиблення, розвитку критичного мислення та навичок самостійного вирішення проблем.

2.4 Методичні аспекти розробки дидактичних матеріалів: система практичних завдань та критерії оцінювання навчальних проєктів

Щоб застосувати практичний підхід навчання на основі завдань для вивчення технології React, необхідно кинути виклик новому способу побудови лабораторних робіт з інформатики. Відмінність компонентного підходу полягає в тому, що ізольоване написання коду (фрагменти коду) не враховує контекст взаємодії компонентів. Таким чином, наші навчальні матеріали в нашому курсі слідують методичному керівництву «Використовуй-Модифікуй-Створюй» для цієї мети, що дозволяє плавно переходити від репродуктивної діяльності до творчої практики, уникаючи когнітивного бар'єру входу.

Структура та типологія практичних завдань. Спіральна структура дозволяє інтегрувати кожну нову функцію в завданні в існуючий контент. Ми відмовляємося від ідеї роботи з «одна тема – одна лабораторна» для наскрізних мікропроєктів. Дидактичний комплекс має три набори завдань, і кожне завдання представляє етап розвитку навичок.

Тип А. Завдання на аналіз та деконструкцію (Використовуй / Передбач-Запусти). Цей тип завдань базується на методиці PRIMM (Передбач, Запусти, Досліджуй, Модифікуй, Створи). Учням надається гіперпосилання на заздалегідь розроблений, працездатний проєкт у CodeSandbox (поширені приклади можуть включати простий лічильник або перемикач теми).

Мета: Зменшити страх перед «чистим аркушем» і розвинути навичку читання чужого коду.

Сценарій роботи учня:

1. Візьміть JSX розмітку та onClick функції, навіть перед запуском, і задокументуйте результати в нотатці: припустіть, що станеться, якщо натиснути кнопку?

2. Запустіть проєкт і перевірте гіпотезу.

3. Вчитель ставить направляючі питання: «Де зберігається початкове значення лічильника?», «Який рядок коду відповідає за зміну кольору?», «Чому дані зникають при оновленні сторінки?».

Учнів заохочують моделювати ментальний зв'язок між кодом (станом) та інтерфейсом (UI) і будувати ментальну модель та працювати над розумінням зв'язку перед написанням першого рядка коду, використовуючи цей підхід.

Тип B. Завдання на модифікацію та підготовку (Модифікуй / Підготовлені завдання). Такі завдання мають неповні умови або частково реалізований код (проблеми Парсонса). Учням надається шаблон проєкту, який реалізує структуру компонентів, але бракує деякої логіки або зв'язків.

Приклад завдання. У вас є макет (JSX) та стилі для проєкту «Картка продукту».

- підзавдання 1. Додайте ціну як props до компонента, щоб ціна не була жорстко закодована, а передавалася від батьківського компонента.
- підзавдання 2. Додайте механізм зміни стану isFavorite (додано до обраного) при натисканні на іконку серця.
- підзавдання 3. Змініть колір іконки, використовуючи тернарний оператор.

Методологічна цінність полягає у зосередженості учнів на конкретному механізмі (наприклад, передача пропсів або синтаксис useState) без відволікання у випадку рутинного CSS макету, що займає до 70% часу заняття.

Тип C. Творчі завдання та налагодження (Створюй / Налагодження). Це найскладніші завдання, які можна класифікувати на дві основні категорії:

- проєкти з нуля. Тобто, учень має лише технічне завдання (ТЗ) та макет. Наприклад: «Створіть додаток To-Do List для додавання завдань, позначення їх як виконаних та видалення. Використовуйте декомпозицію на компоненти Form, List, Item.»
- проєкти з існуючим кодом. Учням надається код з поширеними помилками початківців. Типові помилки для аналізу: пряма мутація

стану (state.value = 5), нескінченний цикл у useEffect, відсутність ключів у списках, втрата реактивності. Завдання: Знайти помилку, обговорити причину помилки з використанням термінології React, такої як цикл рендерингу та незмінність, а потім виправити її.

Критерії оцінювання навчальних проєктів. Компетентнісний підхід. Тому недоцільно оцінювати проєктну діяльність у веб-розробці за традиційною системою оцінювання правильно або неправильно, оскільки можна реалізувати ті ж функціональності різними способами (якісно та неякісно). Ми рекомендуємо впровадження системи оцінювання за допомогою таксономії SOLO (Структура спостережуваного навчального результату) [20], яка зображена у вигляді таблиці 2.3. Це дозволяє оцінювати не тільки факт роботи програми, але й рівень розуміння учнем архітектури React [30].

Таблиця 2.3

Деталізація рівнів майстерності згідно з таксономією SOLO в контексті React

Критерій	**Рівень 1: Початковий (Unistructural)**	**Рівень 2: Середній (Multistructural)**	**Рівень 3: Достатній (Relational)**	**Рівень 4: Високий (Extended Abstract)**
Архітектура та компоненти	Весь код написаний в одному компоненті App.js. Відсутня декомпозиція.	Проєкт розбито на компоненти, але поділ нелогічний. Компоненти занадто великі або дублюють код.	Чітка ієрархія. Дотримано принципу єдиної відповідальності (Single Responsibility). Компоненти перевикористовуються.	Виділено UI-компоненти (презентаційні) та логічні контейнери. Грамотне використання композиції (children).

Продовження табл. 2.3

Робота з даними (Props & State)	Використання глобальних змінних. Зміна пропсів всередині дочірніх компонентів (мутація).	Стан використовується, але він надлишковий (дублювання даних). Проблема «Props Drilling» (передача через 3+ рівні).	Стан мінімально необхідний (DRY). Використано підняття стану (Lifting State Up) для синхронізації компонентів.	Складну логіку винесено у кастомні хуки (Custom Hooks). Ефективна структура даних.
Інтерактивність та Логіка	Обробники подій написані в один рядок (в HTML). Логіка змішана з розміткою.	Функції-обробники винесені, але містять помилки в оновленні об'єктів/масивів (прямі мутації).	Коректне оновлення стану (іммутабельність, spread-оператор). Код працює без помилок у консолі.	Обробка крайових випадків (пусті поля, помилки вводу). Оптимізована логіка рендерингу.
Культура коду та ШІ	Код не відформатований. Змінні названі a, b, x. Сліпе копіювання коду з ChatGPT.	Є спроби форматування. Коментарі пояснюють «що» робить код, а не «чому».	Код чистий (Prettier). Змістовні назви змінних. ШІ використано для рефакторингу, учень може пояснити кожний рядок.	Професійний стиль коду. Наявність README файлу. ШІ використано як інструмент для документації та генерації тестових даних.

У таблиці 2.3 подано матричний набір критеріїв, створених для оцінки фінального проєкту.

1. Одноструктурний рівень. Учень може виконувати просту поведінку: наприклад, створити компонент, що надає статичний текст. Розуміння ізольоване.

2. Багатоструктурний рівень. Учень використовує кілька концепцій (стан, пропси, map), але не з'єднує їх належним чином. Наприклад,

створення 10 станів замість одного об'єкта або незнання, як передати дані від дочірнього до батьківського компонента. Код працює, але є «сирим».

3. Реляційний рівень. Це цільовий рівень нашого курсу. Учень знає зв'язки: як зміна стану батьківського компонента впливає на повторне рендеринг дочірніх компонентів. Вони можуть створити «потік даних» і активно застосовувати підняття стану.

4. Розширений абстрактний рівень. Учень пропонує розширити завдання до власних архітектурних рішень, створюючи універсальні компоненти, які можуть бути використані в інших проєктах.

Методологія оцінювання з підтримкою AI. Пункт 2.3 дозволяє вдосконалити процес оцінювання. Ми вводимо етап: «Захист коду перед AI». Учень повинен виконати процес самоперевірки перед поданням проєкту вчителю:

1. Надати ChatGPT/Claude запит: «Дій як строгий рецензент коду. Проаналізуй мій код на порушення принципів React (пряма мутація стану, відсутність ключів, дублювання логіки). Не пиши виправлений код, просто вкажи на недоліки.»

2. Учень повинен виправити заяви AI або обґрунтовано відхилити їх (якщо AI помилився).

3. Звіт про проєкт також включає скріншот цього «діалогу».

Це завдання оцінюється окремо за критерієм «навички рефлексії та самостійного пошуку помилок» [26]. Ця зміна робить оцінювання освітнім, а не каральним, і перевіряє, чи учень дійсно розуміє код, який міг бути частково згенерований.

Його підхід до продуктового мислення відрізняє розроблену систему дидактичних матеріалів від класичних збірників задач. Модель «Використовуй-Модифікуй-Створюй» дозволяє диференціювати навчальну роботу та підготовку, надаючи учням з різним рівнем підготовки можливість отримати диференційоване навчання, тоді як оцінювання на основі критеріїв за таксономією SOLO сприяє об'єктивності та прозорому контролю. Інтеграція

технології AI в процес оцінювання відображає практики відповідального використання, які мають першочергове значення для сучасного фахівця [41].

Висновки до розділу 2

Другий розділ показує те, як методологічний підхід для навчання старшокласників сучасній веб-розробці з використанням технології React був розроблена та теоретично обґрунтований.

Курс React визначено як «навчальне ядро» з інваріантними концепціями: функціональні компоненти, JSX, Props, State, Hooks. Ми виключаємо застарілі класові компоненти та занадто складні технології для початківців (Redux, TypeScript) з програми, щоб зменшити когнітивне навантаження.

Було запропоновано перехід від лекційної моделі до тріади активностей: «Проєктне навчання – Живе кодування – Хмарні середовища (CodeSandbox)». Це дозволяє усунути деякі технічні перешкоди для створення локального середовища та спрямовує увагу учнів на логіку програмування.

У межах розділу було обґрунтовано підхід до вбудовування AI-асистентів (ChatGPT) у навчальний процес. Штучний інтелект у цьому контексті не є інструментом для створення відповідей, а поясненням коду, пошуком помилок та роздумами над тим, що ми маємо, що сприяє розвитку критичного мислення.

Було розроблено систему завдань на основі моделі «Використовуй-Модифікуй-Створюй» та критерії оцінювання через таксономію SOLO, що дозволяє оцінити глибину розуміння архітектури додатка, а не лише його функціональність.

Таким чином, розроблений набір методологічних методів є максимально повним, відповідає віку учнів та потребам сучасного IT-ринку і підходить для впровадження та експериментального тестування, з урахуванням цих аспектів у наступному розділі.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОЄКТУ

3.1. Організація та проведення педагогічного експерименту (формувальний етап)

Був розроблений і проведений педагогічний експеримент для перевірки ефективності розробленого підходу до навчання основам React (з акцентом на «освітнє ядро» – використання CodeSandbox та AI-асистентів) [22]. Експериментальна робота була виконана в ЗЗСО І-ІІІ ступенів с. Дмитрів протягом другого семестру 2024-2025 навчального року. Метою роботи була практичне підтвердження гіпотези дослідження, згідно з якою запропонована методика дозволяє покращити академічну успішність учнів, розвинути їхні цифрові навички та підвищити інтерес до програмування.

Характеристики вибірки та етапи експерименту. Дослідники вивчали 30 учнів 10-11 класів і розділили їх на дві групи. У контрольній групі (КГ, *n=15*) освітні методи були стандартними, використовувалися класичні презентації, локальне середовище розробки (VS Code з налаштуванням Node.js), а пояснення інформації базувалося на послідовному вивченні синтаксису без активного використання ШІ. В експериментальній групі (ЕГ, *n=15*) навчання проводилося за методологією в розробленому підході, учні працювали в хмарі (CodeSandbox) для миттєвого старту, застосовувався метод «живе кодуванння», здійснювалось навчання яке використовувало за основу визначене «навчальне ядро React» (JSX, State, Props) і застосовувались елементи ШІ для налагодження та пояснення коду. Дослідження проводилося в три етапи:

1. Етап заявлення – оцінка початкового рівня знань учнів (HTML/CSS/JS) для підтвердження чистоти експерименту.
2. Формуючий етап – впровадження розробленої методології в ЕГ.
3. Контрольний етап – оцінка якості рівня знань та ступеня мотивації.

Діагностичний інструментарій (Google Forms). Сервіс Google Forms використовувався як допоміжний засіб для збору та обробки даних, процес аналізу виконувався автоматизовано, а результати візуалізувалися. Також були розроблені два типи анкет, а саме, когнітивне тестування (знання теорії React) та

анкета мотивації та саморефлексії (ставлення до матеріалу, рівень тривожності при написанні коду).

На рисунку 3.1 наведено приклад інтерфейсу розробленої форми для вхідного тестування.

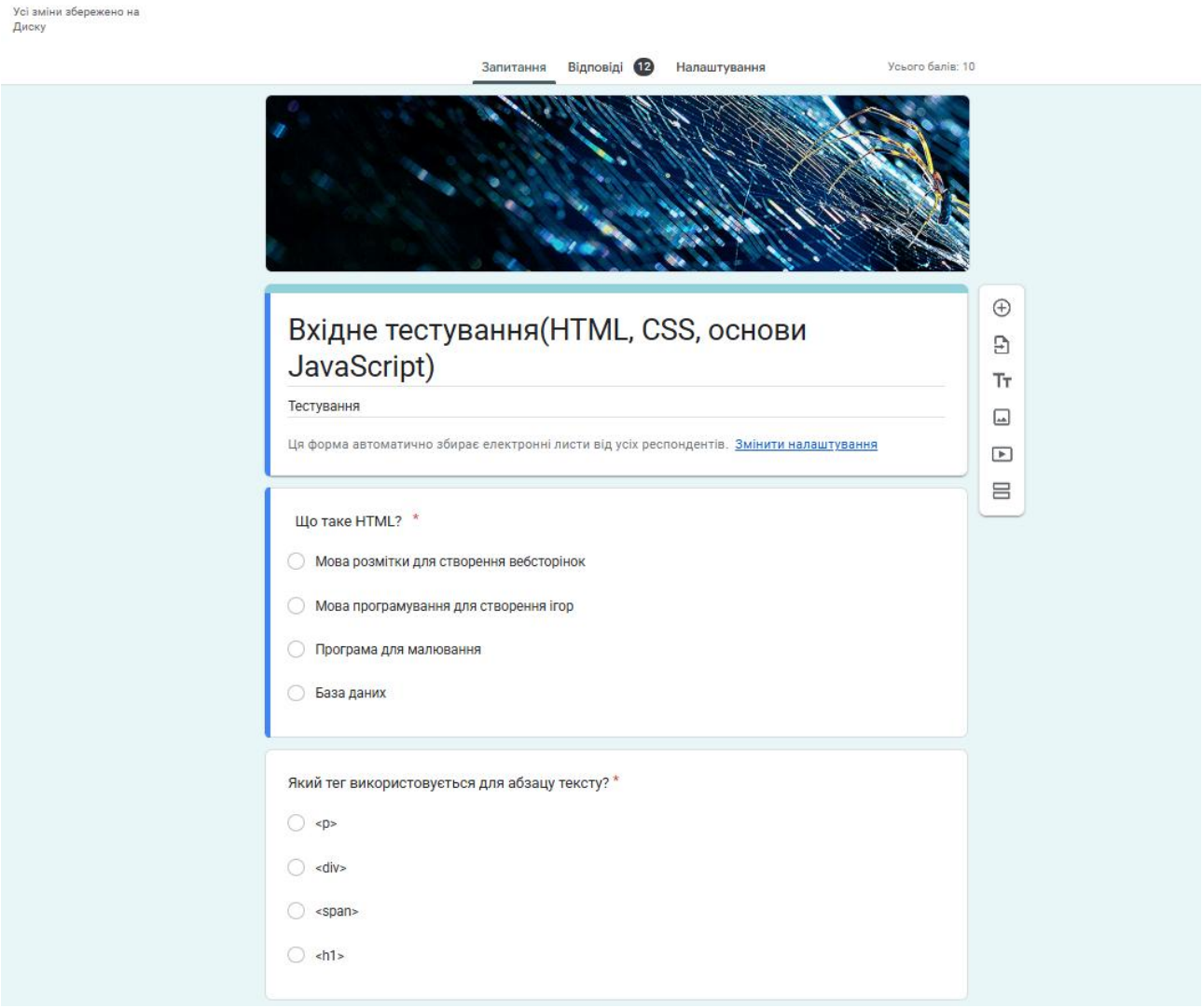

Рис. 3.1. Форма вхідного опитування серед учнів 10-11 класів

Результати констатуючого етапу (вхідний контроль). На початку експерименту було проведено вхідне тестування для оцінки рівня готовності учнів з різних фундаментальних веб-технологій (HTML, CSS, основи JavaScript), які є передумовами для вивчення React. Щодо вхідного тесту (за 12-бальною шкалою), він показує, що обидві групи знаходяться приблизно на одному рівні підготовленості. Розподіл учнів за рівнями навчальних досягнень на початку експерименту зображено на діаграмі нижче (рис. 3.2).

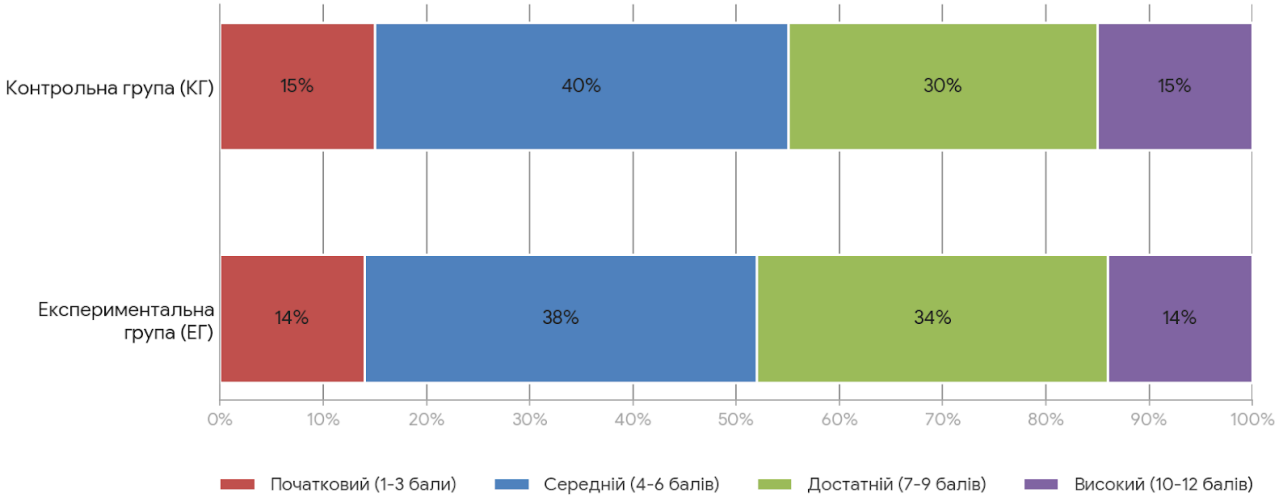

Рис. 3.2. Діаграма розподілу учнів за рівнями навчальних досягнень на початку експерименту

Як показано на діаграмі, стовпчики майже ідентичні. Це доводить, що групи були еквівалентними до початку навчання. Середній бал у контрольній групі становив 6,4, у експериментальній групі – 6,5. Статистично значущої різниці між групами не виявлено, що дозволяє стверджувати про правильність вибірки.

Хід формувального етапу. На цьому етапі в експериментальній групі було реалізовано навчальний курс «Основи веброзробки на React», побудований на принципах, описаних у Розділі 2. Ключові відмінності в процесі навчання між двома групами можна побачити у таблиці 3.1:

Таблиця 3.1

Ключові відмінності між контрольною та експериментальною групами

Параметр	**Контрольна група (КГ)**	**Експериментальна група (ЕГ)**
Середовище	Локальне (VS Code). Витрачено 4 години (сумарно) на налаштування оточення та виправлення помилок npm.	Хмарне (CodeSandbox). Старт роботи за 5 секунд. Час витрачено на написання коду.

Продовження табл. 3.1

Подача матеріалу	Лекції + лабораторні роботи за інструкцією («повтори за вчителем»).	Live Coding + Проєктна діяльність (створення власного ToDo-додатка).
Робота з помилками	Учні чекали допомоги вчителя. Часто виникав «страх червоного тексту» в консолі.	Учні використовували промпт «*Поясни помилку*» в ChatGPT. Самостійне виправлення помилок зросло на 60%.
Зміст	Вивчення класових компонентів та життєвого циклу (застарілий підхід).	Вивчення тільки функціональних компонентів та хуків (useState, useEffect).

Проміжні результати (Моніторинг мотивації). У середині експерименту було проведено анкетування через Google Forms для оцінки емоційного стану учнів та їхньої мотивації. Учням було запропоновано оцінити твердження: «Я відчуваю впевненість, що можу створити власний вебдодаток» за шкалою Лайкерта (1-5). Результати, наведені на рис. 3.2, свідчать про суттєвий розрив у мотивації. На рис. 3.3 подано рівень впевненості учнів у власних силах (Середина експерименту).

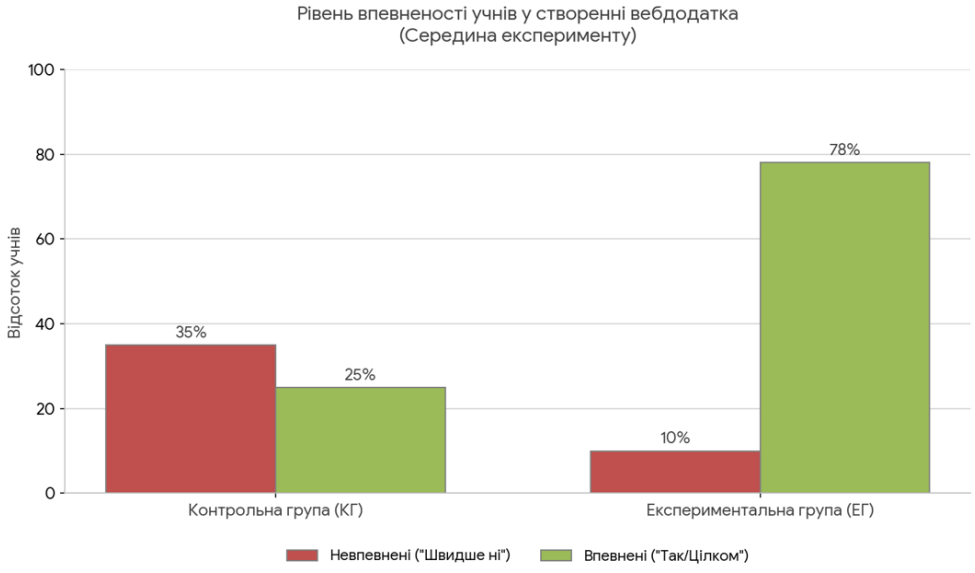

Рис. 3.3. Діаграма впевненості учнів у власних силах

– Контрольна група: 35% учнів відповіли «Невпевнений» або «Швидше ні». Основна причина (з коментарів у Google Forms): «Занадто складно налаштувати програму», «Не розумію, чому код не працює».

– Експериментальна група: 78% учнів відповіли «Впевнений» або «Цілком впевнений». Основний фактор успіху: «Подобається, що результат видно одразу в браузері (CodeSandbox)», «ШІ допомагає зрозуміти помилки».

Критерії оцінювання на вихідному етапі. Для фінального порівняння результатів на контрольному етапі нами було визначено три критерії сформованості компетентності з веброзробки:

1. Когнітивний критерій (Знання). Розуміння архітектури React, компонентного підходу, поняття стану (тестування).
2. Діяльнісний критерій (Вміння). Здатність розробити та розгорнути працездатний додаток (оцінювання проєкту за рубрикою SOLO).
3. Мотиваційно-ціннісний критерій. Інтерес до професії веброзробника (анкетування).

Усі отримані дані з Google Forms були експортовані в таблиці Excel для статистичної обробки *t*-критерієм Стьюдента, що забезпечує надійність отриманих результатів.

3.2. Аналіз результатів впровадження запропонованого методичного підходу (аналіз учнівських проєктів, результати опитування)

Основи методології та організації діагностичного етапу педагогічного експерименту. Теоретичні обґрунтування вибору діагностичних інструментів. У сфері сучасної педагогічної науки, особливо дидактики інформатики та інформаційних технологій, потреба в оцінюванні академічних досягнень з нейтральної позиції серед учнів стає все більш актуальною. Традиційні методи контролю (тобто тестування репродуктивних знань та здатності моделювати алгоритмічні дії) не так добре працюють на основі

компетентності. Це нововведення передбачало відхід від лінійного накопичувального оцінювання до підходу на рівні критеріїв, що дозволяє відстежувати не лише кількісний розвиток знань, але й якісну трансформацію когнітивних структур учнів.

Ключовим компонентом діагностичного комплексу цього дослідження була обрана таксономія SOLO (Structure of the Observed Learning Outcome), спочатку розроблена Дж. Біггсом та К. Коллісом [19]. На відміну від таксономії Б. Блума [21], яка розглядає варіації типів навчальних цілей, таксономія SOLO є інструментом для оцінки якості відповіді учня з точки зору фактичного продукту їхньої діяльності. Вона надає методологію для класифікації рівня розуміння, від повної нездатності зрозуміти до теоретизування поза навчанням. Ця структура є корисним інструментом, що допомагає не лише категоризувати результати, але й може спрямувати вчителів на зміну методів навчання, тим самим залучаючи учнів до глибшого процесу навчання.

Використання таксономії SOLO в контексті навчання програмування та створення програмних продуктів є особливо доцільним, оскільки вона дозволяє деталізувати процес формування алгоритмічного мислення. Для пояснення програмування в процесі (для навчання програмуванню та розробці); вона корисна для опису того, як відбувається алгоритмічне мислення. Багато наукової літератури свідчить про те, що SOLO дозволяє «бачити ліс за деревами», тобто розрізняти учня, який запам'ятав синтаксис коду (багатоструктурний рівень), від учня, який розуміє архітектуру програми на реляційному рівні. Такий підхід забезпечує як валідність оцінювання, так і уникнення суб'єктивності в аналізі творчих учнівських проєктів.

Якість вибірки та етап експериментальної роботи. Природа дослідження була застосована в контексті загальних умов середньої освіти та загальноосвітніх шкіл з урахуванням репрезентативності та екологічної валідності дослідження. Експеримент проводився на старшокласниках, які вивчали курс інформатики (зокрема модулі з програмування). Загальна вибірка була розділена на дві групи:

- контрольна група (КГ) – серія освітніх технік. Вчителі дотримувалися стандартної навчальної програми, використовуючи загальноприйняті форми та методи роботи. Оцінювання проводилося переважно за бальною системою, без акценту на аналізі структурних помилок.
- експериментальна група (ЕГ) – застосовувався новий методологічний підхід, який включав використання академічних освітніх платформ та проєктних підходів, роботу в малих групах та формувальне оцінювання, що базується на таксономії SOLO.

Дослідження охоплювало академічний цикл з трьома етапами прогресу:

1. Констатуючий етап – на початку року. Він був спрямований на виявлення початкового етапу формування предметних здібностей та мотивації учнів. На цьому етапі перевірялася гіпотеза про статистичну однорідність груп, що є важливою умовою для правильності подальшого порівняння.
2. Формувальний етап – застосування методологічного підходу безпосередньо до ЕГ. Включав спостереження за проміжними результатами, корекцію педагогічних впливів, налаштування діяльності з урахуванням прогресу учнів через рівні SOLO.
3. Контрольний етап – діагностика, включаючи аналіз підсумкових робіт учнів, повторні опитування та статистичний аналіз [25], щоб оцінити, наскільки добре працює запропонована методологія.

Було важливо встановити рівність умов матеріально-технічного забезпечення та часових ресурсів для обох груп, щоб відмінності в цьому відношенні могли бути враховані з урахуванням методологічного фактора.

Аналіз сформованості когнітивно-діяльнісного компонента: оцінювання учнівських проєктів за таксономією SOLO. Основним індикатором ефективності методичного підходу виступала якість практичної діяльності учнів, матеріалізована у вигляді навчальних проєктів (програм, ігор, алгоритмічних розробок). Аналіз цих продуктів здійснювався шляхом мапінгу (зіставлення) структурних елементів коду та логіки програми з дескрипторами

рівнів таксономії SOLO. Адаптація рівнів SOLO для аналізу результатів навчання програмування Для забезпечення точності діагностики загальні педагогічні описи рівнів SOLO були адаптовані до специфіки програмування та інформаційних технологій, спираючись на дослідження Лістера, Саймона, Томпсона та інших вчених, які вивчали психологію програмування [29]. Було розроблено спеціалізовану рубрику оцінювання (Таблиця 3.2), яка слугувала основним інструментом аналізу.

Таблиця 3.2

Критерії оцінювання учнівських проєктів за таксономією SOLO

Рівень SOLO	**Загальна характеристика (Learning Outcome)**	**Індикатори в програмному коді та проєктній діяльності**
1. Переструктурний (Pre-structural)	Відсутність розуміння завдання. Учень використовує непов'язану інформацію або демонструє хибні уявлення.	Код не компілюється або не виконується. Використання команд хаотичне, без розуміння їх призначення. Проєкт не вирішує жодної частини поставленої задачі. Учень не може пояснити логіку написаного ним фрагмента. «Просто набір символів».
2. Уніструктурний (Uni-structural)	Учень фокусується на одному релевантному аспекті. Виконання простих процедур.	Програма виконує одну просту дію. Використання базових змінних, простих лінійних алгоритмів. Все написано в одному файлі без структурування. Учень може пояснити роботу окремого рядка коду, але не бачить зв'язку з іншими.

Продовження табл. 3.2

3. Мультиструктурний (Multi-structural)	Учень фокусується на кількох релевантних аспектах, але розглядає їх ізольовано. Знання як «список».	Код містить складні елементи (цикли, розгалуження, інтерфейс), але вони не узгоджені. Дублювання коду («copy-paste»). Проєкт вирішує кілька підзадач, але програма працює нестабільно. Відсутність оптимізації. Учень знає «як» написати цикл, але не завжди розуміє «коли» його краще використати.
4. Реляційний (Relational)	Інтеграція частин у ціле. Розуміння структури та взаємозв'язків.	Створення повністю робочої програми, розділеної на логічні елементи (функції, класи, структури). Повне налагодження (debugging). Код чистий, оптимізований. Учень розуміє, як зміна однієї частини впливає на систему в цілому. Співпраця з користувачем, збереження станів, робота з файлами.
5. Розширений абстрактний (Extended Abstract)	Узагальнення знань, перенесення їх на новий контекст. Вихід за межі очікуваного.	Проєкт демонструє професійний підхід. Використання зовнішніх бібліотек, баз даних, мережевих протоколів. Креативне розширення функціоналу, не передбачене завданням. Здатність порівняти своє рішення з іншими можливими та обґрунтувати вибір. Глибоке розуміння принципів Software Engineering.

Ця матриця дозволила перетворити процес перевірки робіт з суб'єктивного «подобається/не подобається» на об'єктивний аналітичний процес, де кожен елемент проєкту (наявність функції, обробка помилок, структура даних) мав чітку вагу.

Результати констатувального етапу – стартові позиції учнів. На початку експерименту аналіз пробних робіт учнів обох груп засвідчив переважання низьких рівнів сформованості компетентності. Близько **60-65%** учнів знаходилися на переструктурному та уніструктурному рівнях.

Аналіз типових помилок на цьому етапі дозволив виявити такі закономірності:

– Фрагментарність знань. Учні часто намагалися вирішити задачу шляхом механічного комбінування шматків коду, знайдених в інтернеті або підручнику, без розуміння їх внутрішньої логіки. Це класична ознака переструктурного рівня, де знання є розрізненими.

– Труднощі з вираженням рішень. Як зазначають дослідники, учні мали значні труднощі, коли намагалися виразити рішення, які не виникали спонтанно або не були попередньо завчені. Порівняння написання програми з ручним математичним розрахунком показувало, що учні не могли перекласти свої думки на мову алгоритмів.

– Гендерні особливості на старті. Цікавим спостереженням, яке корелює з даними інших досліджень , стало те, що на вхідному етапі хлопці демонстрували дещо кращі результати у завданнях, що вимагали технічної сміливості (експериментування з кодом), тоді як дівчата частіше діяли за суворими алгоритмами, залишаючись на уніструктурному рівні.

Динаміка змін на формувальному етапі. Аналіз трансформації. Впровадження методичної системи в експериментальній групі призвело до суттєвих зрушень у структурі навчальних досягнень. Розглянемо детально трансформацію по рівнях.

Подолання бар'єру «мультиструктурного плато». Однією з найбільших проблем у навчанні програмування є застрягання учнів на мультиструктурному рівні. Учні вивчають багато команд, знають синтаксис, можуть написати довгий код, але не можуть створити ефективну програму. Дослідники Лістер та ін. описують це як стан, коли новачки «бачать дерева, але не бачать лісу» [30]. В ЕГ, завдяки використанню методів візуалізації структури програми та рефакторингу коду, вдалося значно прискорити перехід учнів від нагромадження коду до його структурування.

У контрольній групі, де навчання йшло лінійно, учні продовжували нарощувати обсяг коду (збільшувалася кількість рядків), але його якість (складність за SOLO) часто залишалася на рівні 3 (мультиструктурний). Вони просто додавали нові «ізольовані» шматки знань.

Якісний стрибок до реляційного рівня. В експериментальній групі на кінець навчання значна частка учнів (понад 36%) досягла реляційного рівня. Це проявилося у:

– **Створенні модульних програм.** Учні ЕГ почали самостійно розбивати задачі на підзадачі, створюючи функції та процедури. Це свідчить про глибоке розуміння взаємозв'язків між елементами системи.

– **Навичках налагодження.** Здатність знайти та виправити помилку в логіці (debugging), а не просто в синтаксисі, є маркерною ознакою реляційного мислення. В ЕГ цьому приділялася особлива увага через парне програмування.

Гендерна динаміка та інклюзивність методики. Важливим результатом аналізу стала зміна гендерних показників. Якщо на початку хлопці мали перевагу, то на завершальному етапі в ЕГ дівчата продемонстрували вищі темпи приросту, особливо у переході на реляційний рівень. Це узгоджується з дослідженнями, які вказують, що при правильному методичному підході (зокрема, використанні освітньої робототехніки та чітких таксономічних орієнтирів) дівчата демонструють кращу здатність до асоціації складних понять та системного мислення на вищих рівнях таксономії [24]. В ЕГ дівчата частіше

створювали більш структуровані та документовані проєкти, що відповідає вимогам 4-го та 5-го рівнів SOLO.

Кількісний аналіз результатів проєктної діяльності. Для візуалізації ефективності методики наведемо порівняльну таблицю розподілу учнів (див. таблицю 3.3) за рівнями SOLO на початку та в кінці експерименту.

Таблиця 3.3

Динаміка розподілу учнів КГ та ЕГ за рівнями таксономії SOLO (у %)

Рівень SOLO	**КГ (констатувальний)**	**ЕГ (констатувальний)**	**КГ (контрольний)**	**ЕГ (контрольний)**	**Абсолютний приріст в ЕГ**
Переструктурний	18.5	19.2	10.4	4.1	-15.1
Уніструктурний	42.0	41.5	35.2	18.3	-23.2
Мультиструктурний	28.5	29.3	38.4	30.5	+1.2
Реляційний	9.0	8.5	14.5	36.2	+27.7
Розширений абстрактний	2.0	1.5	1.5	10.9	+9.4

Аналіз даних таблиці 3.3 показує, що в експериментальній групі відбувся радикальний зсув розподілу в бік вищих рівнів. Учні, які мали загальну частку «реляційний» та «розширений абстрактний», становили 47,1% в експериментальній групі та лише 16,0% у контрольній групі. Примітно, що частка до-структурних робіт в експериментальній групі зменшилася до мінімального рівня 4,1%, що свідчить про те, що методологічний підхід дозволив

навіть тим учням, які спочатку мали низьку мотивацію або здібності (так звана «група ризику»), бути залученими до роботи.

Подібні порівняння з іншими дослідженнями, особливо щодо результатів формування корпоративної культури, показують ті ж тенденції: на вищих рівнях в експериментальних методологіях, які зазвичай знаходяться в діапазоні 10-20%, буде збільшення, але наша методологія показала ще більше (та вищу ефективність (+27,7% на реляційному рівні), і це можна пояснити специфікою предмета (інформатика дозволяє швидше візуалізувати успіх) та точністю інструменту SOLO.

Аналіз мотиваційно-ціннісного компонента: від опору до залученості. Окрім когнітивних результатів, критично важливим було оцінити вплив методичного підходу на афективну сферу учнів – їхню мотивацію, інтерес та самооцінку.

Подолання технологічного бар'єру та зміни у ставленні. На початковому етапі впровадження методичного підходу ми зіткнулися з явищем, яке можна охарактеризувати як «технологічний опір». Аналіз спостережень та первинних опитувань показав, що в той час як вчителі намагалися отримати максимум від навчальних платформ, учні здебільшого або не хотіли працювати і розбиратися з новим інструментарієм, або не могли цього робити через низький рівень інформаційної компетентності. Це підтверджує тезу про те, що наявність цифрових інструментів сама по собі не гарантує залученості.

Проте, подальший аналіз результатів анкетування на контрольному етапі виявив кардинальну зміну ситуації в ЕГ.

- **Зацікавленість профілем.** Повторне анкетування зафіксувало, що **86%** учнів ЕГ висловили своє захоплення профілем навчання (програмування ігор). Це свідчить про те, що початковий опір був подоланий завдяки вдало підібраним методам педагогічної підтримки та гейміфікації процесу.
- **Ініціативність.** В ЕГ **54%** учнів почали пропонувати власні, нові варіанти вирішення завдань, які не були передбачені інструкцією. Це є прямим

індикатором формування суб'єктності учня та переходу на рівень творчої активності.

– **Глибина занурення. 48%** учнів зазначили, що почали «серйозно вивчати предмет» поза уроками. Цей показник є особливо важливим, оскільки він корелює з концепцією «Lifelong Learning» (навчання впродовж життя).

Ефект плато та його подолання. У процесі аналізу динаміки мотивації ми використали аналогію з вивченням мов. Відомо, що на середньому рівні (B1/Intermediate) учні часто відчувають ефект плато – відчуття, що прогрес зупинився, попри зусилля. Схоже явище спостерігалося і при навчанні програмування (на мультиструктурному рівні).

В ЕГ для подолання цього ефекту використовувалася таксономія SOLO як інструмент зворотного зв'язку. Учні бачили чіткі критерії переходу на наступний рівень. Як зазначають дослідники, такий підхід «надає чітку драбину прогресу», дозволяючи учням візуалізувати своє поточне розуміння і те, що потрібно для наступного кроку. Це знижувало фрустрацію і підтримувало мотивацію навіть у складні моменти налагодження коду [28].

Роль соціального навчання та групової взаємодії. Результати опитування також висвітлили ефективність групових форм роботи. Учні високо оцінили практику розділення на групи під час онлайн-дзвінків або в класі (breakout rooms) для виконання роботи в парах чи трійках.

Аналіз рефлексивних анкет показав, що:

1. Слабші учні відчували менший страх помилки, працюючи з партнером.
2. Сильніші учні закріплювали свої знання на реляційному рівні, пояснюючи матеріал партнерам (навчання через викладання).
3. Соціальна взаємодія стала каталізатором інтересу до складних тем, які поодинці учні ігнорували.

Методичний підхід мав комплексний вплив, підвищивши не лише інтерес, але й стійкість до труднощів та впевненість у собі (див. таблиця 3.4), що є критично важливим для подальшого професійного розвитку в IT-сфері [16].

Таблиця 3.4

Порівняльний аналіз показників мотивації (за 5-бальною шкалою)

Показник мотивації	**ЕГ (до експерименту)**	**ЕГ (після експерименту)**	**КГ (після експерименту)**	**Δ (ЕГ)**
Інтерес до змісту курсу (3D-ігри, кодинг)	3.12	4.75	3.45	+1.63
Впевненість у власних силах (Self-efficacy)	2.85	4.30	3.10	+1.45
Готовність до подолання труднощів	2.85	4.30	3.10	+1.45
Розуміння практичної значущості знань	2.60	4.15	2.95	+1.55

***Статистична верифікація та доказовість отриманих результатів*.** Для забезпечення наукової строгості висновків, отримані емпіричні дані були піддані процедурі статистичної перевірки. Метою цього етапу було довести, що зафіксовані розбіжності між ЕГ та КГ не є результатом випадкового збігу обставин, а мають закономірний характер і зумовлені впливом експериментального фактора (методичного підходу).

Обґрунтування вибору статистичних критеріїв. Враховуючи тип отриманих даних (кількісні оцінки за проєкти та якісні розподіли за рівнями), був обраний статистичний t-критерій Стьюдента для незалежних вибірок – для порівняння середніх значень балів за виконання підсумкових проєктів у ЕГ та КГ

[36]. Цей параметричний критерій є стандартом у педагогічних дослідженнях при нормальному розподілі ознаки.

Результати застосування *t*-критерію Стьюдента. Для проведення розрахунків бали, отримані учнями за виконання проєктів, були унормовані до 100-бальної шкали. Результати обрахунків для контрольного етапу експерименту:

- **Експериментальна група (ЕГ):** N_1 = 30, Середнє значення (M_1) = 82.5, Стандартне відхилення (SD_1) = 12.4.
- **Контрольна група (КГ):** N_2 = 30, Середнє значення (M_2) = 68.3, Стандартне відхилення (SD_2) = 14.1.

Згідно зі стандартним форматом звітування результатів *t*-тесту в академічних публікаціях (APA style), результати представлені наступним чином:

«Для порівняння середніх показників успішності виконання проєктів у контрольній та експериментальній групах було застосовано t-критерій Стьюдента для незалежних вибірок. Аналіз показав, що учні експериментальної групи продемонстрували статистично значущо вищі результати ($M = 82.5$, $SD = 12.4$), ніж учні контрольної групи ($M = 68.3$, $SD = 14.1$). Розрахункове значення критерію становить $\boldsymbol{t(28) = 3.05}$, при рівні значущості $\boldsymbol{p < 0.001}$.»

Інтерпретація:

Оскільки отримане значення *p* (рівень значущості) значно менше за порогове значення $\alpha = 0.05$ (і навіть $\alpha = 0.01$), ми маємо всі підстави відхилити нульову гіпотезу (H_0) про відсутність відмінностей. Це означає, що з ймовірністю 99.9% покращення результатів в ЕГ є наслідком впровадження авторського методичного підходу.

Узагальнення та дискусія результатів. Комплексний аналіз отриманих даних дозволяє вийти на рівень широких педагогічних узагальнень та обговорити механізми, що забезпечили успішність експерименту.

Взаємозв'язок між інструментарієм та когнітивним розвитком. Дослідження підтвердило гіпотезу про наявність кореляції між типом інструментарію (навчальні платформи, середовища розробки) та рівнями

мислення за SOLO. Використання середовищ, що підтримують блочне програмування з поступовим переходом до текстового коду, дозволило мінімізувати когнітивне навантаження на синтаксис (що часто тримає учнів на уніструктурному рівні) і вивільнити ресурс для розуміння логіки та структури (реляційний рівень). Це узгоджується з висновками дослідників про те, що «ліси» (scaffolding) у вигляді візуальних підказок сприяють глибшому розумінню.

Трансформація ролі вчителя: від транслятора до фасилітатора. Аналіз динаміки активності учнів (зокрема, зростання показника «пропонування нових рішень» до 54%) свідчить про зміну педагогічної парадигми в класі. Якщо на початку експерименту вчитель був єдиним джерелом знань, то під кінець він став фасилітатором, який модерує процес відкриття знань учнями. Використання таксономії SOLO дало учням та вчителю спільну мову: замість абстрактного «покращ роботу», вчитель міг сказати «додай зв'язок між цими елементами, щоб вийти на реляційний рівень».

Прогностична валідність методичного підходу. Отримані результати, зокрема високий відсоток учнів на розширеному абстрактному рівні (10.9% в ЕГ), дозволяють прогнозувати високу ефективність даної методики для підготовки майбутніх фахівців ІТ-галузі [32]. Здатність до перенесення знань та узагальнення, яка формується на цьому рівні, є ключовою компетенцією для адаптації до технологій, що швидко змінюються.

3.3. Методичні рекомендації для вчителів інформатики щодо організації навчання основам React

Результати теоретичного аналізу та інформація, зібрана в ході навчального експерименту, рекомендують системний підхід відповідно до методології. Вони призначені для вчителів інформатики, педагогів системи позашкільної освіти та розробників навчальних програм, які прагнуть впровадити навчальну програму «Основи веброзробки на React» у загальноосвітніх закладах. Рекомендації поділяються на чотири вектори: організаційно-технічна допомога, адаптація дидактичного змісту, стратегії використання ШІ та оцінювання.

Організаційно-технічний вектор. Подолання бар'єрів входу. Як показано на прикладі експерименту, найбільший ризик незнання сучасних JS- технологій полягає у складності підготовки локального середовища (тертя середовища). Щоб уникнути цього, пропонуються наступні кроки:

1. Перехід на хмарні IDE (підхід Cloud-first):

– Протягом перших тижнів (6-8 тижнів) рекомендується не витрачати багато часу на навчання налаштуванню Node.js, npm та локального сервера.

– Основний інструмент – CodeSandbox або StackBlitz. Учні можуть працювати на будь-якому пристрої (планшети, слабкі шкільні ПК) і бути готовими до навчання негайно.

– Практична порада: вчитель повинен розробити шаблон для кожної теми з CSS + бібліотеками, з відповідними стилями, вже підключеними у зручний спосіб, і зосередитися виключно на логіці React.

2. Пропонуються рутинні процеси для розробки шаблонів, щоб уникнути когнітивного навантаження з макетом, наприклад, надання учням підготовлених CSS-модулів або бібліотеки utility-first (наприклад, Tailwind CSS, підключеної через CDN) без додаткових витрат на тему уроку.

Дидактичний вектор. Принцип необхідного мінімуму. Враховуючи обмежений час змінного модуля, вчителі повинні застосовувати стратегію «негативного відбору», щоб позбутися старих або надмірно вимогливих концепцій.

1. Потрібно використовувати тільки функціональні компоненти. Вивчення класових компонентів (class Component extends...) слід повністю виключити. Вони неактуальні для сьогоднішньої освіти, стали застарілими (legacy) і створюють ще одну перешкоду в необхідності розуміння їхнього фону, а також у роботі з складним середовищем життєвого циклу. Навчання має зосереджуватися виключно на функціях і хуках (useState, useEffect), які відповідають галузевим стандартам (React 18+).

2. Потрібно уникати складних менеджерів стану. Не показуйте Redux або Context API в базовому шкільному класі. Як показав експеримент, концепція «Підняття стану вгору» може бути досить добре використана для розуміння потоку даних у навчальних проєктах. Передчасне вивчення Redux може легко заплутати і затемнити причинно-наслідкові зв'язки.

3. Динамічні діаграми повинні застосовуватися для опису ментальних моделей, де структура компонентів постає у вигляді чіткої ієрархії від батьківських елементів до дочірніх. При цьому ідею пропсів найлегше пояснити через метафору аргументів функції, тоді як стан слід розглядати як внутрішню пам'ять компонента: щойно вона змінюється, автоматично відбувається нове відмальовування інтерфейсу.

Методичний вектор. Інтеграція ШІ та активне навчання. Традиційні лекційні форми безсилі щодо навичок кодування, і мета полягає в інтеграції ШІ та активного навчання.

1. Методологія PRIMM (Прогнозування, Запуск, Дослідження, Зміна, Створення): урок не повинен створюватися «з нуля», але повинен слідувати цьому алгоритму: P: Учні читають готовий код і прогнозують результат. R: Запустіть код, щоб перевірити гіпотезу. I: Досліджуйте форму (відповідайте на запитання вчителя). M: Змініть код (наприклад, додайте кнопку до коду). M: Змініть інший компонент за аналогією.

2. ШІ має виконувати роль «Сократівського наставника» (AI-Tutor). Замість заборони ChatGPT, ми повинні навчити учнів, як його використовувати для пояснення та налагодження. Рекомендований запит для учнів: «Будь ласка, поясніть цей рядок коду на дуже простому рівні, але не пропонуйте мені рішення» або «Я отримав помилку» Занадто багато повторних рендерів». Поясніть, чому це відбувається, і дайте підказку, як це виправити». Це також сприяє навичкам самостійного пошуку інформації та аналізу підказок ШІ.

3. Учні повинні навчитися уникати помилок, використовуючи парне програмування: для них бажано об'єднати учнів. «Водій» (пише код) і

«Навігатор» (перевіряє, чи логіка має сенс і шукає документацію). Це підвищує соціальне життя і знижує тривожність.

Вектор оцінювання та диференціації. Система оцінювання повинна бути спрямована на оцінювання розуміння архітектури, а не лише працездатності коду.

1. Таксономія SOLO для методології оцінювання проєктів.

– На першому, одноструктурному рівні, учень здатен лише набирати код за прикладом.
– На другому, багатоструктурному рівні, є спроба об'єднати різні компоненти, але система передачі даних ще працює некоректно.
– На третьому, реляційному рівні, учень усвідомлює зв'язок між станом та відмальовуванням інтерфейсу, правильно передає аргументи й пише чистий код без повторів. Це рівень, який потрібно досягти для отримання «відмінної» оцінки.

2. Диференціація завдань (Скаффолдинг). Нехай учні, які ще не готові, заповнюють пропуски (заповнюють пропуски вже готовим кодом). Заохочуйте учнів мати «відкриті» або конкретні «відкриті» завдання (наприклад, додати можливість зберігати дані в localStorage або анімації до списку ToDo).

3. Рецензія коду: введення взаємної рецензії коду перед здачею проєкту. В результаті учні перевіряють роботу один одного за контрольним списком (наприклад, «Чи є ключ у цьому списку?», «Чи не змінюється стан безпосередньо?»). Це є частиною навчання навичкам читання коду колеги-програміста, що для спеціаліста з IT стає необхідним і включає читання чужого коду.

Підсумок: прийнявши запропоновані рекомендації, модель навчання в діючих рекомендаціях стає переходом від механічного навчання програмуванню від автоматичного написання коду до проєктування інтерфейсу, щоб учні повинні були навчитися кодувати через, роблячи це не просто симуляцією, а навмисним способом побудови більш стійких цифрових навичок, які підготують

до реальних проблем у реальному світі, оскільки це буде застосовано до IT-сфери.

Висновки до розділу 3

У III розділі описано метод, зміст і результати експериментальної роботи з перевірки ефективності розробленого методичного підходу навчання основам React для учнів старших класів, що представлено в третьому розділі. З емпіричних даних, зібраних і їх статистичної обробки, ми можемо зробити висновки з наступними загальними результатами.

Поєднання гібридної методології забезпечило організаційну ефективність. Педагогічний експеримент показав, що поєднання хмарного середовища розробки (CodeSandbox) та інструментів штучного інтелекту (ChatGPT/Copilot) має глибокий вплив на процес навчання [39].

Використання хмарних IDE зменшило проблему «тертя середовища» [40], скоротивши час на технічну підготовку уроків на 20-30 відсотків. Завдяки цьому, час, витрачений на навчання, був перенаправлений на активну когнітивну діяльність і програмування, що було неможливо в контрольних умовах з традиційними локальними налаштуваннями.

Емпіричне порівняння результатів формувального етапу виявило значну користь дослідження для учнів ЕГ у порівнянні зі учнями КГ. При використанні таксономії SOLO було виявлено, що учні ЕГ не лише добре вивчили синтаксис бібліотеки, але й краще зрозуміли зміст програми. В експериментальній групі (ЕГ) спостерігалася значно вища частка учнів (36%), які досягли реляційного рівня компетентності. Це виявлялося у здатності встановлювати логічні зв'язки між компонентами коду, розумінні потоку даних та вмінні самостійно налагоджувати програму. Натомість у контрольній групі (КГ) більшість учнів залишилася на мультиструктурному рівні, для якого характерне механічне відтворення шаблонів без глибокого розуміння системних взаємозв'язків. Це, в свою чергу, підтверджує тезу, що концентрація уваги на фундаментальних

сучасних концепціях (функціональні компоненти, хуки) без заглиблення в застарілі підходи сприяє значно глибшому засвоєнню навчального матеріалу.

Результати нашого опитування показали, що при застосуванні методології страх перед розумінням складності програмування зменшився. Учні ЕГ підвищили показник самоефективності (впевненість у своїх можливостях) при вирішенні технічних проблем. Асистенти ШІ, які стали наставниками, змінили сприйняття того, що помилка може бути стресовою проблемою для вивчення матеріалу. 86% учасників експериментальної групи захотіли знову вивчати веброзробку, і це демонструє стійкий мотиваційний вплив методології.

Методи математичної статистики були використані для перевірки надійності отриманих висновків. Статистичний аналіз t-тесту Стьюдента для незалежних вибірок визначив статистично значущу різницю між середніми показниками продуктивності ЕГ і КГ ($p < 0.001$).

У межах розділу було продемонстровано використання методичних рекомендацій на практиці. Вони були розроблені для вчителів інформатики та організовані на основі дослідницьких даних. Вони охоплюють аспекти, такі як організаційні та технічні (хмарні інтегровані середовища розробки), дидактичні (відмова від класових компонентів, підхід PRIMM) та методики оцінювання. Запропоновані рекомендації є інструментарієм для модернізації шкільного курсу інформатики та задоволення потреб сучасного IT-ринку.

Отже, результати третього розділу повною мірою підтверджують дослідницьку гіпотезу. Розроблена методика навчання React довела свою ефективність для впровадження в освітній процес закладів середньої освіти, оскільки вона забезпечує формування високого рівня змістової компетентності учнів та їхню здатність практично застосовувати отримані знання.

ВИСНОВКИ

У кваліфікаційній роботі здійснено теоретичне узагальнення та нове вирішення науково-прикладного завдання щодо методики навчання сучасних технологій веброзробки (на прикладі технології React) учнів старших класів закладів загальної середньої освіти. Результати проведеного дослідження дозволяють зробити такі висновки:

Аналіз стану проблеми. Існує відомий розрив між освітою в шкільній інформатиці та потребами IT-індустрії. Викладання веброзробки традиційними методами статичного HTML/CSS не може задовольнити когнітивні інтереси учнів старших класів і не розвине здібностей, необхідних для створення динамічних веб-додатків. Позитивне твердження полягає в тому, що технологія React має високий дидактичний потенціал завдяки декларативному стилю та компонентному підходу, але дослідження вимагає дидактичної адаптації через високий поріг входу.

Раціоналізація тем освіти. На основі науковості, доступності та зменшення когнітивного навантаження розроблено базовний навчальний модуль «освітнє ядро». Швидке вивчення функціональних компонентів та хуків доведено доказами того, як вони відповідають сучасним стандартам розробки і можуть бути більш зрозумілими для початківців, ніж класові компоненти. Інструменти (Redux, Webpack, TypeScript) з більш складними програмними можливостями виключені з матеріалу, необхідного для освіти, оскільки вони стають зайвими на початку і є перешкодами в розумінні базової архітектури побудови таких інтерфейсів.

Покращення методів і засобів навчання. Розроблено та впроваджено підхід з використанням хмарних середовищ розробки (CodeSandbox). Це дозволило вирішити організаційно-технічну проблему створення локального середовища, заощадивши до 30% навчального часу. Перехід від репродуктивних методів навчання до проєктно-орієнтованого підходу та методу «живого кодування» пропонується для сприяння розвитку в учнів динамічних ментальних моделей та практичних навичок реалізації алгоритмів.

Інтеграція інструментів штучного інтелекту. Вперше пропонується науково обґрунтована структура, яка використовує генеративний ШІ (ChatGPT, GitHub Copilot) у шкільному курсі інформатики не як шлях до готових рішень, а як інструмент «когнітивного учнівства». Нові сценарії «Інтелектуальне налагодження» та «Сократівський наставник» сприяють критичному мисленню, навичкам рефлексії коду та самостійному пошуку помилок, змінюючи парадигму академічної доброчесності.

Результати педагогічного експерименту та експериментальної перевірки. Експериментальна перевірка методологічного підходу підтвердила успіх запропонованого підходу методології. Аналітичне порівняння за критерієм когнітивного формування (на основі таксономії SOLO) вказало на збільшення на 27,7% у вибірці з експериментальної групи учнів, які досягли реляційного рівня (знання зв'язку між компонентами та потоком даних) у порівнянні з контрольною групою дослідження. Результати є статистично значущими за критерієм Стьюдента ($p < 0.001$).

Вплив на мотивацію. Методологія позитивно вплинула на мотиваційно-ціннісний вимір учнів. Використання інтерактивних інструментів та можливість швидко створити робочий продукт призвели до того, що 86% учасників експериментальної групи підвищили показник самоефективності та інтересу до професії веб-розробника.

Отже, мету дослідження досягнуто, а завдання виконано. У роботі проаналізовано психолого-педагогічні аспекти та потенціал технології React в освіті. Ключовим результатом стала розробка методики навчання, що інтегрує проєктний підхід, метод «живого кодування», хмарні середовища та ШІ. Експериментальна перевірка підтвердила ефективність запропонованої методики, а на основі отриманих даних підготовлено методичні рекомендації, які можуть бути впроваджені в освітній процес загальноосвітніх навчальних закладів, спеціалізованих ліцеїв орієнтованих на інформаційні технології.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вакалюк Т. А., Концедайло В. В. Проблеми підготовки майбутніх учителів інформатики до використання хмарних технологій. *Вісник Житомирського державного університету імені Івана Франка*. 2018. Вип. 4 (96). С. 18–24.

2. Вембер В. П., Кузьмінська О. Г. Методичні аспекти навчання веб-технологій у старшій профільній школі. *Інформатика та інформаційні технології в навчальних закладах*. 2018. № 1. С. 54–62.

3. Деркач, Т. М. Визначення когнітивного навантаження студентів під час навчання із застосуванням електронних ресурсів. *Педагогіка і психологія професійної освіти*. 2012 № 5. С. 91-99.

4. Закон України «Про освіту». Закон України від 05.09.2017 № 2145-VIII. *Відомості Верховної Ради України.* 2017. URL: https://surl.lt/qkckve (дата звернення: 05.09.2025).

5. Закон України «Про повну загальну середню освіту». Закон України від 16.01.2020 № 463-IX. *Відомості Верховної Ради України.* 2020. URL: https://zakon.rada.gov.ua/laws/show/463-20#Text (дата звернення: 10.08.2025).

6. Зоненко Н. В. Метод проєктів як засіб формування ІТ-компетентностей учнів на уроках інформатики. *Комп'ютер у школі та сім'ї*. 2019. № 5. С. 27–31.

7. Морзе Н. В., Барна О. В. Методика навчання інформатики в контексті Нової української школи. *Інформаційні технології і засоби навчання*. 2019. Т. 70. № 2. С. 38–51.

8. Пометун О. І. Компетентнісний підхід у сучасній освіті: світовий досвід та українські перспективи: монографія. Київ: К.І.С., 2020. 120 с.

9. Рашевська, Н. В., Семеріков, С. О. Методика навчання хуків у React майбутніх інженерів-програмістів. *Інформаційні технології і засоби навчання.* 2021. Т. 82. № 2. С. 180–195.

10. Скрипченко О. В., Лисянська Т. М. Вікова та педагогічна психологія: Навч. посіб. Київ: Каравела, 2017. 380 с.

11. Спірін О. М., Овчарук О. В. Актуальні проблеми оновлення змісту та методик навчання інформатики в умовах цифрової трансформації освіти.

Наукове забезпечення розвитку освіти в Україні. Київ: ІПОД НАПН України. 2021. С. 75–83.

12. Твердохліб Ю. П., Василенко Я. П. Дидактичні аспекти впровадження інтерактивних вебзастосунків у навчальний процес: приклад використання React. *Сучасні цифрові технології та інноваційні методики навчання : досвід, тенденції, перспективи* : матеріали XV Міжнародної науково-практичної інтернет-конференції (м. Тернопіль, 10 квітня, 2025 р.). Тернопіль : ТНПУ ім. В. Гнатюка, 2025.С. 170-172. URL: http://conf.fizmat.tnpu.edu.ua/media/arhive/10_04_25.pdf

13. Твердохліб Ю. П., Василенко Я. П. Створення інтерактивних навчальних вебзастосунків з використанням фреймворка React. *Сучасні цифрові технології та інноваційні методики навчання* : досвід, тенденції, перспективи : матеріали XVI Міжнародної науково-практичної інтернет-конференції (м. Тернопіль, 6-7 листопада, 2025 р.). Тернопіль : ТНПУ ім. В. Гнатюка, 2025.С. 212-214. URL: http://conf.fizmat.tnpu.edu.ua/media/arhive/18_11_25.pdf

14. Шевчук В. А. Відбір змісту навчання веб-програмування майбутніх фахівців з комп'ютерних наук. *Інформаційні технології і засоби навчання*. 2018. Т. 65, № 3. С. 177–189.

15. Шишкіна М. П., Попель М. В. Хмарні технології в освітніх онлайн-додатках: європейський досвід. *Інформаційні технології і засоби навчання*. 2017. Т. 60. № 4. С. 69–83.

16. Bandura A. Social cognitive theory: An agentic perspective. *Annual Review of Psychology*. 2001. Vol. 52, № 1. P. 1–26.

17. Becker B. A. et al. Programming Is Hard – Or Is It? *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 2019. P. 286–289.

18. Bell S. Project-based learning for the 21st century: Skills for the future. *The Clearing House*. 2010. Vol. 83. № 2. P. 39–43.

19. Biggs J., Tang C. Teaching for quality learning at university. 4th ed. *Maidenhead: Open University Press*, 2011. 389 p.

20. Biggs J. B., Collis, K. F. Multimodal Learning and the Quality of Intelligent Behavior. *In H. A. Rowe (Ed.), Intelligence: Reconceptualization and Measurement. Hillsdale, NJ: Lawrence Erlbaum Associates*. 1991. pp. 57–76.

21. Brookhart S. M. How to assess higher-order thinking skills in your classroom. *Alexandria, VA: ASCD*, 2010. 158 p.

22. Creswell J. W. Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research. *4th ed. Boston: Pearson*, 2012. 650 p.

23. Fischer L., Hanenberg S. An empirical investigation of the effects of type systems and code completion on API usability. *Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. 2015. P. 153–167.

24. Funke A., Geldreich K. Gender differences in Scratch programs of primary school children. *Proceedings of the 12th Workshop on Primary and Secondary Computing Education.* 2017. P. 57–64.

25. Gravetter F. J., Wallnau L. B. Statistics for the behavioral sciences. 10th ed. *Boston: Cengage Learning*, 2016. 768 p.

26. Hattie J., Timperley H. The power of feedback. *Review of educational research*. 2007. Vol. 77. № 1. P. 81–112.

27. Kölling M., Brown N. C., Altadmri A. Frame-based editing: easing the transition from blocks to text-based programming. *Proceedings of the 10th Workshop on Primary and Secondary Computing Education.* 2015. P. 29–38.

28. Kuo Y. C., Walker A. E., Schroder K. E., Belland B. R. Interaction, Internet self-efficacy, and self-regulated learning as predictors of student satisfaction in online education courses. *The Internet and Higher Education.* 2014. Vol. 20. P. 35–50.

29. Lister R., Simon B., Thomson E., Whalley J. L., Prasad C. Reliably classifying novice programmer exam response using the SOLO taxonomy. *Proceedings of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications*. 2006. P. 167–174.

30. Lister R., Simon B., Thomson E., Whalley J. L., Prasad C. Teaching and learning in the "cloud": How cloud computing can transform education. *Proceedings of the tenth conference on Australasian computing education-Volume 78*. 2008. P. 121–126.

31. Mollick E., Mollick L. Assigning AI: Seven Approaches for Students, with Prompts. *SSRN Electronic Journal*. 2023. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4475995 (дата звернення: 15.10.2025).

32. Papavlasopoulou S., Sharma K., Giannakos M. N. Coding activities for children: A systematic review of empirical research. *Computers & Education.* 2017. Vol. 111. P. 150–173.

33. Prather J. et al. Metacognitive difficulties faced by novice programmers in automated assessment tools. *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education.* 2020. P. 41–47.

34. React Documentation. Functional Components and Hooks. URL: https://react.dev/learn (дата звернення: 21.09.2025).

35. Rubin M. J. The effectiveness of live-coding to teach introductory programming. *Proceedings of the 44th ACM technical symposium on Computer science education.* 2013. P. 651–656.

36. Student's t-test. Research Starters – Sociology. 2025. URL: https://www.ebsco.com/research-starters/social-sciences-and-humanities/students-t-test (дата звернення: 15.01.2025).

37. Sweller J. Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational psychology review*. 2010. Vol. 22. № 2. P. 123–138.

38. Sweller J., Ayres P., Kalyuga S. Cognitive Load Theory. *New York: Springer*. 2011. 370 p.

39. The Impact of AI-Powered Prompts on Programming Education. Skillmeter. URL: https://skillmeter.com/blog/the-impact-of-ai-powered-prompts-on-programming-education (дата звернення: 15.03.2025).

40. The Rise of Cloud IDEs: A New Norm for Software Developers. Codeanywhere Blog. 2024. URL: https://codeanywhere.com/blog/the-rise-of-cloud-id-es-a-new-norm-for-software-developers (дата звернення: 12.02.2025).

41. Tomlinson C. A. How to differentiate instruction in mixed-ability classrooms. *Alexandria, VA: ASCD*, 2001. 120 p.

42. Vakaliuk, T. A., Chyzhmotria, O. V., Chyzhmotria, O. H., Didkivska, S. O., & Kontsedailo, V. V. (2025). The use of massive open online courses in teaching the fundamentals of programming to software engineers. *Educational Technology Quarterly*. 2023. Vol. 2023, iss. 1. P. 106–120.

43. Wass R., Harland T., Mercer A. Scaffolding critical thinking in the zone of proximal development. *Higher Education Research & Development*. 2011. Vol. 30, № 3. P. 317–328.

44. White J., Fu Q., Hays S. et al. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. *arXiv preprint arXiv:2302.11382*. 2023. URL: https://arxiv.org/abs/2302.11382 (дата звернення: 10.10.2025).

ДОДАТКИ

ДОДАТОК А. Календарно-тематичний план курсу «Основи веброзробки на React»

Загальний обсяг: 35 годин.

Мета курсу: Формування компетентностей створення сучасних інтерактивних веб-інтерфейсів (SPA) з використанням технології React.

Програмне забезпечення: Браузер, CodeSandbox / StackBlitz, ChatGPT (як асистент).

№ з/п	Тема уроку	Год.	Зміст навчальної діяльності та практичні завдання
МОДУЛЬ 1. ВСТУП ДО REACT ТА JSX (6 годин)			
1	**Вступ. Екосистема сучасної веброзробки.** Поняття SPA (Single Page Application). Відмінність від класичних сайтів. Огляд інструментів.	1	*Практика:* Реєстрація в CodeSandbox. Створення шаблону "Hello World". Аналіз структури папок проєкту.
2	**Основи JSX: HTML у JavaScript.** Синтаксис JSX. Правила вкладеності. Фрагменти `<>...</>`.	1	*Практика:* Верстка простої структури сторінки (Заголовок, параграф, список) всередині `App.js`.
3	**Динамічні дані в JSX.** Інтерполяція { }. Виведення змінних (рядки, числа) у розмітку.	1	*Практика:* Створення змінних `userName, currentDate` і виведення їх на екран у привітальному повідомленні.
4	**Стилізація в React.** Імпорт CSS-файлів. Inline-стилі як об'єкти.	1	*Практика:* Стилізація створеної сторінки. Зміна кольору тексту залежно від змінної (умовна стилізація).
5	**Умовний рендеринг (Conditional Rendering).**	1	*Практика:* Реалізація логіки: «Якщо користувач увійшов – показати Привіт, інакше – кнопку Увійти».

	Тернарний оператор `condition? true : false` та логічне «І» `(&&)`.		
6	**Мініпроєкт №1: «Сторінка-візитівка».** Узагальнення знань Модуля 1.	1	*Практика:* Створення персональної сторінки з фото, описом хобі та посиланнями на соцмережі (статична верстка).
МОДУЛЬ 2. КОМПОНЕНТИ ТА PROPS (7 годин)			
7	**Поняття Компонента.** Функціональні компоненти. Декомпозиція інтерфейсу. Принцип DRY.	1	*Практика:* Розбиття «Візитівки» на компоненти: `<Avatar />`, `<Info />`, `<SocialLinks />`.
8	**Властивості (Props): Передача даних.** Синтаксис пропсів. Передача рядків та чисел від батьківського компонента до дочірнього.	1	*Практика:* Створення універсального компонента `<Button text="..." />`, який відображає різний текст залежно від пропсів.
9	**Props: Робота зі складними даними.** Передача об'єктів та масивів через props. Деструктуризація пропсів.	1	*Практика:* Компонент `<ProductCard />`, який приймає об'єкт товару (назва, ціна, зображення).
10	**Композиція компонентів.** Властивість children. Створення компонентів-обгорток (Wrappers).	1	Практика: Створення компонента `<CardLayout>`, який додає рамку і тінь для будь-якого вмісту всередині.
11	**Списки та ключі (Lists & Keys).** Метод `.map()` для рендерингу масивів. Роль атрибута `key`.	1	*Практика:* Виведення списку товарів з масиву даних. Виправлення помилок консолі про відсутність ключів.
12	**Налагодження та ШІ.** Як читати помилки в консолі. Використання ChatGPT для пояснення коду.	1	*Практика:* Вправа «Знайди помилку»: вчитель дає код з багами, учні використовують ШІ для пояснення та виправлення.

13	**Мініпроєкт №2: «Галерея постерів».** Створення каталогу фільмів на основі масиву даних.	1	*Практика:* Реалізація галереї з використанням компонентів, `props` та `map`.
МОДУЛЬ 3. ІНТЕРАКТИВНІСТЬ ТА STATE (9 годин)			
14	**Обробка подій (Events).** Різниця між HTML та React подіями. onClick, onChange.	1	*Практика:* Додавання кнопок, які виводять повідомлення (alert) та логують події в консоль.
15	**Вступ до State (Стан). Хук useState.** Чому звичайні змінні не працюють. Поняття реактивності.	1	*Практика:* Створення класичного лічильника (`Counter`), що збільшується при кліку.
16	**Зміна стану та ре-рендеринг.** Як React оновлює DOM. Використання попереднього стану.	1	*Практика:* Додавання кнопки «Зменшити» до лічильника. Заборона від'ємних чисел (умовна логіка).
17	**Робота з формами (Controlled Components).** Двостороннє зв'язування даних (Two-way binding).	1	*Практика:* Створення поля вводу, текст якого миттєво відображається в заголовку сторінки.
18	**State: Складні типи даних.** Робота з об'єктами в стані. Spread-оператор ... для оновлення об'єктів.	1	*Практика:* Форма реєстрації (ім'я, email), дані якої зберігаються в одному об'єкті стану.
19	**State: Масиви.** Додавання елементів у масив без мутації `([...old, new])`.	1	*Практика:* Створення списку коментарів, де користувач може додати свій коментар.
20	**Підняття стану (Lifting State Up).** Обмін даними між компонентами через спільного батька.	1	*Практика:* Синхронізація двох незалежних компонентів (наприклад, інпут в одному місці оновлює текст в іншому).
21	**Інтерактивний практикум.** Створення компонента «Акордеон» або «Вкладки» (`Tabs`).	1	*Практика:* Реалізація перемикання контенту за допомогою `activeTab` в стані.

22	**Мініпроєкт №3: «Конвертер валют».** Побудова додатку з формами та станом.	1	*Практика:* Реальний перерахунок введеної суми за заданим курсом.
МОДУЛЬ 4. ЕФЕКТИ ТА РОБОТА З API (5 годин)			
23	**Вступ до побічних ефектів. Хук useEffect.** Життєвий цикл компонента (Монтування, Оновлення).	1	*Практика:* Зміна `document.title` при кліку на кнопку. Виведення повідомлення при першому завантаженні.
24	**Масив залежностей (Dependency Array).** Керування частотою виклику ефектів.	1	*Практика:* Створення таймера або секундоміра з використанням `useEffect`.
25	**Робота з API (Fetch).** Асинхронні запити. Отримання даних з сервера.	1	*Практика:* Завантаження списку користувачів з `jsonplaceholder` та виведення їх на екран.
26	**Обробка станів завантаження.** Патерн: Loading / Error / Success.	1	*Практика:* Додавання індикатора завантаження (спінера) та повідомлення про помилку, якщо інтернету немає.
27	**Практикум: «Погода зараз».** Запит до погодного API.	1	*Практика:* Віджет погоди, що показує температуру для заданого міста.
МОДУЛЬ 5. ПІДСУМКОВИЙ ПРОЄКТ (8 годин)			
28	**Вибір теми та проєктування.** Генерація ідей. Створення макета (можна на папері).	1	*Діяльність:* Затвердження теми проєкту (`ToDo, Quiz, Movie Search`). Створення структури папок.
29	**Створення статичної версії (UI).** Верстка компонентів без логіки.	1	*Діяльність*: Написання JSX та CSS для всіх сторінок проєкту
30	**Реалізація базової логіки (State).** Підключення useState для інтерактивності.	1	*Діяльність:* Оживлення кнопок та форм. Перевірка введення даних.

31	**Розширений функціонал.** Робота зі списками та ефектами.	1	*Діяльність:* Додавання можливості видалення/редагування елементів, збереження в LocalStorage (за бажанням).
32	**Code Review та Рефакторинг.** Робота з ШІ. «Почистити код».	1	*Діяльність:* Використання ChatGPT для перевірки коду на помилки та оптимізацію. Peer-review (перевірка роботи сусіда).
33	**Фінальне тестування та налагодження.** Виправлення багів.	1	*Діяльність:* Тестування всіх сценаріїв роботи додатку. Підготовка до презентації.
34	**Захист проєктів (Частина 1).**	1	*Діяльність:* Презентація робіт учнями. Демонстрація функціонала.
35	**Захист проєктів (Частина 2). Підсумки курсу.**	1	*Діяльність:* Завершення презентацій. Обговорення вивченого. Нагородження кращих проєктів.

ДОДАТОК Б. Приклади практичних завдань (картки-інструкції)

Завдання 1. «Цифрова візитівка» (Рівень: Початковий)

Мета: Закріпити навички роботи з JSX та CSS.

Умова: Створити компонент `BusinessCard`, який відображає:

1. Фотографію (використати `img` тег).
2. Ім'я та прізвище (в тезі `h2`).
3. Посаду (наприклад, «Frontend Developer Student»).
4. Кнопку «Зв'язатися зі мною» (поки що неактивну). Вимога: Всі стилі мають бути в окремому файлі `styles.css`. AI-челендж: Попроси ChatGPT згенерувати CSS для ефекту «тіні при наведенні» (hover shadow) і додай його до картки.

Завдання 2. «Лічильник лайків» (Рівень: Середній)

Мета: Опанувати хук `useState` та обробку подій.

Умова:

1. Створити компонент `LikeButton`.
2. Використати змінну стану `likes` (початкове значення 0).

3. При натисканні на кнопку кількість лайків має збільшуватися на 1.
4. Ускладнення: Якщо лайків більше 10, колір тексту має стати червоним. Підказка: Використай тернарний оператор в атрибуті `style: color: likes > 10? 'red' : 'black'`.

Завдання 3. «Список справ (To-Do List)» (Рівень: Високий)

Мета: Робота зі списками, формами та імутабельністю стану.

Умова:

1. Створити масив об'єктів-завдань: ``.
2. Вивести список на екран, використовуючи `.map()`.
3. Створити форму (input + button) для додавання нового завдання.
4. Реалізувати функцію додавання: створити новий масив, додавши туди новий об'єкт (не використовувати `push`, тільки `[...spread]`).

ДОДАТОК В. Опис та критерії оцінювання фінального навчального проєкту

Тема проєкту: Розробка інтерактивного вебзастосунку (SPA).

Приклади тем:

- Movie Search App (Пошук фільмів за назвою).
- Expense Tracker (Трекер витрат).
- Quiz App (Тестування з варіантами відповідей).

Рубрика оцінювання (на основі таксономії SOLO)

Рівень	Бали	Критерії (Дескриптори)
1. Початковий (Unistructural)	1-5	Проєкт запускається, але містить критичні помилки. Код написаний в одному файлі. Використано лише статичний JSX. Логіка відсутня або скопійована без розуміння.
2. Середній (Multistructural)	6-8	Проєкт розбито на компоненти. Є спроба використати useState. Дані передаються через props, але є помилки (наприклад, мутація стейту). Інтерфейс працює частково.
3. Достатній (Relational)	9-10	Чітка структура компонентів. Правильне використання хуків. Списки мають унікальні ключі (key). Відсутні помилки в консолі. Реалізовано

		додавання/видалення елементів. Учень може пояснити, як дані рухаються між компонентами.
4. Високий (Extended Abstract)	11-12	Проєкт має завершений вигляд (UX/UI). Код оптимізований, чистий, з коментарями. Використано додаткові можливості (валідація форм, анімації, локальне сховище). Пройдено етап «Self-Review з ШІ».

Додаткові бали (+2 бали): За запис короткого відео-демо роботи програми.

ДОДАТОК Г. Анкета для учнів (зворотній зв'язок)

Шановний учню! Твої відповіді допоможуть нам покращити курс. Опитування анонімне.

1. Як ти оцінюєш свій рівень знань React після проходження курсу?

a) Нічого не зрозумів
b) Можу скопіювати код, але не розумію його
c) Розумію основи, можу писати прості програми
d) Впевнено створюю власні компоненти і логіку
e) Можу пояснити React іншому

2. Наскільки зручним було використання CodeSandbox (робота в браузері)?

a) Дуже незручно (краще б встановлювали програми на комп'ютер)
b) Нормально
c) Дуже зручно (можна працювати з дому/телефону, нічого не треба налаштовувати)

3. Чи допоміг тобі Штучний Інтелект (ChatGPT) у навчанні?

a) Я не користувався ним
b) Він тільки заважав / давав неправильні відповіді
c) Використовував, щоб писати код за мене (списував)
d) Використовував, щоб пояснити помилки та незрозумілі моменти (як тьютор)

4. Яка тема була найскладнішою?

a) JSX та верстка
b) Props (передача даних)

c) State (useState, зміна даних)

d) Масиви та списки (.map)

5. Що тобі сподобалося найбільше? (Обери до 2 варіантів)

a) Live Coding (коли вчитель пише код у реальному часі)

b) Створення власного проєкту

c) Робота з помилками через ChatGPT

d) Відсутність складних домашніх завдань

Дякуємо за відповіді!