МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

# КРОС-ПЛАТФОРМНЕ ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальностями F3 «Комп'ютерні науки»,
F6 «Інформаційні системи та технології»

Київ 2026

УДК 004.42
К83

Укладачі: Ю. О. Науменко, канд. техн. наук, доцент
А. Г. Хаддад, керівник ФОП

Рецензент О. В. Горда, канд. техн. наук, доцент

Відповідальна за випуск Т. А. Гончаренко, д-р техн. наук, професор

*Затверджено на засіданні кафедри інформаційних технологій, протокол № 6 від 16 лютого 2026 року.*

В авторській редакції.

**Крос-платформне** програмування [електронний ресурс]: методичні
К83 вказівки до виконання лабораторних робіт / уклад.: Ю.А. Науменко та ін. – Київ: КНУБА, 2026. – 32 с.

Містять зміст, порядок оформлення і вказівки до виконання лабораторних робіт.

Призначено для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальностями F3 «Комп'ютерні науки», F6 «Інформаційні системи та технології».

**ЗМІСТ**

# ЗАГАЛЬНІ ПОЛОЖЕННЯ

Дисципліна «Крос-платформне програмування» є наскрізною для програм підготовки фахівців з комп'ютерних наук та програмної інженерії, оскільки формує у здобувачів практичну здатність проєктувати, розробляти та підтримувати програмні продукти, що працюють на різних платформах (Web, Mobile, Desktop) із використанням спільної кодової бази, повторного використання компонентів і єдиних принципів архітектури. Актуальність курсу зумовлена потребою ринку у швидкій розробці та масштабуванні цифрових продуктів, уніфікації інтерфейсів і бізнес-логіки, а також вимогами до продуктивності, безпеки й якості користувацького досвіду в багатоплатформних середовищах.

Методичні вказівки з дисципліни охоплюють послідовний шлях формування компетентностей: від створення базових вебсторінок і розуміння семантичної структури документа – до побудови адаптивних інтерфейсів на CSS (Flexbox, Grid, Media Queries), програмування на JavaScript (типи даних, умови, масиви, цикли, функції, об'єкти, модулі, JSON) та роботи з даними. Окремий блок присвячено налаштуванню робочого середовища і сучасним інструментам збірки та автоматизації (SCSS, Gulp, Webpack, production-оптимізація, Babel). Завершальні лабораторні спрямовані на розробку SPA на React (стан, ефекти, маршрутизація, Context API) та створення серверної частини на Node.js/Express із реалізацією REST API (маршрути, обробники POST/PUT/DELETE, JSON) і fullstack-інтеграцією клієнта та сервера.

Окремий акцент у навчальному процесі зроблено на переході від верстки та стилізації до компонентного підходу й побудови застосунків із чітким розподілом відповідальностей: інтерфейс - стан - дані - API. Здобувачі вивчають не лише створення окремих елементів UI, а й принципи адаптивного лейауту, повторного використання компонентів, керування станом застосунку, маршрутизацію та організацію коду (BEM, модульність). Це закладає фундамент для розуміння того, як фронтенд-частина інтегрується з бекендом, як обробляються дані та як готується проєкт до збірки і розгортання у production.

Вказівки містять рекомендації й типові приклади, що у практичному зрізі відображають рівень засвоєння курсу, зокрема здобувач повинен уміти:

- ✓ створювати базові HTML-сторінки та застосовувати семантичну розмітку, розуміючи структуру документа і блокову модель;
- ✓ працювати з таблицями та формами, застосовувати float та sprites у межах навчальних задач;

- ✓ будувати адаптивні інтерфейси за допомогою CSS Flexbox, Grid і Media Queries;
- ✓ оперувати змінними та типами даних у JavaScript, виконувати базові обчислення та перетворення;
- ✓ реалізовувати логіку прийняття рішень через умовні конструкції та логічні вирази;
- ✓ працювати з масивами, циклами й методами, організовуючи обробку даних;
- ✓ створювати функції та об'єкти, проєктуючи повторно використовуваний код;
- ✓ організовувати модульну структуру, працювати з JSON та обміном даними;
- ✓ налаштовувати робоче середовище та інструменти розробки для кросплатформних проєктів;
- ✓ використовувати SCSS (змінні, вкладеність, міксіни, функції, partials) для побудови масштабованих стилів;
- ✓ застосовувати методологію BEM для створення UI-компонентів і підтримуваної структури стилів;
- ✓ налаштовувати автоматизацію збірки (Gulp) та базову/production-збірку (Webpack, Babel, оптимізація);
- ✓ створювати SPA на React: компоненти, useState/useEffect, маршрутизацію (React Router) та глобальний стан (Context API);
- ✓ працювати з Node.js filesystem для базових операцій із файлами;
- ✓ розробляти REST API на Express: маршрути, обробники POST/PUT/DELETE, роботу з JSON;
- ✓ інтегрувати React-клієнт із серверним API, реалізовуючи повний цикл обміну даними.

Для полегшення засвоєння матеріалу наводяться інструкції з конфігурації інструментів, готові лістинги коду та практичні вказівки до виконання лабораторних робіт і мініпроєктів.

Завершивши вивчення дисципліни, здобувач повинен уміти самостійно: проєктувати адаптивні інтерфейси та компонентні UI; реалізовувати бізнес-логіку на JavaScript із модульною організацією коду; налаштовувати інструменти збірки та оптимізації; розробляти SPA на React; створювати та документувати REST API на Node.js/Express; виконувати fullstack-інтеграцію клієнта та сервера. Це забезпечує необхідну базу для подальшого вивчення складніших архітектурних підходів, патернів проєктування та сучасних кросплатформних технологій.

## Лабораторна робота №1

**Тема:** Базова веб-сторінка

***Мета:*** Опанувати базову структуру HTML-документа та навички підключення CSS і JavaScript.

### *Короткі теоретичні відомості*

**HTML-документ** – документ має містити коректний doctype, тег <html> з атрибутом lang, розділи <head> і <body>.

**Метатеги** (charset, viewport) впливають на кодування та адаптивне відображення.

**Підключення ресурсів**: CSS через <link>, JS через <script> (рекомендовано з атрибутом defer).

**Валідація розмітки та базова доступність** (alt для зображень, заголовки) — обов'язкова частина якості.

### *Порядок виконання:*

***Крок 1.*** Створіть структуру проєкту: /index.html, /css/style.css, /js/main.js, /assets/

***Крок 2.*** У index.html зберіть базову сторінку-візитівку (заголовок, абзац, список, зображення, посилання).

***Крок 3.*** Підключіть style.css і додайте базові стилі: шрифти, кольори, відступи, максимальну ширину контейнера.

***Крок 4.*** Підключіть main.js (defer) і виведіть у консоль повідомлення про старт застосунку

***Крок 5.*** Перевірте сторінку у валідаторі HTML та виправте помилки.

### Контрольні запитання (самоконтроль)

1. Яка роль <!DOCTYPE html>?
2. Навіщо потрібні метатеги charset і viewport?
3. Що таке семантичний заголовок сторінки (h1) і чому він має бути один?
4. Для чого потрібен атрибут alt в <img>?

## Лабораторна робота №2

**Тема:** Семантична розмітка + структура документа, display, блокова модель

***Мета:*** Навчитися будувати семантичну структуру сторінки та керувати відображенням елементів через display і box model.

### *Короткі теоретичні відомості*

**Семантичні теги** (<header>, <nav>, <main>, <section>, <article>, <footer>) описують зміст і структуру, покращують доступність та SEO.

**display** визначає модель відображення елемента (block, inline, inline-block, none, flex, grid).

**Блокова модель** (box model): content + padding + border + margin. box-sizing: border-box спрощує розрахунки.

**Нормалізація стилів:** reset/normalize, базові глобальні правила для типографіки.

***Порядок виконання:***

***Крок 1.*** Переробіть сторінку з ЛР1, використовуючи семантичні теги (header/nav/main/section/footer)

***Крок 2.*** Створіть навігацію з якірними посиланнями на секції сторінки.

***Крок 3.*** Додайте CSS правило box-sizing: border-box для всіх елементів.

***Крок 4.*** Зробіть 2-3 блоки контенту з різними display (block/inline-block) і продемонструйте відмінності відступів.

***Крок 5.*** Оформіть макет: контейнер, відступи між секціями, простий хедер.

### Контрольні запитання (самоконтроль)

1. У чому перевага семантичних тегів над ?
2. Як працює box-sizing: border-box?.
3. Яка різниця між display: inline та inline-block?
4. Коли доцільно використовувати <section>, а коли <article>?
5. Що таке колапс вертикальних margin?

## Лабораторна робота №3

**Тема:** Таблиці та форми, float, sprites

***Мета:*** Закріпити навички створення таблиць і форм та ознайомитися з float і sprites на практиці.

### *Короткі теоретичні відомості*

**Таблиці** (<table>, <thead>, <tbody>, <tr>, <th>, <td>) використовують для табличних даних, а не для лейауту.

**Форми:** <form>, <input>, <select>, <textarea>, <label>; атрибути name, value, required, pattern, type.

**Float -** застарілий інструмент для обтікання; потребує clearfix. У сучасних макетах переважно flex/grid.

**CSS sprites -** об'єднання іконок в один файл для зменшення кількості запитів; позиціювання через background-position.

### *Порядок виконання:*

***Крок 1.*** Створіть таблицю розкладу/прайсу (мінімум 4x4) з заголовками стовпців і рядків.

***Крок 2.*** Зверстайте форму з полями: ім'я, email, тип запиту (select), повідомлення (textarea), кнопка submit.

***Крок 3.*** Додайте валідацію HTML (required, type=email).

***Крок 4.*** Зробіть блок з float-обтіканням зображення текстом та очисткою потоку (clearfix).

***Крок 5.*** План аналізу (рис.3).

Додайте 3 іконки зі sprites (або підготуйте один png з трьома іконками) і підключіть їх як background.

### Контрольні запитання (самоконтроль)

1. Коли таблиця є доречною, а коли ні?
2. Навіщо потрібен <label for="...">?
3. Чим float відрізняється від flex/grid з точки зору призначення?
4. Що таке clearfix і чому він потрібен?
5. Які плюси/мінуси використання sprites у порівнянні з SVG?

## Лабораторна робота №4

**Тема:** CSS Flexbox + адаптивний лейаут

***Мета:*** Опанувати Flexbox для побудови адаптивних компонентів і сіток.

### *Короткі теоретичні відомості*

**Flexbox –** модель для вирівнювання елементів в одному вимірі (рядок або колонка).

**Основні властивості контейнера:** display:flex, flex-direction, justify-content, align-items, flex-wrap, gap.

**Властивості елементів:** flex (grow/shrink/basis), order, align-self.

**Адаптивний лейаут:** зміна напрямку, перенесення, масштабування типографіки та відступів під ширину екрана.

### *Порядок виконання:*

***Крок 1.*** Створіть секцію з картками (6–8 карток) та зверстайте її на Flexbox з переносом (flex-wrap).

***Крок 2.*** . Зробіть навігацію у хедері: логотип зліва, меню справа, на мобільному — меню в колонку.

***Крок 3.*** Налаштуйте відстані через gap замість margin між елементами.

***Крок 4.*** Зробіть одну картку «featured», що займає більше місця (через flex-basis/ grow).

***Крок 5.*** . Перевірте вигляд на 320px, 768px, 1024px.

### Контрольні запитання (самоконтроль)

1. У чому різниця між justify-content та align-items?
2. Що робить властивість flex-wrap?
3. Для чого потрібен flex-basis?
4. Як працює order і які ризики для доступності він несе?
5. Чому gap часто кращий за margin для відстаней між елементами?

## Лабораторна робота №5

**Тема:** CSS Grid та Media Queries

***Мета:*** Опанувати CSS Grid та навчитися застосовувати Media Queries для адаптивності

### *Короткі теоретичні відомості*

**CSS Grid -** двовимірна сітка (рядки і колонки) для складних макетів

**Основні поняття:**grid-container, grid-item, tracks, areas, implicit/explicit grid**.**

**Ключові властивості:** grid-template-columns/rows, gap, grid-template-areas, auto-fit/auto-fill, minmax().

**Media queries:** дозволяють змінювати сітку, типографіку і відступи залежно від ширини/орієнтації/щільності єкрану пристроя.

### *Порядок виконання:*

***Крок 1.*** Зробіть лейаут сторінки: header / sidebar / content / footer на Grid.

***Крок 2.*** Налаштуйте grid-template-areas для читабельного опису макета.

***Крок 3.*** Реалізуйте сітку галереї з auto-fit + minmax(200px, 1fr).

***Крок 4.*** Додайте 2 брейкпоінти: до 768px (sidebar під контентом), до 480px (одна колонка).

***Крок 5.*** Додайте префіксну перевірку: протестуйте у двох браузерах.

### Контрольні запитання (самоконтроль)

1. Коли Grid кращий за Flexbox?
2. Що таке grid-template-areas і навіщо воно?
3. Чим auto-fit відрізняється від auto-fill?
4. Що робить minmax()?
5. Як правильно обирати брейкпоінти для макета?

## Лабораторна робота №6

**Тема:** Змінні, типи даних, базові обчислення Javascript

***Мета:*** Навчитися працювати зі змінними, типами даних і базовими обчисленнями у JavaScript.

### *Короткі теоретичні відомості*

**JavaScript змінні:** let/const (блокова область видимості), var (функціональна).

**Типи даних:** number, string, boolean, null, undefined, object, symbol, bigint.

**Оператори :** арифметичні, присвоєння, порівняння (== vs ===), логічні.

**Перетворення типів:** дозволяють змінювати сітку, типографіку і відступи залежно від ширини/орієнтації/щільності єкрану пристроя.

### *Порядок виконання:*

***Крок 1.*** Створіть файл main.js і підключіть його до сторінки

***Крок 2.*** Оголосіть змінні (const/let) для ціни, кількості, знижки та обчисліть суму замовлення.

***Крок 3.*** Виведіть результати в консоль і в DOM (в окремий ).

***Крок 4.*** Продемонструйте різницю між == та === на 3 прикладах.

***Крок 5.*** Зробіть простий калькулятор: два поля вводу + кнопка, що рахує суму.

### Контрольні запитання (самоконтроль)

1. Коли використовувати const, а коли let?
2. Яка різниця між null та undefined?
3. Чому === вважається безпечнішим за ==?
4. Що таке NaN і як його перевірити?
5. Які є способи перетворити рядок у число?

## Лабораторна робота №7

**Тема:** Умовні конструкції + логіка у JavaScript

***Мета:*** : Навчитися реалізовувати розгалуження та перевірки, використовуючи умовні конструкції та логіку.

### *Короткі теоретичні відомості*

**Умовні конструкції:** if/else, else if, switch, тернарний оператор.

**Логічні оператори:** &&, ||, !; коротке замикання (short-circuit).

**Оператори:** Truthy/Falsy значення у JS впливають на поведінку умов

### *Порядок виконання:*

***Крок 1.*** Зробіть форму перевірки віку: якщо <18 — показати повідомлення про обмеження, інакше — доступ дозволено.

***Крок 2.*** Реалізуйте оцінювання: за балом (0–100) виводьте категорію A/B/C/D/F через if/else if.

***Крок 3.*** Зробіть switch за типом доставки (pickup, courier, post) і виведіть опис

***Крок 4.*** Додайте перевірку на порожній ввід (trim) та повідомлення про помилку.

***Крок 5.*** Зробіть тернарний оператор для швидкого вибору тексту кнопки.

### Контрольні запитання (самоконтроль)

1. Що таке truthy/falsy? Наведіть приклади.
2. Чим відрізняється тернарний оператор від if/else?
3. Як працює коротке замикання в && та ||?
4. Коли switch кращий за if/else?
5. Як коректно порівнювати рядки без помилок регістру?

## Лабораторна робота №8

**Тема:** Масиви, цикли, методи

***Мета:*** : Опанувати масиви, цикли та популярні методи масивів для обробки даних..

### *Короткі теоретичні відомості*

**Масиви:** створення, доступ за індексом, довжина (length), ітерації.

**Цикли:** for, while, do...while, for...of, forEach.

**Методи масивів:** map, filter, reduce, find, some/every, sort

**Мутація vs імутабельність**: push/pop/splice змінюють масив, а map/filter повертають новий

### *Порядок виконання:*

***Крок 1.*** Створіть масив товарів (об’єкти з name, price) та виведіть список у DOM.

***Крок 2.*** Додайте фільтр: показати товари дорожче за задану суму (filter).

***Крок 3.*** Порахуйте загальну вартість (reduce).

***Крок 4.*** . Відсортуйте товари за ціною (sort) і виведіть результат.

***Крок 5.*** . Зробіть цикл, що генерує 10 випадкових чисел і зберігає їх у масив.

### Контрольні запитання (самоконтроль)

1. Яка різниця між for...of і for...in?
2. Для чого використовують map()?
3 Чим filter() відрізняється від find()?
4. Що таке reduce() і які задачі він вирішує
5. Які методи масиву є мутуючими?

## Лабораторна робота №9

**Тема:** Функції і об’єкти

***Мета:*** : Закріпити навички роботи з функціями та об’єктами й основами організації коду.

### *Короткі теоретичні відомості*

**Функції:** function declaration, function expression, arrow functions.

**Об’єкти:** властивості, методи, доступ через крапку та дужки, деструктуризація.

**Контекст this -** базові правила його визначення (звичайна vs стрілкова функція)

### *Порядок виконання:*

***Крок 1.*** Напишіть функції: formatPrice(price), calcDiscount(price, percent).

***Крок 2.*** Створіть об’єкт користувача з полями name/email та методом getDisplayName().

***Крок 3.*** Додайте деструктуризацію об’єкта у функції renderUser({name, email}).

***Крок 4.*** Напишіть приклад із this у методі об’єкта та порівняйте зі стрілковою функцією.

***Крок 5.*** . Створіть масив об’єктів і відрендерте їх у список.

### Контрольні запитання (самоконтроль)

1. Чим function declaration відрізняється від function expression?
2. Коли зручно використовувати arrow function?
3 Що таке деструктуризація і навіщо вона?
4. Як працює this у методі об’єкта?
5. Що таке rest-параметри і чим вони відрізняються від arguments?

## Лабораторна робота №10

**Тема:** Модулі, JSON, робота з даними

***Мета:*** : Навчитися працювати з модулями, JSON та завантаженням даних у браузері.

### *Короткі теоретичні відомості*

**ES Modules:** export/import, іменовані та дефолтні експорти.

**JSON:** серіалізація (JSON.stringify) та парсинг (JSON.parse).

**Робота з даними у браузері:** fetch API, проміси, async/await.

**Локальне зберігання:** localStorage/sessionStorage (ключ-значення).

### *Порядок виконання:*

***Крок 1.*** Створіть модуль utils.js з функціями форматування (export).

***Крок 2.*** Імпортуйте функції у main.js та використайте їх у проєкті.

***Крок 3.*** Завантажте дані з відкритого JSON-ендпоїнта (або локального data.json) через fetch.

***Крок 4.*** Відобразіть список елементів у DOM та додайте обробку помилок (try/catch).

***Крок 5.*** Збережіть вибір користувача в localStorage та відновіть його при перезавантаженні сторінки.

### Контрольні запитання (самоконтроль)

1. Чим відрізняється named export від default export?
2. Що таке JSON і які типи він підтримує?
3. Яка різниця між Promise.then() і async/await?
4. Як правильно обробляти помилки мережевого запиту?
5. Коли доречно використовувати localStorage?

## Лабораторна робота №11

**Тема:** Налаштування робочого середовища

***Мета:*** Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**Робоче середовище:** VS Code, браузерні DevTools, Node.js та npm для менеджменту залежностей.

**Структура проєкту та базові налаштування:** .editorconfig, форматування, лінтинг (за потреби).

**Git:** репозиторій, коміти, гілки, .gitignore.

### *Порядок виконання:*

***Крок 1.*** Встановіть Node.js LTS та перевірте версії node -v, npm -v.

***Крок 2.*** Налаштуйте VS Code: форматування, авто-збереження, корисні розширення (ESLint/Prettier за потреби).

***Крок 3.*** Створіть репозиторій Git для лабораторних; додайте .gitignore (node_modules, dist тощо).

***Крок 4.*** Підготуйте шаблон папок для наступних робіт та додайте README з інструкціями запуску.

### Контрольні запитання (самоконтроль)

1. Навіщо потрібен npm та package.json?
2. Що таке DevTools і які вкладки найчастіше використовують фронтенд-розробники?
3. Які файли не варто комітити в репозиторій і чому?
4. Для чого потрібен README у навчальному проєкті?

## Лабораторна робота №12

**Тема:** SCSS: змінні, вкладеність, базові компоненти

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**SCSS додає надбудови над CSS:** змінні, вкладеність, часткові файли, імпорт/використання.

Каскад і специфічність залишаються такими ж, як у CSS.

Компіляція SCSS у CSS виконується інструментами збірки або CLI.

### *Порядок виконання:*

***Крок 1.*** Створіть scss/ з файлами _variables.scss, _base.scss, components/_button.scss та main.scss.

***Крок 2.*** Опишіть змінні кольорів/відступів/шрифтів та використайте їх у стилях.

***Крок 3.*** Застосуйте вкладеність для компонента кнопки та станів :hover/:focus.

***Крок 4.*** Скомпілюйте SCSS у CSS (будь-яким способом) і підключіть результат до сторінки..

### Контрольні запитання (самоконтроль)

1. Яка різниця між SCSS і CSS?
2. Навіщо потрібні змінні в SCSS?
3. Чим небезпечна надмірна вкладеність?
4. Що таке partial-файл і чому він починається з '_'?

## Лабораторна робота №13

**Тема:** SCSS міксіни, функції, partials

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**Міксіни (mixins)** — багаторазові фрагменти стилів із параметрами.

**Функції (functions)** повертають значення і зручні для обчислень (кольори, розміри).

**Partials** та структура SCSS допомагають організувати код: base, layout, components, utilities.

### *Порядок виконання:*

***Крок 1.*** Створіть mixin для медіа-запитів (наприклад, respond($bp)).

***Крок 2.*** Створіть mixin для кнопки (розміри, радіус, кольори) з параметрами.

***Крок 3.*** `Створіть SCSS-функцію для переведення px → rem.`

***Крок 4.*** Розбийте стилі на partials і зберіть їх у main.scss.

### Контрольні запитання (самоконтроль)

1. Коли міксін кращий за placeholder/selectors?
2. Чим функції відрізняються від міксінів?
3. Навіщо переводити px у rem?
4. Як організація partials впливає на підтримуваність стилів?

## Лабораторна робота №14

**Тема:** Методологія BEM + створення UI-компоненту

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**BEM (Block__Element--Modifier)** — методологія іменування класів для масштабованих інтерфейсів.

**Блок** — незалежний компонент; елемент — частина блока; модифікатор — варіація стану/вигляду.

**Компонентний підхід:** один UI-компонент — один набір класів і стилів.

### *Порядок виконання:*

***Крок 1.*** Обрати UI-компонент: картка товару або модальне вікно або випадаюче меню.

***Крок 2.*** Описати структуру HTML з BEM-класами (block, elements, modifiers).

***Крок 3.*** Написати стилі (CSS або SCSS) так, щоб модифікатори змінювали стан (наприклад, --active, --disabled).

***Крок 4.*** Додати мінімальну інтерактивність JS (відкриття/закриття, перемикання стану).

### Контрольні запитання (самоконтроль)

1. Які проблеми вирішує BEM у великих проєктах?
2. Чим модифікатор відрізняється від елемента?
3. Як називати блоки, що використовуються повторно на сторінці?
4. Які типові помилки при використанні BEM?

## Лабораторна робота №15

## Тема: Gulp: автоматизація збірки SCSS → CSS

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**Gulp** — таск-ранер для автоматизації рутинних задач (компіляція, мінімізація, автопрефікси).

**Типовий пайплайн:** SCSS → CSS, автопрефіксер, sourcemaps, watch.

**Збірка має бути повторюваною:** npm scripts + gulpfile

### *Порядок виконання:*

***Крок 1.*** Ініціалізуйте проєкт npm та встановіть gulp + плагіни для SCSS, sourcemaps, autoprefixer.

***Крок 2.*** Налаштуйте задачу build: компіляція SCSS у dist/css з sourcemaps.

***Крок 3.*** Налаштуйте задачу watch для автоматичної збірки при зміні scss-файлів.

***Крок 4.*** Додайте npm scripts: "dev" і "build" та перевірте роботу.

### Контрольні запитання (самоконтроль)

1. Чим Gulp відрізняється від Webpack за підходом?
2. Навіщо потрібні sourcemaps?
3. Що робить autoprefixer і на що він спирається?
4. Чому важливо мати watch-режим у розробці?

## Лабораторна робота №16

**Тема:** Webpack базова збірка

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**Webpack** — модульний бандлер для JS/CSS/asset-ів.

`Entry → loaders/plugins → output; dev server прискорює розробку.`

**Базові поняття:** module graph, tree-shaking (за можливості), source maps.

### *Порядок виконання:*

***Крок 1.*** Створіть мінімальний webpack.config.js з entry src/index.js та output dist/bundle.js.

***Крок 2.*** Додайте loader для CSS (import './style.css' у JS).

***Крок 3.*** Запустіть dev server і перевірте hot reload (за можливості).

***Крок 4.*** Зберіть production build і переконайтесь, що bundle згенеровано.

### Контрольні запитання (самоконтроль)

1. Що таке entry та output у Webpack?
2. Для чого потрібні loaders?
3. Чим dev server відрізняється від статичного відкриття HTML?
4. Які проблеми вирішує бандлер у порівнянні з ручним підключенням скриптів?

## Лабораторна робота №17

**Тема:** Production збірка, Babel, оптимізація

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**Production-збірка**: мінімізація, оптимізація, розділення код-сплітингом, кешування.

**Babel** компілює сучасний JS до сумісного з цільовими браузерами (за конфігом).

**Оптимізація ресурсів:** minify CSS/JS, оптимізація зображень, аналіз bundle.

### *Порядок виконання:*

***Крок 1.*** Додайте режим production у збірку та увімкніть мінімізацію.

***Крок 2.*** Налаштуйте Babel для транспіляції (preset-env) та перевірте на сучасному синтаксисі.

***Крок 3.*** Налаштуйте розділення vendor/app (за можливості) або dynamic import для частини функціоналу.

***Крок 4.*** Запустіть аналіз розміру bundle (webpack-bundle-analyzer або аналог) та зробіть короткі висновки.

### Контрольні запитання (самоконтроль)

1. Навіщо потрібен Babel у 2026 році, якщо браузери стали сучаснішими?

2. Що таке code splitting і яку проблему він вирішує?

3. Як кешування пов’язане з хешами у назвах файлів?

4. Які метрики продуктивності фронтенду ви знаєте (LCP, CLS тощо)?

## Лабораторна робота №18

**Тема:** React: створення SPA проєкту

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**React** — бібліотека для побудови UI на основі компонентів і декларативного рендерингу.

**SPA (Single Page Application)** — застосунок, що оновлює контент без повного перезавантаження сторінки.

**Інструменти:** Vite або CRA (за вибором), структура src/components.

***Порядок виконання:***

***Крок 1.*** Створіть React-проєкт (Vite рекомендується) та запустіть dev server.

***Крок 2.*** Створіть 3 компоненти: Header, PageTitle, CardList.

***Крок 3.*** Передайте props у CardList і відрендерте список елементів через map.

***Крок 4.*** Додайте базові стилі та перевірте, що збірка працює.

### Контрольні запитання (самоконтроль)

1. Що таке компонент у React?

2. Що таке props і для чого вони потрібні?

3. Чому не можна використовувати index як key у списках (у загальному випадку)?

4. У чому різниця між SPA та MPA?

## Лабораторна робота №19

**Тема:** Відпрацювання useState + useEffect

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**useState** керує локальним станом компонента.

**useEffect** запускається після рендера і використовується для сайд-ефектів (запити, підписки).

Залежності у useEffect визначають, коли ефект перезапускається.

### *Порядок виконання:*

***Крок 1.*** Реалізуйте лічильник з useState (increment/decrement/reset).

***Крок 2.*** Додайте useEffect, що змінює document.title залежно від значення лічильника.

***Крок 3.*** Зробіть завантаження списку даних через fetch у useEffect (async/await) та відобразіть стан loading/error.

**Крок 4.** Додайте очищення (cleanup) в useEffect на прикладі таймера setInterval.

### Контрольні запитання (самоконтроль)

1. Коли React викликає useEffect?
2. Що буде, якщо не вказати масив залежностей?
3. Навіщо потрібен cleanup у useEffect?
4. Чим відрізняється локальний state від props?

## Лабораторна робота №20

**Тема:** Робота з React Router

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**React Router -** забезпечує маршрутизацію на клієнті.

**Поняття:** Route, Link/NavLink, Outlet, параметри маршруту, навігація.

Динамічні маршрути і захищені маршрути.

### *Порядок виконання:*

***Крок 1.*** Додайте React Router у проєкт і створіть 3 сторінки: Home, Products, About.

***Крок 2.*** Налаштуйте меню з NavLink і підсвічуванням активного маршруту.

***Крок 3.*** Створіть динамічний маршрут /products/:id та сторінку ProductDetails.

***Крок 4.*** Додайте 404-сторінку (route '*').

### Контрольні запитання (самоконтроль)

1. Чим Link відрізняється від <a href> у SPA?
2. Що таке Outlet і для чого він потрібен?
3. Як отримати параметр :id у компоненті?
4. Навіщо потрібна 404-сторінка в клієнтській маршрутизації?

## Лабораторна робота №21

**Тема:** Використання Context API

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**Context API** дозволяє передавати дані через дерево компонентів без «prop drilling».

**Типові кейси:** тема, мова, авторизація, налаштування застосунку.

Важливо контролювати перерендери та розділяти контексти за відповідальністю.

### *Порядок виконання:*

***Крок 1.*** Створіть контекст ThemeContext з двома темами: light/dark.

***Крок 2.*** Додайте перемикач теми у Header та застосуйте клас до кореневого контейнера.

***Крок 3.*** Збережіть вибір теми у localStorage і відновлюйте при старті.

***Крок 4.*** Додайте другий контекст (наприклад, AuthContext з фейковим логіном) і продемонструйте споживання у двох компонентах.

### Контрольні запитання (самоконтроль)

1. Яку проблему вирішує Context API?
2. Коли Context використовувати не варто?
3. Чим Context відрізняється від state-менеджерів (Redux/Zustand) на концептуальному рівні?
4. Як зменшити зайві перерендери при використанні Context?

## Лабораторна робота №22

**Тема:** Node.js filesystem

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**Node.js** надає API для роботи з файловою системою через модуль fs (sync/async).

**Path module** допомагає формувати кросплатформні шляхи.

**Найчастіші операції:** читання/запис файлів, створення директорій, читання JSON.

### *Порядок виконання:*

***Крок 1***. Створіть Node-скрипт, що читає файл input.txt і виводить кількість рядків та слів.

***Крок 2.*** Додайте запис результатів у output.json (об’єкт зі статистикою).

***Крок 3.*** Реалізуйте обробку помилок (try/catch або callback error-first).

***Крок 4.*** Додайте CLI-аргумент для шляху до файлу (process.argv).

### Контрольні запитання (самоконтроль)

1. Чим асинхронні методи fs кращі за синхронні на сервері?
2. Для чого потрібен модуль path?
3. Як обробити помилку, якщо файл не існує?
4. Що таке event loop у Node.js (на базовому рівні)?

## Лабораторна робота №23

**Тема:** Express API

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**Express -** мінімалістичний фреймворк для HTTP-серверів на Node.js.

**Маршрути (routes) -** визначають обробники для методів і шляхів.

**Middleware -** функції, що виконуються між запитом і відповіддю.

### *Порядок виконання:*

***Крок 1.*** Створіть Express-проєкт і підніміть сервер на порту 3000.

***Крок 2.*** Додайте маршрути: GET /health, GET /api/products, GET /api/products/:id.

***Крок 3.*** Створіть middleware для логування методу/шляху/часу відповіді.

***Крок 4.*** Підключіть express.json() і перевірте, що сервер приймає JSON.

### Контрольні запитання (самоконтроль)

1. Що таке middleware в Express?
2. Чим відрізняються app.get і app.use?
3. Як отримати параметр маршруту :id?
4. Навіщо потрібен маршрут /health?

## Лабораторна робота №24

**Тема:** Створення обробника POST / PUT / DELETE

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

**CRUD-операції** у REST: POST (create), PUT/PATCH (update), DELETE (delete).

**Статус-коди:** 201 Created, 200 OK, 204 No Content, 400/404/500.

**Валідація** даних на сервері та структура JSON-відповіді з помилками.

### *Порядок виконання:*

***Крок 1.*** Реалізуйте в Express ендпоїнти: POST /api/products, PUT /api/products/:id, DELETE /api/products/:id.

***Крок 2.*** Зберігайте дані у пам’яті (масив) або у файлі JSON (за бажанням).

***Крок 3.*** Додайте валідацію: name (не порожній), price (number > 0).

***Крок 4.*** Поверніть коректні статус-коди і тіло відповіді для успіху/помилок.

***Крок 5.*** Перевірте через Postman/Insomnia або curl.

### Контрольні запитання (самоконтроль)

1. Яка різниця між PUT і PATCH?
2. Коли доречно повертати 204 No Content?
3. Що таке ідемпотентність і які методи її мають?
4. Навіщо потрібна серверна валідація, якщо є клієнтська

## Лабораторна робота №25

**Тема:** Fullstack інтеграція React + API

***Мета:*** : Опанувати практичні навички відповідно до теми лабораторної роботи та застосувати їх у невеликому проєкті.

### *Короткі теоретичні відомості*

Fullstack інтеграція: React (клієнт) споживає API (сервер) через HTTP.

Проблеми інтеграції: CORS, базові помилки мережі, узгодження форматів даних.

Архітектура: розділення клієнта/сервера, змінні оточення, проксі у dev-режимі.

### *Порядок виконання:*

***Крок 1***. Запустіть API-сервер (ЛР23–24) і React-клієнт (ЛР18–21).

***Крок 2.*** У React створіть сервіс-обгортку для запитів (fetch/axios) та функції getProducts/createProduct.

***Крок 3***. Відобразіть список продуктів, додайте форму створення продукту і оновлення списку без перезавантаження.

***Крок 4.*** Обробіть помилки та стани loading.

***Крок 5.*** Налаштуйте CORS на сервері або проксі на клієнті (dev) і опишіть рішення у README.

### Контрольні запитання (самоконтроль)

1. Що таке CORS і чому він виникає у браузері?
2. Де краще зберігати URL API — у коді чи у змінних оточення?
3. Які стани UI треба передбачити при запитах (loading/error/empty)?
4. Які ризики несе пряме звернення клієнта до приватних ключів/секретів?

# Список літератури

1. Хайрова Н. Ф., Петрасова С. В. Сучасні технології Web-програмування : навч. посіб. – Харків : ФОП Панов А. М., 2020. – 112 с.

2. Шелестов А. Ю., Куссуль Н. М. Web-програмування. Лабораторний практикум : навч. посіб. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 102 с.

3. Бородкіна І. Л., Бородкін Г. О. Web-технології та Web-дизайн : застосування мови HTML для створення електронних ресурсів : навч. посіб. – Київ : Ліра-К, 2020. – 212 с.

4. Бунке О. С. WEB-технології : навч. посіб. – Київ : КПІ ім. Ігоря Сікорського, 2020. – 83 с.

5. Пасічник В. В., Пасічник О. В., Угрин Д. І. Веб-технології та веб-дизайн. Книга 1 : Веб-технології : навч. посіб. – Львів : Магнолія-2006, 2018. – 336 с.

6. Duckett J. HTML and CSS: Design and Build Websites. – Indianapolis : Wiley, 2011. – 490 p.

7. Flanagan D. JavaScript: The Definitive Guide. – Sebastopol : O'Reilly Media, 2020. – 704 p.

8. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley, 2002. – 560 p.

Навчально-методичне видання

# КРОС-ПЛАТФОРМНЕ ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальностями F3 «Комп'ютерні науки»,
F6 «Інформаційні системи та технології»

Укладачі: **Науменко** Юрій Олександрович,
**Хаддад** Антон Георгійович

Випусковий редактор *Ю.М. Долгополова*
Комп’ютерне верстання *Ю.М. Долгополової*

Ум. друк. арк. 2,09. Обл.-вид. арк. 2,25
Електронний документ. Вид № 72/V-26.

Виконавець і виготовлювач

Київський національний університет будівництва і архітектури
Проспект Повітряних Сил, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб’єктів
видавничої справи ДК № 808 від 13.02.2002