оптимальний для початківців, Gitpod – для середнього рівня, GitHub Codespaces – для просунутих курсів з акцентом на DevOps. Щодо педагогічних переваг, зменшення когнітивного навантаження за рахунок автоматизації технічного налаштування, індивідуалізація через ізольовані середовища для кожного студента, спільна співпраця завдяки real-time спільному редагуванню, автентичність через використання інструментів реальної розробки (Git, CI/CD).

Список використаних джерел

1. Mishra P., Koehler M. J. Technological pedagogical content knowledge : A framework for integrating technology in teachers' knowledge. Teachers College Record. 20016. Vol. 108. № 6. P. 1017–1054. URL: https://doi.org/10.1111/j.1467-9620.2006.00684.x (дата звернення: 27.10.2025).

2. Sweller J. Cognitive load theory. In J. P. Mestre & B. H. Ross (Eds.). The psychology of learning and motivation: Cognition in education. P. 37–76. Elsevier Academic Press. URL: https://doi.org/10.1016/B978-0-12-387691-1.00002-8 (дата звернення: 27.10.2025).

3. GitHub Codespaces Documentation. URL: https://docs.github.com/codespaces (дата звернення: 27.10.2025).

4. Gitpod Documentation. URL: https://www.gitpod.io/docs (дата звернення: 27.10.2025).

5. Replit Docs. URL: https://docs.replit.com (дата звернення: 27.10.2025).

ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ІГОР, СТВОРЕНИХ З ВИКОРИСТАННЯМ РУШІЯ GODOT ТА C#

Якименко Карина Миколаївна
здобувач першого рівня вищої освіти, спеціальність Комп'ютерні науки
Тернопільський національний педагогічний університет імені Володимира Гнатюка
yakymenko_km@fizmat.tnpu.edu.ua
Василенко Ярослав Пилипович
викладач кафедри інформатики та методики її навчання
Тернопільський національний педагогічний університет імені Володимира Гнатюка
yava@fizmat.tnpu.edu.ua

У сучасних умовах стрімкого розвитку інформаційних технологій вебзастосунки стали основним інструментом для реалізації бізнес-процесів, надання послуг і забезпечення комунікації між користувачами та сервісами. Щодня зростають вимоги до швидкодії, масштабованості, інтерактивності та зручності вебсистем. Тому перед розробниками постає завдання не лише створювати функціональні рішення, а й забезпечувати ефективну взаємодію між клієнтською та серверною частинами програмного продукту.

Одним із найпоширеніших підходів до побудови сучасних вебсервісів є архітектура REST API, яка забезпечує обмін даними у форматі JSON між різними компонентами системи. Такий підхід дає можливість розділити логіку застосунку на незалежні частини, що підвищує гнучкість, масштабованість і зручність підтримки проєкту [1].

Серед технологій, що забезпечують ефективну розробку REST API, особливе місце посідає Express.js – мінімалістичний і гнучкий фреймворк для Node.js, який спрощує створення серверних застосунків, маршрутизацію запитів і роботу з middleware. Його перевагами є висока продуктивність, модульність і розвинена

екосистема додаткових бібліотек [2; 3]. Саме завдяки цим характеристикам Express.js став стандартом де-факто у сфері JavaScript-розробки на бекенді.

Водночас для створення динамічних сторінок і гнучкої візуалізації даних у вебзастосунках актуальним є використання шаблонізаторів, серед яких Handlebars.js забезпечує простоту, читабельність і підтримку принципу MVC (Model–View–Controller) [4; 5]. Його інтеграція з Express.js дозволяє відокремити логіку від представлення та підвищити зручність супроводу коду.

Таким чином, актуальність обраної теми зумовлена потребою в ефективних, швидких і гнучких засобах розроблення серверних вебзастосунків, які забезпечують динамічний рендеринг даних і зручну взаємодію з клієнтами через REST API.

Дослідження технологій Express.js та Handlebars.js, а також практична реалізація вебзастосунку на їх основі є важливим кроком у підготовці фахівців у галузі веброзробки.

Фреймворк Express.js є одним із найпопулярніших інструментів у середовищі Node.js для створення серверної логіки вебзастосунків. Його головна перевага полягає в мінімалістичності, що дозволяє розробнику самостійно визначати структуру проєкту та використовувати лише ті модулі, які необхідні для конкретного застосунку. Це забезпечує високу гнучкість і масштабованість розробки [2].

Ключовими елементами архітектури Express.js є маршрутизація (routing), middleware, обробка HTTP-запитів і шаблонізація.

Маршрутизація дозволяє налаштовувати різні шляхи доступу до ресурсів за допомогою методів HTTP (GET, POST, PUT, DELETE).

Middleware-функції виступають проміжними обробниками запитів між клієнтом і сервером, забезпечуючи такі процеси, як логування, авторизація, парсинг JSON чи обробка статичних файлів.

Підтримка REST API вбудована безпосередньо у фреймворк, що дозволяє швидко створювати сервери, які віддають відповіді у форматі JSON. Ось приклад базового сервера в Express.js:

```
const express = require('express');
const app = express();

app.use(express.json()); // обробка JSON

app.get('/api/users', (req, res) => {
  res.json([
    { id: 1, name: 'Karina' },
    { id: 2, name: 'Ivan' }
  ]);
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

Рис. 1. Приклад базового сервера в Express.js

Завдяки модульності Express.js може бути основою як для невеликих сайтів, так і для складних систем із мікросервісною архітектурою. На його основі побудовано такі фреймворки, як Nest.js, Sails.js та LoopBack, що свідчить про високу універсальність цієї технології [3].

У сучасних вебзастосунках важливо забезпечити не лише обробку даних, а й зручне представлення інформації для користувача. Для цього використовуються шаблонізатори – інструменти, що дозволяють динамічно формувати HTML-сторінки на основі даних із сервера. Одним із найзручніших засобів для Express є

Handlebars.js, який дотримується принципів MVC (Model–View–Controller) і сприяє розділенню логіки та подання [4]. Підключення Handlebars до Express відбувається таким чином:

```
const express = require('express');
const hbs = require('hbs');
const app = express();

app.set('view engine', 'hbs');
app.set('views', __dirname + '/views');

app.get('/', (req, res) => {
  res.render('index', { name: 'Карина', isAdmin: true });
});
```

Рис. 2. Підключення Handlebars до Express

Ця інтеграція дозволяє створювати динамічні вебсторінки, де дані з REST API безпосередньо відображаються в інтерфейсі користувача. Крім того, Handlebars за замовчуванням екранує HTML, що підвищує рівень безпеки проти XSS-атак [5].

REST API (Representational State Transfer) – це архітектурний стиль, який визначає принципи обміну даними між клієнтом і сервером через стандартні HTTP-запити [1]. Для створення REST API у Node.js фреймворк Express.js надає вбудовані засоби для опису маршрутів, обробки запитів і відправлення відповідей у форматі JSON.

Створення REST API для керування користувачами може мати такий вигляд:

```
app.get('/api/users', (req, res) =>
  res.json([
    { id: 1, name: 'Karina' },
    { id: 2, name: 'Ivan' }
  ]);
});
```

Рис. 3. Створення REST API для керування користувачами

Клієнтська частина або браузер отримує ці дані й може відобразити їх через шаблон Handlebars. Така архітектура забезпечує чітке розділення між бекендом (логіка і дані) та фронтендом (представлення).

Перевагами використання REST API у поєднанні з Express.js і Handlebars є:

– універсальність та легкість інтеграції з іншими застосунками;
– швидкість обробки запитів завдяки оптимізованій Node.js-платформі;
– простота масштабування вебзастосунку;
– зручність роботи з JSON, що є нативним форматом для JavaScript;
– підтримка модульної архітектури.

У результаті реалізації дослідження було створено експериментальний вебзастосунок, який складається з таких компонентів:

– серверна частина (Express.js) – обробляє запити, формує REST API, взаємодіє з базою даних;

– шаблонізатор (Handlebars) – відповідає за візуалізацію сторінок, формування HTML на основі отриманих даних;

– JSON API – забезпечує обмін інформацією між клієнтським інтерфейсом і сервером.

Отриманий результат демонструє, що використання Express.js у поєднанні з Handlebars значно скорочує час розробки, спрощує структуру коду й забезпечує ефективну взаємодію між компонентами системи. Це робить даний підхід доцільним для створення вебзастосунків різного масштабу – від навчальних проєктів до промислових сервісів.

У результаті проведеного дослідження було розглянуто теоретичні та практичні аспекти побудови вебзастосунків на основі фреймворку Express.js та шаблонізатора Handlebars.js з реалізацією REST API. Здійснений аналіз показав, що даний технологічний стек забезпечує високу продуктивність, гнучкість архітектури та зручність у розробленні серверних вебрішень.

У дослідження показано, що використання Express.js як серверної платформи дозволяє створювати масштабовані вебзастосунки з мінімальними витратами ресурсів і спрощеною структурою коду.

Дослідження інтеграції Handlebars.js із фреймворком Express продемонструвало ефективність застосування шаблонізаторів для реалізації принципу MVC (Model–View–Controller), що сприяє чіткому розділенню логіки застосунку та подання даних. Використання Handlebars забезпечує зручну організацію динамічного інтерфейсу користувача, підтримку часткових шаблонів і підвищення безпеки від XSS-атак завдяки автоматичному екрануванню HTML-вмісту.

Практична частина дослідження підтвердила можливість ефективного поєднання Express.js і Handlebars для створення повнофункціонального вебзастосунку з REST API. Реалізований застосунок продемонстрував коректну взаємодію клієнтської та серверної частин, швидкий обмін даними у форматі JSON і зручне представлення результатів у вебінтерфейсі.

Отримані результати свідчать про доцільність використання зв’язки Express.js + Handlebars.js для створення сучасних вебсистем, орієнтованих на інтерактивність, швидкість обробки запитів і простоту розширення. У перспективі подальших досліджень доцільно розглянути інтеграцію цієї архітектури з базами даних MongoDB або PostgreSQL, впровадження механізмів JWT-аутентифікації, а також розширення застосунку засобами React або Vue.js для реалізації повного стеку MERN/MEHN.

Список використаних джерел

1. REST API Tutorial: What is REST? URL: https://restfulapi.net/ (дата звернення: 04.11.2025).

2. Express.js: Official Documentation URL: https://expressjs.com/ (дата звернення: 04.11.2025).

3. ARM Community. *Node.js and Express in Modern Backend Development.* URL: https://community.arm.com/ (дата звернення: 04.11.2025).

4. Handlebars.js: Official Documentation. URL: https://handlebarsjs.com/ (дата звернення: 04.11.2025).

5. Node.js Foundation. *Integrating Templating Engines with Express* URL: https://nodejs.org/en/docs/ (дата звернення: 04.11.2025).