УДК 004.4:004.2]:004.6
DOI https://doi.org/10.32782/2663-5941/2025.6.2/51

Удод О.С.
Запорізький національний університет

Кривохата А.Г.
Запорізький національний університет

РОЗРОБКА ВИСОКОМАСШТАБОВАНОЇ БЕЗСЕРВЕРНОЇ ХМАРНОЇ АРХІТЕКТУРИ СИСТЕМИ УПРАВЛІННЯ ЗАВДАННЯМИ

У статті розглянуто актуальну науково-прикладну задачу підвищення ефективності розробки та експлуатації систем управління завданнями в умовах динамічних та нерівномірних навантажень. Проаналізовано недоліки традиційних монолітних та серверних архітектур, пов'язані з обмеженою масштабованістю та високими операційними витратами на утримання інфраструктури. Запропоновано використання концепції безсерверних обчислень (Serverless Computing) та моделі «функція як послуга» (FaaS). Даний підхід, на відміну від існуючих рішень, дозволяє оптимізувати використання ресурсів та забезпечити автоматичне горизонтальне масштабування.

Для визначення вимог здійснено порівняльний аналіз провідних комерційних (Wrike, JIRA) та відкритих (OpenProj) аналогів. Комплекс функціональних та нефункціональних вимог до системи сформульовано з акцентом на показники продуктивності, безпеки даних та відмовостійкості. Розроблено набір UML-діаграм (прецедентів, компонентів, розгортання), які ілюструють структуру розподіленого вебзастосунку.

Обґрунтовано вибір технологічного стека: фреймворк Angular для реалізації клієнтської частини (Single Page Application) та екосистеми Amazon Web Services (Lambda Node.js, API Gateway, RDS PostgreSQL) для бекенду. Особливу увагу приділено питанням інформаційної безпеки: впроваджено механізм захищеного управління обліковими даними через AWS Secrets Manager та stateless-автентифікацію.

Емпірична верифікація розробленого рішення проведена шляхом багаторівневого тестування. Застосування модульних тестів (Jest) та навантажувального тестування (Artillery) дозволило оцінити поведінку системи під стресовим навантаженням. Результати експерименту підтвердили, що запропонована архітектура забезпечує стабільну обробку до 20 запитів на секунду при збереженні стабільних часових характеристик. Отримані дані свідчать про те, що розроблена система є ефективним, економічно вигідним та масштабованим рішенням, готовим до впровадження в корпоративному середовищі для автоматизації проєктного менеджменту.

Ключові слова: *комп'ютерні інформаційні технології, система управління завданнями, безсерверна архітектура, Angular, AWS Lambda, PostgreSQL, API Gateway, JWT, навантажувальне тестування.*

Постановка проблеми. У сучасному цифровому середовищі системи управління завданнями є критично важливими для забезпечення ефективності командної роботи та організації бізнес-процесів. У сфері розробки програмного забезпечення та управління проєктами спостерігається значне розгалуження подібних систем. Це призвело до появи значної кількості різноманітних систем, кожна з яких пропонує свій унікальний набір функцій, підтримуваних методологій (Agile, Kanban, Waterfall) та власний підхід до організації робочого процесу (від простих одноріневих до комплексних та декомпозованих багаторівневих задач). Це розгалуження створює гостру необхідність детального аналізу існуючих варіантів, щоб зрозуміти їхні функціональні та нефункціональні можливості, особливості інтеграції, масштабованість та придатність до різних типів команд і проєктів, які суттєво відрізняються за своєю спрямованістю, складністю управління та специфікою бізнес-процесів.

Зі зростанням обсягів даних, збільшенням кількості користувачів та динамічністю робочих навантажень, ключовими вимогами до систем управління завданнями стають висока масштабованість, надійність та економічна ефективність експлуатації. Традиційні монолітні та серверні архітектури мають суттєві обмеження для систем

управління завданнями з динамічними робочими навантаженнями, що проявляється у таких ключових недоліках:

1. Неефективне масштабування. Складнощі у швидкому масштабуванні інфраструктури під час пікових навантажень та, навпаки, надлишкові ресурси та неефективні витрати у періоди низької активності, що є критичним для систем, які обслуговують різнорівневі робочі процеси.

2. Операційні витрати. Значні витрати часу та ресурсів на адміністрування, оновлення та обслуговування серверів.

Концепція безсерверних обчислень позиціонується як новий та сучасний підхід, що може забезпечити оптимальне рішення для підтримки різноманітних та динамічних робочих процесів систем управління завданнями, мінімізуючи операційні витрати [1].

Виходячи з існуючої проблеми, метою даного дослідження є розробка, імплементація та експериментальна оцінка ефективності високомасштабованої Serverless-архітектури системи управління завданнями, що забезпечить оптимальне співвідношення між продуктивністю, гнучкістю підтримки складних робочих процесів та економічною ефективністю експлуатації.

Аналіз останніх досліджень і публікацій. У межах дослідження здійснено порівняльний аналіз функціональних особливостей існуючих аналогів проєктованої системи. До переліку провідних рішень на ринку, які є об'єктом подальшого аналізу, включено: Wrike, Worksection, Asana, Trello, Microsoft Planner, JIRA, Microsoft Project та OpenProj [2, 3]. Детальний огляд деяких платформ представлено нижче.

Trello – це хмарна платформа, що імплементує методологію Kanban для візуалізації робочих процесів та відстеження завдань. Система використовує структуру дошок, списків та карток для організації завдань, забезпечуючи високу гнучкість та низький поріг входження. Архітектура Trello орієнтована на прості та гнучкі робочі процеси та ефективну командну співпрацю. Проте, аналіз функціональних можливостей виявляє суттєві обмеження для комплексного проєктного менеджменту: відсутність нативних підзадач та управління залежностями, слабкі інструменти звітності та недостатня масштабованість для великих, багатокомпонентних корпоративних проєктів [2, 4].

Wrike позиціонується як комплексна платформа для автоматизації повного життєвого циклу управління проєктами, орієнтована на оптимізацію робочих процесів та розподіл ресурсів. Архітектура системи забезпечує гнучку адаптацію під специфічні бізнес-вимоги завдяки широким можливостям кастомізації та розвиненим механізмам інтеграції із зовнішніми сервісами. До ключових переваг платформи належать універсальність, розвинений інструментарій для управління залежностями завдань та засоби узгодження етапів роботи. Водночас, аналіз експлуатаційних характеристик виявив низку суттєвих обмежень: висока вартість ліцензування (стартова ціна становить близько $10 за користувача/місяць), функціональна надлишковість, що ускладнює інтерфейс для малих команд, та суворі ліміти безкоштовної версії, що знижує доступність системи для стартапів та бюджетних проєктів [4, 5].

Jira (Atlassian) є спеціалізованим інструментом для автоматизації життєвого циклу розробки програмного забезпечення, орієнтованим на використання методологій Agile. Функціональна архітектура платформи забезпечує глибоку інтеграцію з системами контролю версій та CI/CD (Bitbucket, Jenkins), а також гнучке моделювання робочих процесів. Попри розвинені можливості трекінгу дефектів, впровадження системи ускладнюється високим порогом входження через складність адміністрування та значною вартістю масштабування. Крім того, функціональна надлишковість робить її використання неефективним для малих команд із обмеженими ресурсами. [6].

OpenProj – це десктопне рішення з відкритим вихідним кодом, що позиціонується як безкоштовна альтернатива пропрієтарним системам (зокрема, MS Project). Функціонал системи охоплює класичні інструменти проєктного менеджменту: діаграми Ганта, мережеві графіки та структуру декомпозиції робіт. Ключовими перевагами є відсутність ліцензійних витрат та повна сумісність форматів даних із MS Project. Проте впровадження OpenProj у сучасних розподілених командах обмежене через відсутність механізмів хмарної синхронізації та інструментів для спільної роботи. До суттєвих недоліків також належать застарілий інтерфейс, відсутність технічної підтримки та проблеми стабільності при обробці великих масивів даних. Крім того, виявлено складнощі із кросплатформенністю в середовищах, відмінних від Windows [7].

Отже, висока насиченість ринку систем управління завданнями створює умови для гнучкого вибору інструментів залежно від специфіки проєктної діяльності. Проте, наявність значної кількості платформ із різними архітектурними та функціональними особливостями вимагає від

організацій ретельного попереднього аудиту для мінімізації ризиків та оптимізації співвідношення ціни та якості.

Постановка завдання. Метою статті є проєктування, програмна реалізація та експериментальна оцінка системи управління завданнями побудованої на базі безсерверної архітектури з використанням стека AWS Lambda та Angular, а також обґрунтування її переваг порівняно з традиційними монолітними або мікросервісними підходами.

Виклад основного матеріалу. При розробці фронтенд частини програмного забезпечення вибір фреймворку відіграє критичну роль у забезпеченні ефективності, масштабованості та зручності використання продукту. Після ретельного аналізу наявних рішень як основну технологію розробки клієнтської частини системи управління задачами було обрано Angular, через його комплексну архітектуру, вбудовані інструменти та TypeScript-інтеграції, які забезпечують необхідну структурованість, безпеку та масштабованість для реалізації функціональних вимог системи. [8, 9, 10].

Для реалізації серверної частини системи управління задачами було проведено аналіз хмарних сервісів Amazon Web Services (AWS), які надають можливості для створення масштабованих, безпечних, та економічно ефективних рішень. При розробці системи управління завданнями, найактивніше було залучено такі сервіси: AWS Lambda; Amazon VPC; Amazon EC2; RDS PostgreSQL; AWS Secrets Manager; Amazon API Gateway [11-15].

Важливим етапом проєктування та імплементації програмного продукту є проведення детального аналізу вимог. Для системи управління завданнями «TaskManager» ключовий функціонал охоплює повний життєвий цикл завдання (від створення до завершення) та підтримку ефективної командної взаємодії. Основні функціональні вимоги включають:

1. Управління об’єктами. Створення, редагування та архівування завдань з підтримкою багаторівневої категоризації (за проєктами та типами) та можливістю встановлення пріоритетів та термінів виконання.

2. Спільна робота. Механізми призначення завдань конкретним виконавцям, система нагадувань та сповіщень про зміни статусу, коментування та обмін файлами у форматі чату.

3. Візуалізація та звітність. Відстеження прогресу виконання завдань у режимі реального часу, підтримка різних способів візуалізації даних (наприклад, Kanban-дошки або списковий формат), а також генерування деталізованих звітів з використанням параметричних фільтрів.

4. Інтеграція та безпека. Інтеграція з календарними сервісами для синхронізації термінів та реалізація механізмів контролю доступу (автентифікація та авторизація) для розмежування прав користувачів.

Особлива увага приділяється створенню інтуїтивного інтерфейсу для швидкого створення та модифікації завдань, що є критичним фактором для підвищення продуктивності користувачів.

Нефункціональні вимоги визначають якісні характеристики системи, які є визначальними для її архітектури, експлуатації та сприйняття користувачем. Категоріально вони систематизовані наступним чином:

1. Продуктивність. Система повинна забезпечувати час відгуку не більше 2 секунд для 95% операцій, навіть при піковому навантаженні, підтримуючи одночасну роботу не менше 20 користувачів.

2. Надійність та Доступність. Забезпечення доступності системи на рівні не менше 99.5% часу, з мінімальними плановими простоями.

3. Масштабованість. Архітектура повинна підтримувати горизонтальне масштабування для коректної обробки зростаючої кількості даних та користувачів без необхідності повного перепроєктування.

4. Безпека. Захист даних через шифрування, аудит дій користувачів та забезпечення захисту від типових кібератак.

5. Зручність та адаптивність. Інтерфейс повинен відповідати сучасним принципам UX/UI дизайну, бути інтуїтивно зрозумілим і забезпечувати коректне відображення на різних пристроях.

На рисунку 1 представлена діаграма варіантів використання, що визначає взаємодію двох ключових акторів: Адміністратора проєкту та Користувача. Адміністратор володіє розширеними дозволами, які охоплюють повний цикл управління проєктами (створення, зміна, видалення) та управління користувацькими завданнями. Авторизований користувач має доступ до повного життєвого циклу управління завданнями (CRUD-операції), включаючи встановлення ключових атрибутів (назва, пріоритет, дедлайн) та організацію завдань у папки. Система також забезпечує необхідні процеси автентифікації/авторизації та комунікаційний функціонал – додавання коментарів із підтримкою текстових та медіавкладень.

На рисунку 2 зображено IDEF1X діаграму бази даних, що складається з 11 сутностей. Центральна сутність Tasks пов'язана з проєктами (Projects), категоріями, статусами та пріоритетами. Управління користувачами (Users) та їхніми ролями (Roles) реалізовано через проміжну сутність Task_Assignees, що забезпечує зв'язок «багато до багатьох» між виконавцями та завданнями. Функціонал обговорення забезпечують сутності Comments та Media (з типами у Content_types). Така структура підтримує категоризацію, розподіл завдань та обмін файлами.

Представлена діаграма компонентів (рис. 3) фронтенд-частини системи, розробленої на базі Angular, відображає модульну архітектуру додатку. Ядром системи є компоненти Project та Task, взаємодія між якими забезпечується через Router. Компонент Task декомпозовано на під-

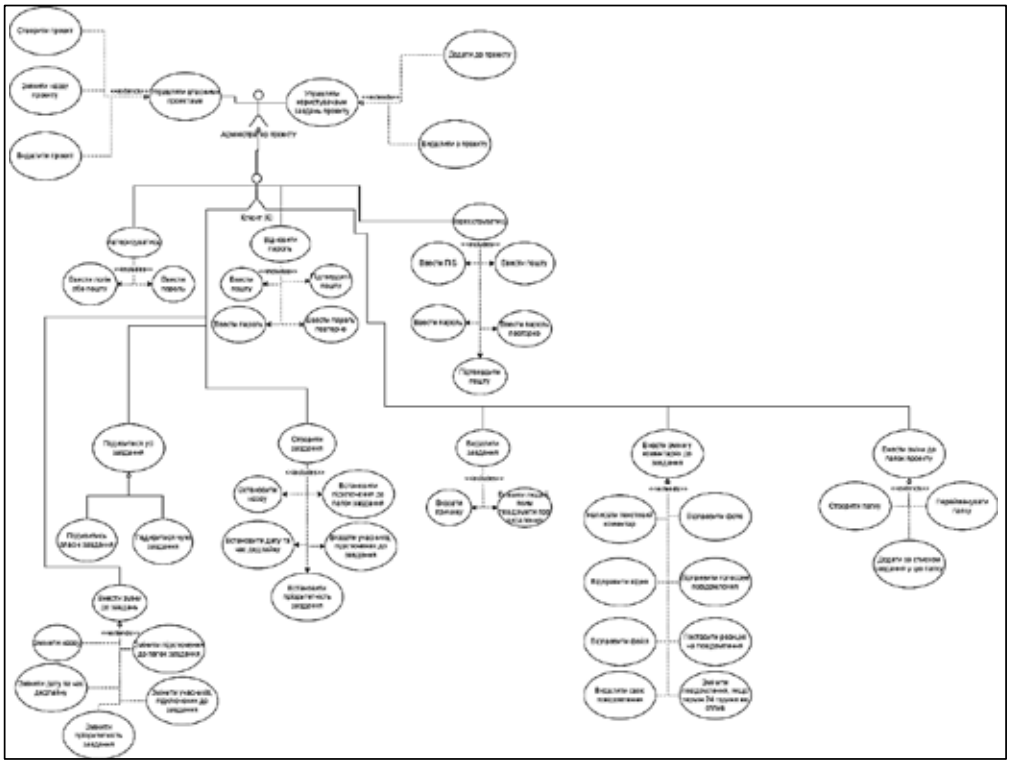


Рис. 1. Діаграма варіантів використання

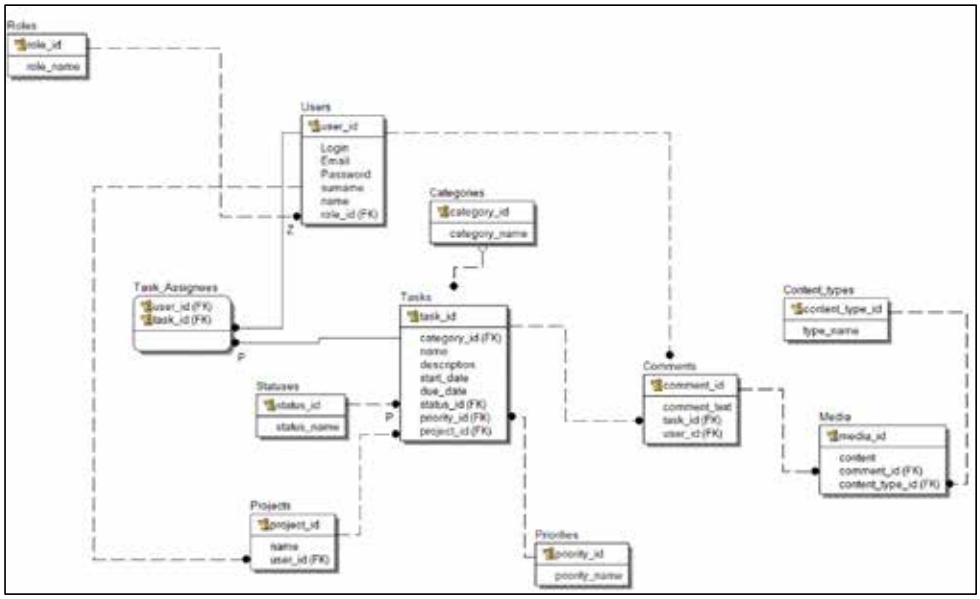


Рис. 2. IDEF1X діаграма системи управління завданнями

модулі: TaskBoard (візуалізація kanban-дошки) та TaskCreator (створення/редагування), що комунікують через систему подій.

Взаємодію з хмарною інфраструктурою AWS Lambda абстраговано через сервісний шар TaskService, що оперує типізованими моделями даних TaskModel, UserModel. Це дозволяє інкапсулювати логіку API-запитів та ефективно застосовувати патерн впровадження залежностей (Dependency Injection). Допоміжні модулі Shared, Admin, Chart забезпечують навігацію, адміністрування та аналітику. Така архітектурна організація сприяє слабкій зв'язаності коду, спрощує підтримку системи та забезпечує виконання вимог щодо продуктивності та масштабованості.

На рисунку 4 представлено реалізацію компонента app-login.ts, що забезпечує механізм авторизації на базі Angular Reactive Forms. Використання реактивних форм дозволяє реалізувати сувору валідацію вхідних даних (формат email, обов'язковість полів) на клієнтському боці. Логіка взаємодії з сервером інкапсульована в сервісі AuthService, який обробляє запити автентифікації, тоді як управління навігацією після успішного входу делеговано модулю Router. Такий підхід забезпечує модульність коду та централізовану обробку бізнес-логіки.

На рисунку 5 зображено TaskCreatorComponent, що забезпечує інтерфейс створення завдань засобами Angular Reactive Forms. Ініціалізація форми здійснюється через FormBuilder із встановленням правил валідації для полів title, priority та status. Архітектура компонента реалізована за принципом «Smart/Dumb components»: він не виконує збереження даних самостійно, а делегує цю операцію батьківському контролеру через механізм подій (@Output, EventEmitter).

Взаємодія з бекендом побудована на безсерверній архітектурі. Клієнтський додаток надсилає запити до Amazon API Gateway, який маршрутизує їх до відповідних функцій AWS Lambda. Така модель дозволяє інкапсулювати бізнес-логіку та операції з базою даних у масштабовані функції, що виконуються виключно за вимогою, забезпечуючи оптимізацію ресурсів.

На рисунку 6 представлено реалізацію Lambda-функції для отримання даних про зв'язки між користувачами та проєктами (project_assignees). У межах безсерверної архітектури функція є бекенд-обробником, що ініціює з'єднання з PostgreSQL. Безпека доступу до бази даних забезпечується інтеграцією з AWS Secrets Manager, який надає необхідні облікові дані без їх жорсткого кодування в тілі функції.

Алгоритм роботи передбачає виконання SQL-запиту на вибірку та серіалізацію отриманих результатів у формат HTTP-відповіді (JSON). Програмна реалізація включає механізми обробки

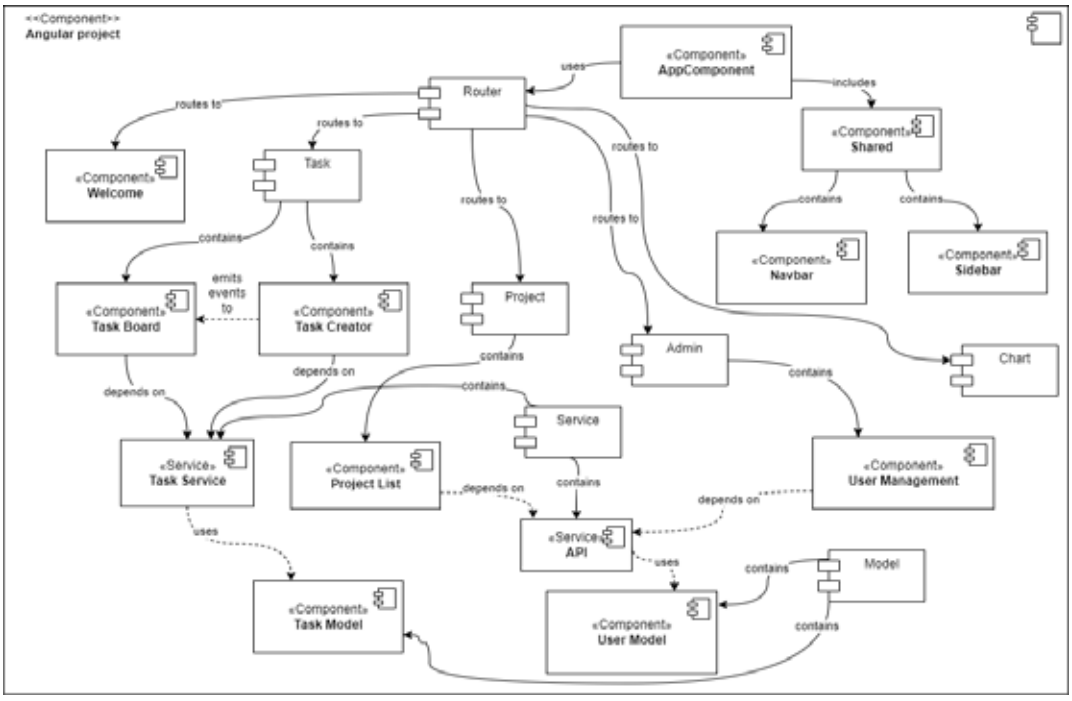
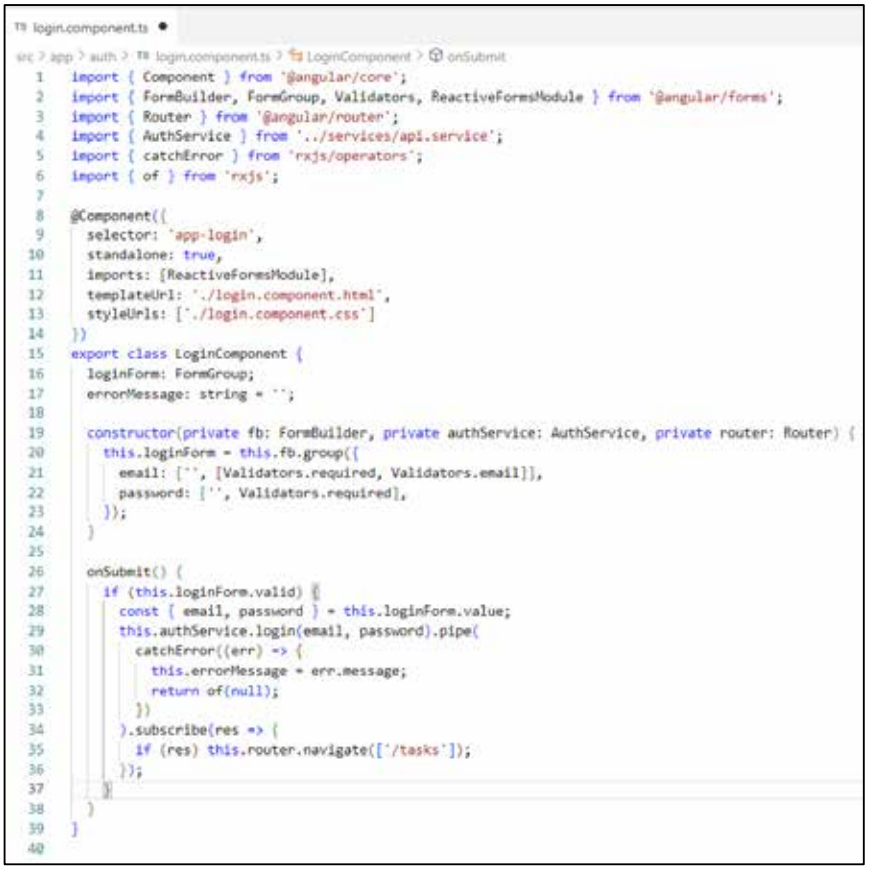


Рис. 3. Діаграма компонентів фронтенд-частини застосунку



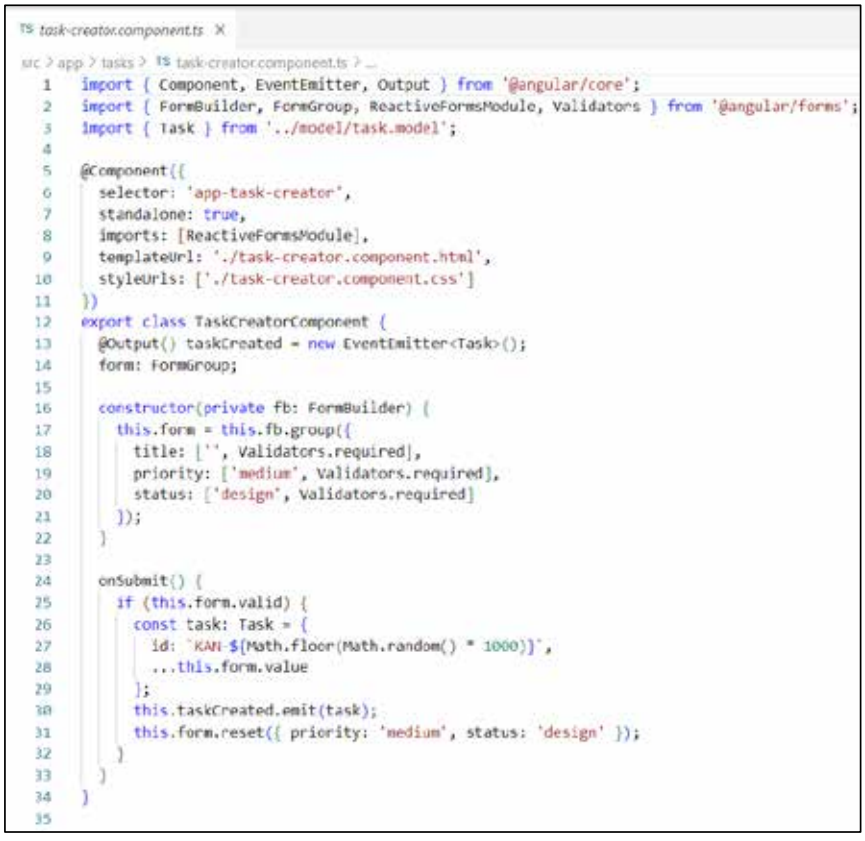
```ts
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from '../services/api.service';
import { catchError } from 'rxjs/operators';
import { of } from 'rxjs';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  loginForm: FormGroup;
  errorMessage: string = '';

  constructor(private fb: FormBuilder, private authService: AuthService, private router: Router) {
    this.loginForm = this.fb.group({
      email: ['', [Validators.required, Validators.email]],
      password: ['', Validators.required],
    });
  }

  onSubmit() {
    if (this.loginForm.valid) {
      const { email, password } = this.loginForm.value;
      this.authService.login(email, password).pipe(
        catchError((err) => {
          this.errorMessage = err.message;
          return of(null);
        })
      ).subscribe(res => {
        if (res) this.router.navigate(['/tasks']);
      });
    }
  }
}
```

Рис. 4. Компонент app-login.ts



```ts
import { Component, EventEmitter, Output } from '@angular/core';
import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';
import { Task } from '../model/task.model';

@Component({
  selector: 'app-task-creator',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './task-creator.component.html',
  styleUrls: ['./task-creator.component.css']
})
export class TaskCreatorComponent {
  @Output() taskCreated = new EventEmitter<Task>();
  form: FormGroup;

  constructor(private fb: FormBuilder) {
    this.form = this.fb.group({
      title: ['', Validators.required],
      priority: ['medium', Validators.required],
      status: ['design', Validators.required]
    });
  }

  onSubmit() {
    if (this.form.valid) {
      const task: Task = {
        id: `KAN-${Math.floor(Math.random() * 1000)}`,
        ...this.form.value
      };
      this.taskCreated.emit(task);
      this.form.reset({ priority: 'medium', status: 'design' });
    }
  }
}
```

Рис. 5. Компонент TaskCreatorComponent

```
index.mjs
import { SecretsManagerClient, GetSecretValueCommand } from "@aws-sdk/client-secrets-manager";
import pkg from "pg";
import AWS from "aws-sdk";

const SM = new AWS.SecretsManager();
const { Client } = pkg;

// Configure the Secrets Manager client
const secretsManagerClient = new SecretsManagerClient({
  region: "eu-north-1",
});

export const handler = async (event) => {
  const dbPasswordSecretName = "rds!db-7946976a-87d4-40fd-b3e8-d92cc8949760";
  const dbCredentialsSecretName = "udod/task-db/credentials";

  try {
    // Fetch and parse admin password secret
    const secretDataPassword = await SM.getSecretValue({ SecretId: dbPasswordSecretName }).promise();
    const adminPasswordData = JSON.parse(secretDataPassword.SecretString);

    const adminUsername = adminPasswordData.username;
    const adminPassword = adminPasswordData.password;

    // Fetch and parse database credentials secret
    const secretDataCredentials = await SM.getSecretValue({ SecretId: dbCredentialsSecretName }).promise();
    const dbCredentialsData = JSON.parse(secretDataCredentials.SecretString);

    const client = new Client({
      host: dbCredentialsData.DB_HOST,
      port: dbCredentialsData.DB_PORT,
      database: dbCredentialsData.DB_NAME,
      user: dbCredentialsData.DB_USER,
      password: adminPassword,
      ssl: {
        rejectUnauthorized: false,
      },
    });
        try {
            await client.connect();
            const query = 'SELECT user_id, project_id FROM project_assignees;';
            const result = await client.query(query);
            console.log("Data from 'project_assignees' was get successfully [GET ALL FROM PROJECT_ASSIGNEES]")

            if (result.rows.length === 0) {
                return {
```

Рис. 6. Lambda-функція для отримання даних з PostgreSQL

```
index.mjs
import { SecretsManagerClient, GetSecretValueCommand } from "@aws-sdk/client-secrets-manager";
import pkg from "pg";
import AWS from "aws-sdk";   Amazon Q Tip 1/3: Start typing to get suggestions (ESC to exit)
import jwt from "jsonwebtoken";
import bcrypt from "bcryptjs";

const SM = new AWS.SecretsManager();
const { Client } = pkg;

const secretsManagerClient = new SecretsManagerClient({ region: "eu-north-1" });

export const handler = async (event) => {
  const dbPasswordSecretName = "rds!db-7946976a-87d4-40fd-b3e8-d92cc8949760";
  const dbCredentialsSecretName = "udod/task-db/credentials";
  const jwtSecretName = "udod/task-api/jwt-secret";

  try {
    const secretDataPassword = await SM.getSecretValue({ SecretId: dbPasswordSecretName }).promise();
    const adminPasswordData = JSON.parse(secretDataPassword.SecretString);
    const adminPassword = adminPasswordData.password;

    const secretDataCredentials = await SM.getSecretValue({ SecretId: dbCredentialsSecretName }).promise();
    const dbCredentialsData = JSON.parse(secretDataCredentials.SecretString);

    const secretJwt = await SM.getSecretValue({ SecretId: jwtSecretName }).promise();
    const jwtSecret = JSON.parse(secretJwt.SecretString).JWT_SECRET;

    const { login, password } = JSON.parse(event.body);

    const client = new Client({
      host: dbCredentialsData.DB_HOST,
      port: dbCredentialsData.DB_PORT,
      database: dbCredentialsData.DB_NAME,
      user: dbCredentialsData.DB_USER,
      password: adminPassword,
      ssl: { rejectUnauthorized: false },
    });

    await client.connect();

    const query = 'SELECT user_id, login, password, role_id FROM users WHERE login = $1';
```

Рис. 7. Lambda-функція для авторизації і автентифікації користувачів

виключних ситуацій, що перехоплюють помилки підключення, збої служби секретів та внутрішні помилки виконання, забезпечуючи стабільність системи.

Система безпеки базується на використанні протоколу JWT (JSON Web Token), що забезпечує stateless-взаємодію між Angular-клієнтом та хмарним бекендом. На рисунку 7 представлено реалізацію Lambda-функції автентифікації, алгоритм роботи якої включає три ключові етапи.

Перший етап – ініціалізація, що передбачає отримання реквізитів доступу до PostgreSQL та ключів підпису JWT через захищене сховище AWS Secrets Manager. Другий етап – верифікація, під час якої здійснюється пошук користувача в базі даних та перевірка валідності пароля за допомогою алгоритму хешування bcrypt. Завершальним етапом є генерація підписаного токена, що містить рольові атрибути та термін дії сесії. Така архітектура гарантує ізоляцію конфіденційних даних та відповідність сучасним стандартам безпеки вебзастосунків.

Апробацію розробленого рішення здійснено за комплексною стратегією. Модульне та інтеграційне тестування компонентів Angular і AWS Lambda реалізовано з використанням фреймворків Jest та Mocha, а валідацію API-інтерфейсів – засобами Postman. Для оцінки нефункціональних вимог проведено навантажувальне тестування за допомогою інструменту Artillery.

Результати випробувань продуктивності наведено на рисунку 8. Аналіз графіків пропускної здатності та часу відгуку свідчить про стабільність системи під навантаженням: протягом тестового інтервалу (60 с) зафіксовано стійку обробку запитів з інтенсивністю 20 RPS при збереженні нормативних показників латентності. Отримані дані підтверджують ефективність обраної архітектури та її здатність до масштабування.

На рисунку 9 наведено реалізовану дошку задач проєкту. Ліва частина інтерфейсу містить навігаційне меню, яке дозволяє швидко переходити між основними розділами проєкту: головна, завдання, issue boards, код, тестування, аналітика тощо.

Центральна частина інтерфейсу поділена на три стовпці: Open, In Progress та Closed, які представляють поточний стан задач у проєкті. Кожне

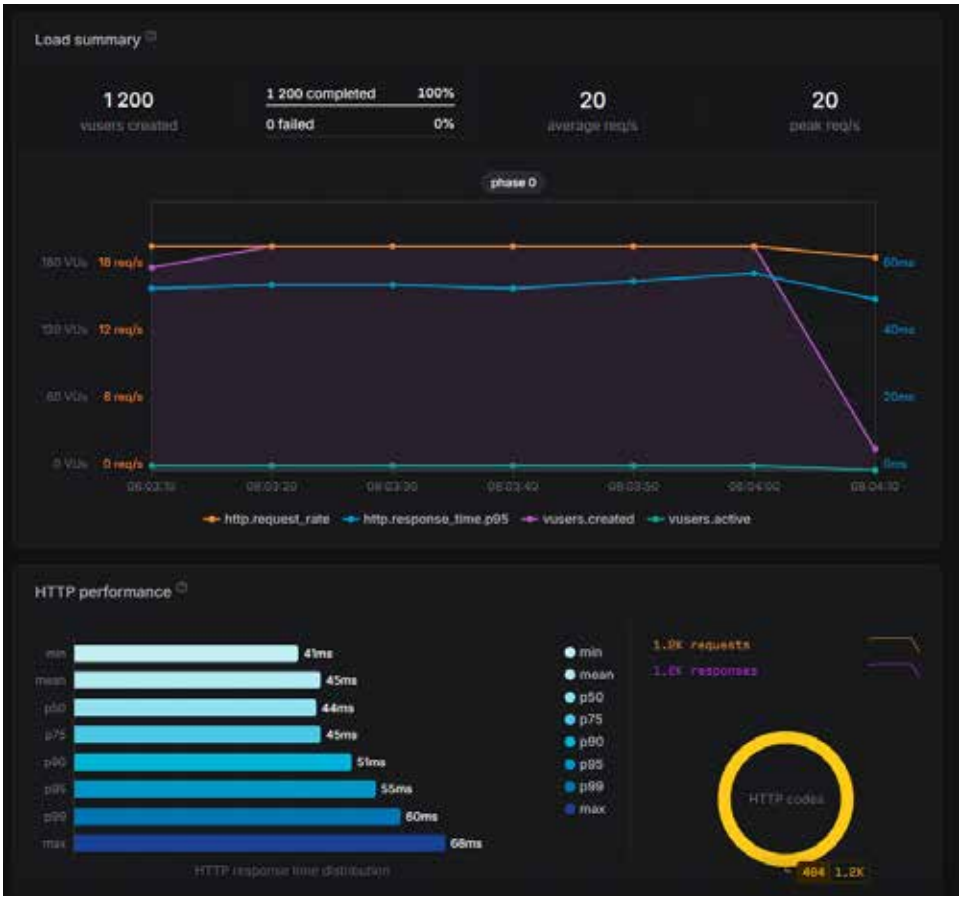


Рис. 8. Тестування навантажень

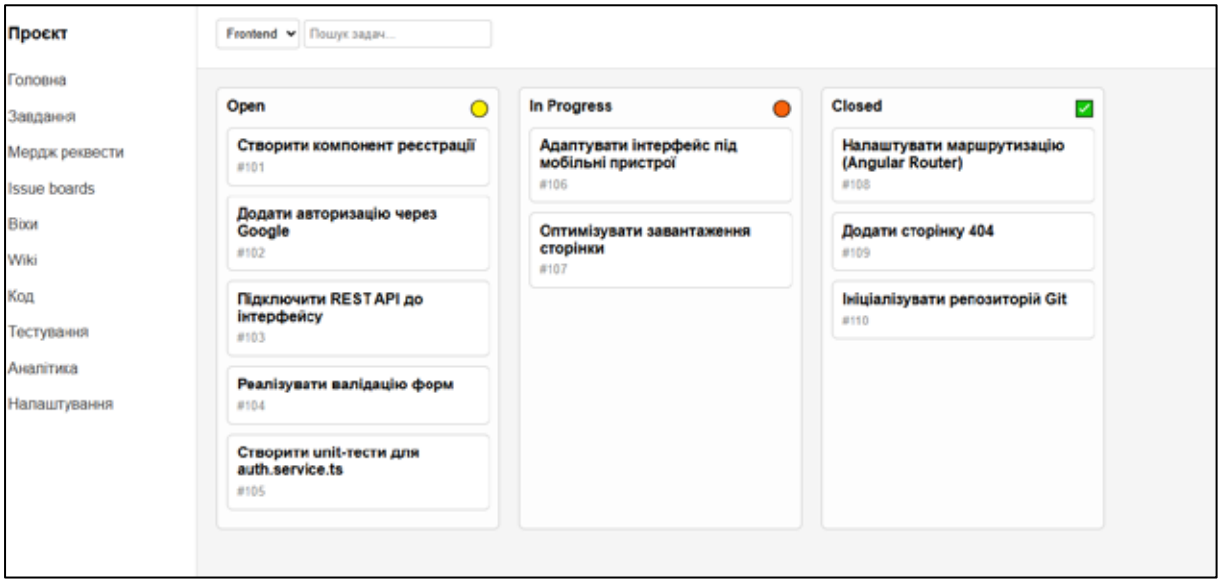


Рис. 9. Дошка завдань

завдання оформлено окремою карткою з коротким описом та унікальним номером. Наприклад, у стовпці «Open» є завдання «Створити компонент реєстрації», а в стовпці «Closed» – завершене завдання «Налаштувати маршрутизацію (Angular Router)». Такий формат дозволяє ефективно візуалізувати робочий процес команди та контролювати виконання задач на різних етапах розробки.

Висновки. У роботі представлено розробку системи управління завданнями на основі безсерверної архітектури. Аналіз аналогів та функціональних вимог дозволив обґрунтувати вибір стека технологій Angular та AWS Lambda як оптимального рішення для забезпечення масштабованості та мінімізації експлуатаційних витрат. У рамках дослідження спроєктовано та реалізовано надійну архітектуру із використанням сервісів AWS (RDS, API Gateway, Secrets Manager), що забезпечує високу доступність даних. Експериментальна перевірка, проведена шляхом навантажувального тестування, підтвердила стабільність роботи системи при інтенсивності 20 запитів за секунду при збереженні стабільних часових характеристик. Отримані результати свідчать про те, що запропонований підхід дозволяє створювати ефективні, відмовостійкі та економічно вигідні системи управління проєктами. Подальший розвиток системи передбачає інтеграцію алгоритмів машинного навчання для автоматизованого розподілу навантаження та прогнозування термінів виконання завдань. Крім того, перспективним напрямком є впровадження технології WebSockets для забезпечення синхронізації даних між учасниками проєкту в режимі реального часу.

Список літератури:

1. Kosur P. Optimizing Scalability and Performance in AWS Lambda: An InDepth Analysis of Best Practices, Case Studies, and Challenges in Serverless Architectures. *Journal of Engineering and Applied Sciences Technology*. 2023. Volume 5(3). P. 1–3. DOI: 10.47363/JEAST/2023(5)E115

2. Топ 10 систем управління проектами для України. URL: https://www.livebusiness.com.ua/ua/tools/pm/ (дата звернення: 11.11.2025).

3. 25 Best Task Management Software Reviewed For 2025. URL: https://thedigitalprojectmanager.com/tools/best-task-management-software/ (дата звернення: 10.11.2025).

4. Pawłowski P., Plechawska-Wójcik M. A comparative analysis of tools dedicated to project management. *Journal of Computer Sciences Institute*. 2022. №24. P. 258–264. DOI: 10.35784/jcsi.3001

5. Day B. Wrike Review: Features, Pros And Cons. *Forbes Advisor*. URL: https://www.forbes.com/advisor/business/software/wrike-review/ (дата звернення: 10.11.2025).

6. Tammy. Jira Pros and Cons in 2025: Features, Pricing & User Reviews. *ONES.com* | Advanced software development management platform for enterprise. URL: https://ones.com/blog/jira-pros-and-cons-in-2025 (дата звернення: 10.11.2025).

7. OpenProj – Project Management. *SourceForge*. URL: https://sourceforge.net/projects/openproj/ (дата звернення: 10.11.2025).

8. What is Angular? URL: https://angular.dev/overview (дата звернення: 10.11.2025).

9. Розуміння зв'язування даних в Angular: Ключ до динамічних веб-додатків – javascript.org.ua – JS Communities. *javascript.org.ua – JS Communities*. URL: https://javascript.org.ua/rozuminnya-zvyazuvannya-danih-v-angular-klyuch-do-dinamichnih-veb-dodatkiv/ (дата звернення: 10.11.2025).

10. Shields K. Angular Development Pros and Cons: Is It Right for Your Web Application. Mobile App & Web Development Greenville, SC | *Designli*. URL: https://designli.co/blog/angular-development-web-application (дата звернення: 10.11.2025).

11. AWS Lambda: Serverless Computing Explained – AWS. URL: https://aws.amazon.com/awstv/watch/64bb50cff05/ (дата звернення: 10.11.2025).

12. What is Amazon VPC? – Amazon Virtual Private Cloud. URL: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html (дата звернення: 10.11.2025).

13. What is Amazon EC2? – Amazon Elastic Compute Cloud. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (дата звернення: 10.11.2025).

14. Hosted PostgreSQL – Amazon RDS for PostgreSQL – AWS. *Amazon Web Services, Inc*. URL: https://aws.amazon.com/rds/postgresql/ (дата звернення: 10.11.2025).

15. Welcome to AWS Documentation. URL: https://docs.aws.amazon.com/ (дата звернення: 10.11.2025).

Udod O.S., Kryvokhata A.G. DEVELOPMENT OF A HIGHLY SCALABLE SERVERLESS CLOUD ARCHITECTURE FOR A TASK MANAGEMENT SYSTEM

The article addresses the relevant scientific and applied problem of improving the development and operational efficiency of task management systems under conditions of dynamic and fluctuating workloads. The shortcomings of traditional monolithic and server-based architectures, associated with limited scalability and high infrastructure maintenance costs, are analyzed. The utilization of the Serverless Computing concept and the Function as a Service (FaaS) model is proposed. This approach, unlike existing solutions, enables resource optimization and automatic horizontal scaling.

To define the requirements, a comparative analysis of leading commercial (Wrike, JIRA) and open-source (OpenProj) analogs was conducted. A set of functional and non-functional requirements for the system was formulated, with a focus on performance, data security, and fault tolerance indicators. A set of UML diagrams (use case, component, and deployment diagrams) was developed to illustrate the structure of the distributed web application.

The choice of the technology stack is substantiated: the Angular framework for the client-side implementation (Single Page Application) and the Amazon Web Services ecosystem (Lambda Node.js, API Gateway, RDS PostgreSQL) for the backend. Particular attention is paid to information security issues: a mechanism for secure credential management via AWS Secrets Manager and stateless authentication has been implemented.

Empirical verification of the developed solution was conducted through multi-level testing. The application of unit tests (Jest) and load testing (Artillery) enabled the assessment of system behavior under stress loads. Experimental results confirmed that the proposed architecture ensures stable processing of up to 20 requests per second while maintaining stable timing characteristics. The obtained data indicate that the developed system is an effective, cost-efficient, and scalable solution, ready for implementation in a corporate environment for project management automation.

Key words: *computer information technologies, task management system, serverless architecture, Angular, AWS Lambda, PostgreSQL, API Gateway, JWT, load testing.*

Дата надходження статті: 28.11.2025
Дата прийняття статті: 17.12.2025
Опубліковано: 30.12.2025