УДК 004.27

ПРОЕКТУВАННЯ АРХІТЕКТУРИ ІНТЕРНЕТ-МАГАЗИНУ

Будьонний М.А., студент, m.budonnyy.fit.122.20@knute.edu.ua, ДТЕУ

Протягом останніх років веб-додатки пройшли значні трансформації в архітектурному плані, з'явилися нові підходи та технології. Одним із ключових виборів, перед якими стоять розробники, є вибір архітектурного підходу, оскільки це має критичний вплив на продуктивність додатка, його масштабованість та зручність обслуговування.

На даний момент існують три основні архітектурні підходи, які користуються найбільшою популярністю: монолітна архітектура, мікросервісна архітектура, та «serverless» архітектура. Кожен з цих підходів має свої унікальні переваги та недоліки, і вибір архітектури залежить від специфічних вимог та обмежень проекту. Монолітна архітектура, яка представляє собою класичний підхід, заснований на однорідному блоку, є простою у розробці та впровадженні [1]. Це спрощує процеси налагодження та масштабування, забезпечуючи безперервну взаємодію між різними частинами додатку без потреби в інтеграції з зовнішніми сервісами. Однак, з часом великі монолітні додатки можуть стати складними для розуміння та обслуговування.

Архітектура мікросервісів розділяє додаток на множину малих, незалежних сервісів, що сприяє легкому масштабуванню та спрощує внесення змін. Цей підхід забезпечує гнучкість у розвитку окремих частин додатку, але може ускладнити управління сервісами та їх взаємодію. Кожен мікросервіс є самостійною одиницею, що дозволяє їх масштабувати незалежно один від одного, спрощуючи роботу з великим обсягом трафіку або навантаженням. Крім того, мікросервіси можуть бути написані різними мовами програмування та використовувати різні технології, що забезпечує гнучкість вибору для розробників.

Архітектура «serverless» дозволяє розробникам зосередитися на коді, використовуючи хмарні сервіси для автоматичного розгортання та управління інфраструктурою. Це усуває потребу в ручному конфігуруванні серверів, спрощуючи розробку та знижуючи витрати на обслуговування. Однак, можуть виникнути обмеження щодо сумісності технологій та управління витратами. Використання безсерверних платформ, таких як AWS Lambda, Azure Functions, чи Google Cloud Functions, відкриває шлях до більш ефективної та масштабованої розробки веб-додатків. «Serverless» архітектура вирізняється економічною ефективністю, передбачаючи автоматичне масштабування, управління та захист інфраструктури. Автоматична масштабованість під час зростання запитів забезпечує легке адаптування до змінного навантаження [1].

Як приклад реалізації монолітної архітектури було спроектовано та реалізовано інтернет-магазин історичних артефактів на ім'я «Історичний

Арсенал». Спочатку було обрано кольорову палітру, після чого в сервісі розробки інтерфейсів та прототипування Figma був створений вайфрейм майбутнього сайту та загальний дизайн [2]. За допомогою професійного режиму редактора «Dev Mode» в Figma всі дизайнерські деталі та елементи, створені в програмі, було перетворено на HTML та CSS код (Рис.1.).

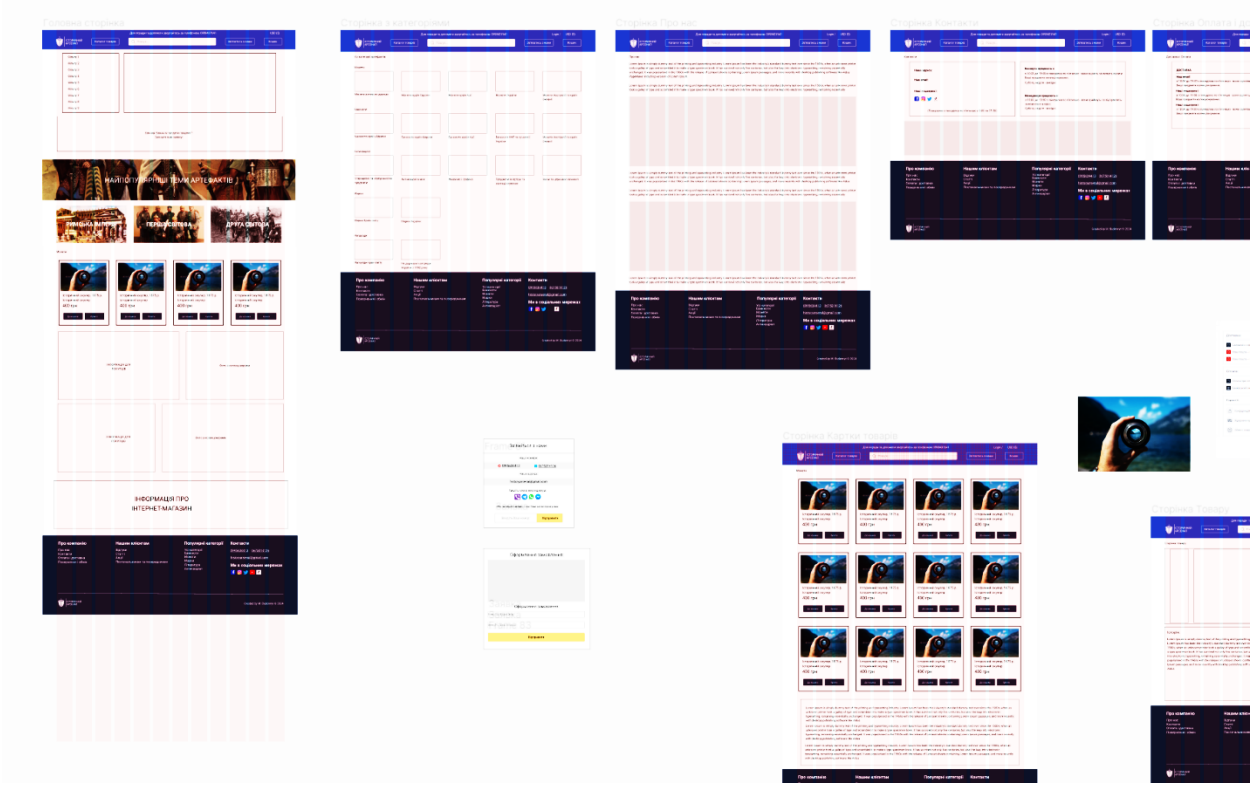

Рисунок 1 – Вайфрейм сайту «Історичний Арсенал»

Як платформу для розробки веб-сайту було обрано Visual Studio Code [4], що є доступною для Windows, macOS і Linux. Для збірки, модифікації та подальшої публікації проекту було обрано GitLab [3] - сервіс, що надає можливість хмарного збереження проекту та його розгортання на власному сервері чи хмарній платформі (Рис.2).

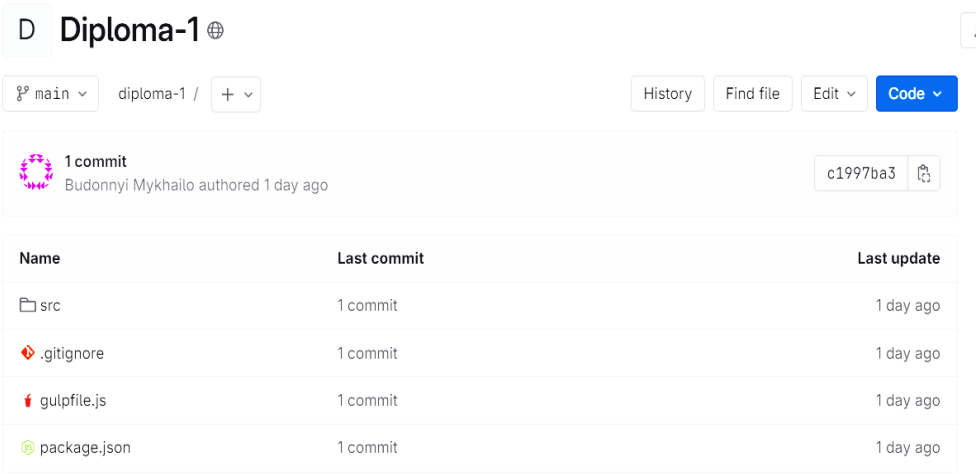

Рисунок 2 – Репозиторій проекту

Також був застосований компілятор проекту Gulp [5] - таск-менеджер для автоматичного виконання завдань, написаний на мові програмування JavaScript (Рис.3.)

```
import gulp from "gulp";
const { src, dest, watch, series, parallel } = gulp;

import imagemin from "gulp-imagemin";
import autoprefixer from "gulp-autoprefixer";
import csso from "gulp-csso";
import clean from "gulp-clean";
import * as dartSass from "sass";
import gulpSass from "gulp-sass";
const sass = gulpSass(dartSass);

import bsc from "browser-sync";
const browserSync = bsc.create();

const htmlTaskHandler = () => {
  return src("./src/*.html").pipe(dest("./dist"));
};

const cssTaskHandler = () => {
  return src("./src/scss/main.scss")
    .pipe(sass().on("error", sass.logError))
    .pipe(autoprefixer())
    .pipe(csso())
    .pipe(dest("./dist/css"))
    .pipe(browserSync.stream());
};
```

Рисунок 3 – Частина коду файлу gulpfile.js

Для оптимізації та зменшення загального розміру проекту було використано препроцесори (в даному випадку SCSS), що дають можливість оголошувати змінні для зберігання значень, таких як кольори, шрифти або