Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Спеціальність 125 «Кібербезпека та захист інформації»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

В.о. зав. кафедрою БІСТ

Марина ЄСІНА

_____

« » 2024 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Аналіз технологій проектування веб додатків з імплементацією методів захисту інформації»

оцінка « »

Голова ЕК

Олександр ЛЕМЕШКО _____

Керівник к. т. н. доцент каф._____

Мелкозьорова О. М.

Рецензент д.т.н. професор

Толстолузька О. Г.

Виконавець : студент групи КБ-61

Левченко Д.О.

Харків – 2024

РЕФЕРАТ

Пояснювальна записка містить 61 сторінок, 46 рисунки, 3 таблиці, 17 джерела.

Метою дипломної роботи є дослідження сучасних технологій створення веб додатків. Побудова веб додатку для торгівлі продукцією. Аналіз сучасних загроз для веб-додатку та методи захисту від них.

Об’єкт дослідження: Предметом дослідження є технології та методи забезпечення інформаційної безпеки, що застосовуються в процесі проектування та реалізації веб-додатків.

Предмет дослідження: технології та методи забезпечення інформаційної безпеки, що застосовуються в процесі проектування та реалізації веб-додатків.

Основні методи дослідження включають аналіз спеціалізованої інформації, комп'ютерне моделювання, розробку програмного прототипу тестового застосунку та узагальнення результатів його тестування.

У роботі ознайомився з основними технологіями проектування веб-додатків, зокрема з архітектурними патернами та фреймворками, які використовуються для створення безпечних веб-систем. Вивчав сучасні загрози інформаційній безпеці веб-додатків, такі як SQL-ін'єкції, міжсайтовий скриптинг (XSS), CSRF-атаки, а також методи їх запобігання. Ознайомився з принципами побудови захищених механізмів автентифікації, авторизації та шифрування даних. Поглиблено досліджував методи інтеграції безпеки на всіх етапах розробки веб-додатків, від проектування до тестування на вразливості. Вивчав популярні фреймворки (Django, Flask, Node.js) і їх можливості для забезпечення захисту інформації. Також набув практичних навичок у тестуванні веб-додатків на наявність вразливостей та виявленні можливих шляхів їх усунення.

Ключові слова: XSS, CSRF, REACT, ІНФОРМАЦІЙНА БЕЗПЕКА, ЗАГРОЗИ БЕЗПЕКИ, SQL-ІН’ЄКЦІЇ, ВЕБ-ДОДАТОК.

ABSTRACT

The explanatory note contains 61 pages, 46 figures, 3 tables, 17 sources.

The purpose of the thesis is to study modern technologies for creating web applications. Building a web application for trading products. Analysis of modern threats to the web application and methods of protection against them.

Object of study: The subject of research is technologies and methods of information security used in the process of designing and implementing web applications.

Subject of research: technologies and methods of ensuring information security used in the process of designing and implementing web applications.

The main research methods include analysis of specialised information, computer-based modelling, development of a software prototype of a test application and generalisation of its testing results.

In this work, he got acquainted with the basic technologies of web application design, including architectural patterns and frameworks used to create secure web systems. He studied modern threats to the information security of web applications, such as SQL injections, cross-site scripting (XSS), CSRF attacks, and methods of preventing them. I learnt the principles of building secure authentication, authorisation, and data encryption mechanisms. He studied in depth the methods of integrating security at all stages of web application development, from design to vulnerability testing. He studied popular frameworks (Django, Flask, Node.js) and their capabilities for information security. He also gained practical skills in testing web applications for vulnerabilities and identifying possible ways to fix them.

Keywords: XSS, CSRF, REACT, INFORMATION SECURITY, SECURITY THREATS, SQL INJECTIONS, WEB APPLICATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА СИМВОЛІВ

HTML	–	HyperText Markup Language
ПЗ	–	Програмне забезпечення
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
HTTP	–	HyperText Transfer Protocol
API	–	Application programming interface
CSS	–	Cascading Style Sheets
XML	–	eXtensible Markup Language
SPA	–	Single Page Application
MVC	–	Model View Controller
DOM	–	Document Object Model
AJAX	–	Asynchronous JavaScript And XML
CSP	–	Content Security Policy
КСЗІ	–	Комплекс засобів захисту
НД ТЗІ	–	Нормативний документ системи технічного захисту інформації
АС	–	Автоматизована система
КЗЗ	–	Комплекс засобів захисту
CSRF	–	Cross-site request forgery
TSL	–	Transport Layer Security
SSL	–	Secure Sockets Layer
WAF	–	Web Application Firewall

ВСТУП

За останні п'ять років Всесвітня павутина зазнала значних змін, і очевидно, що вона також значно еволюціонувала з моменту свого створення.

За 30 років розвитку Інтернету відбулися значні зміни в технологіях і підходах, що використовуються для його побудови. Поява мови сценаріїв JavaScript уможливила створення функціональних веб-додатків. За допомогою JavaScript були розроблені різні методики створення односторінкових сайтів (Single Page Arrangement, SPA). Наразі розробники мають необхідні інструменти, щоб змінити поведінку традиційного веб-сайту і зробити так, щоб він виглядав так, ніби є нативним додатком операційної системи. Серед них найбільш популярною технологією є розробка

Веб-додатки на основі фреймворку React.js та платформи Node.js - це односторінкові додатки (SPA). SPA є повноцінними додатками у вікні браузера і відрізняються від звичайних веб-сторінок більшою функціональністю та швидкістю роботи. SPA не вимагають перезавантаження сторінки, а сама концепція SPA не є новою, вона почала зароджуватися одразу після появи JavaScript та AJAX.

Незважаючи на номенклатуру, основний принцип парадигми односторінкових додатків полягає не в обмеженні кількості сторінок на сайті, а в емуляції зовнішнього вигляду нативного додатка шляхом обходу перезавантажень і перенесення логіки рендерингу з сервера на клієнта, тим самим зменшуючи залежність сайту від сервера і інтернет-з'єднання.

Оновлення проводиться шляхом часткового завантаження контенту на існуючий

Структура процесу завантаження така, що не вимагає значних витрат часу і пам'яті.

Прискорення часу обробки досягається за рахунок заміни HTML-розмітки об'єктами JavaScript, які потім обробляються браузером. Така заміна сприяє підвищенню швидкості роботи веб-додатків.

Останнє покоління SPA розробляється з використанням фреймворків і бібліотек JavaScript.

1 ОГЛЯД МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ ВЕБ-РЕСУРСІВ

1.1 Політика безпеки web-застосувань та серверів

На сьогодні вразливості веб-застосунків залишаються однією з найпоширеніших проблем у сфері інформаційної безпеки. Незважаючи на численні описи цих вразливостей у науково-популярній та спеціалізованій літературі, лише зрідка можна зустріти детальний аналіз превентивних захисних механізмів, спрямованих на зниження ризиків їх використання. Недооцінка серйозності загроз інформаційній безпеці через вразливості веб-застосунків, доступних через мережу Інтернет, є, ймовірно, одним із ключових факторів низького рівня їх захищеності. Статистика свідчить, що у 2013 році критичні вразливості були виявлені на 63% веб-сайтів, що дозволяло здійснювати успішні атаки на веб-застосунки або сервери. Найбільший відсоток застосунків із високо ризиковими вразливостями було зафіксовано у сфері засобів масової інформації: у 80% випадків рівень вразливості оцінювався як критичний. [1].

В останні десятиліття провідні виробники комп'ютерної та програмної інфраструктури, включаючи IBM, Microsoft і Sun Microsystems, інвестували значні ресурси в розширення своїх технологічних можливостей, щоб полегшити розробку, розгортання, обслуговування і забезпечення безпеки веб-серверів. Питання безпеки веб-серверів детально розглядаються в публікаціях [2-4]. Аналіз наукової літератури показав, що, незважаючи на важливість захисту Web-серверів, в нашій країні цьому напрямку приділяється недостатньо уваги. Крім того, наразі відсутня чітко визначена система заходів щодо протидії зловживанням в інформаційній сфері.

Одними з найбільш поширених і небезпечних веб-вразливостей, за даними OWASP (Open Web Application Security Project), є такі: (див. рис. 1.1):

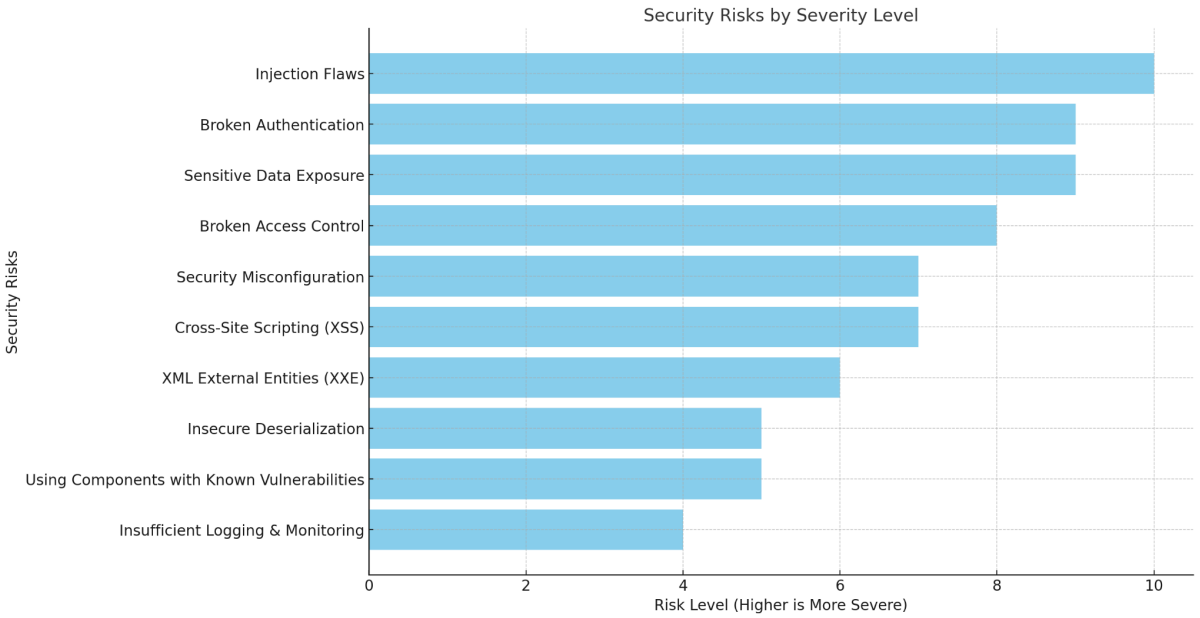

Рисунок 1.1 - Ризики для безпеки за версію OWASP (за даними [2])

Щорічно Acunetix представляє аналіз найпоширеніших вразливостей веб-безпеки та вразливостей мережевого периметру. Щорічний звіт про вразливості веб-додатків (тепер він є частиною Індикатора Invicti AppSec) ґрунтується на емпіричних даних, отриманих з реальних онлайн-сценаріїв, які спостерігаються і реєструються платформою Acunetix Online. Команда експертів випадковим чином відбирає веб-сайти та веб-додатки, які були просканові сканером Acunetix, анонімізує їх та проводить статистичний аналіз. У наступному розділі представлені висновки щодо кібербезпеки, які були зроблені.

На жаль, звіт за 2021 рік представляє досить песимістичні прогнози. Висхідна траєкторія, що спостерігалася в останні роки, вийшла на плато. Поширеність вразливостей високого та середнього рівня зросла порівняно з попереднім роком, з'явилися потенційні ризики безпеки, які можуть призвести до компрометації конфіденційної інформації.

Таке погіршення ситуації спричинене глобальною пандемією, викликаною важким гострим респіраторним синдромом, спричиненим коронавірусом 2 (SARS-CoV-2), широко відомим як новий коронавірус. Пандемія змусила більшість компаній перейти на віддалену роботу, тому багато керівників служб безпеки вирішили зосередитися на безпеці кінцевих точок, безпеці операційних

систем, а також на боротьбі з фішингом, шкідливими веб-сайтами та шкідливим програмним забезпеченням. Як наслідок, не вистачало ресурсів для покращення безпеки в Інтернеті. Замість того, щоб інвестувати в ретельні процеси, компанії обирали швидкі та недосконалі рішення, часто засновані на неправильно налаштованих брандмауерах веб-додатків (WAF). (див. рис. 1.2).

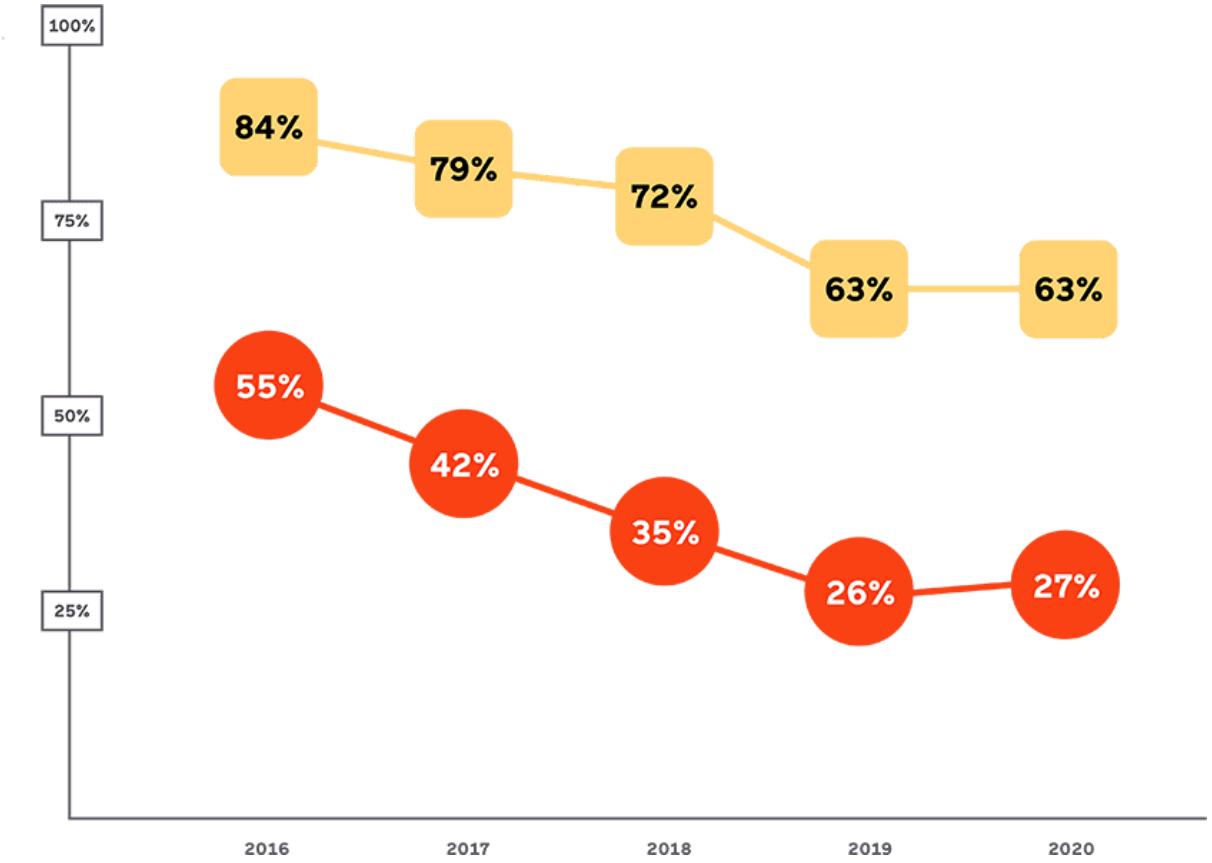

Рисунок 1.2 - Тенденція вразливостей за версією Acunetix (за даними [4])

1.2 Аналіз актуальних атак на веб-ресурси

Багато систем дозволяють використовувати слабкі паролі або ключі шифрування, і користувачі часто вибирають легко вгадуванні або такі, що містяться в словниках парольні фрази.

1) Password selection – (підбір паролів) автоматизований процес спроб і помилок, основною метою якого є з'ясування імені користувача, пароля, номера кредитної картки, ключа шифрування тощо.

2) cross-site scripting (впровадження сценаріїв, XSS) – уразливість, яка дозволяє зловмиснику перехопити конфіденційну інформацію та виконати шкідливий код в контексті веб-браузера.

3) SQL injection (упровадження SQL) – механізм атаки на веб-додатки, які використовують введені користувачем дані в SQL-запитах до бази даних без попередньої обробки для видалення потенційно небезпечних символів і зарезервованих слів. Підробка міжсайтових запитів - ще один спосіб використання уразливостей міжсайтового скриптингу. Це передбачає підробку HTTP-запитів, що дозволяє зловмиснику виконувати запити від імені легітимного користувача.

4) cross-site request forgery – один із способів використання вразливостей cross-site scripting, який полягає в підробці HTTP-запитів; з його допомогою той, хто атакує, дістає можливості виконувати запити від імені законного користувача;

5) command injection (впровадження команд) – н'єкція команд відбувається, коли дані, що передаються на сервер в HTTP-запиті, використовуються для генерації команд операційної системи без перевірки на наявність спеціальних символів. Очевидно, що найефективнішою формою захисту є відсутність вразливостей. Отже, навіть на початкових етапах створення політики безпеки веб-сервера або розробки веб-додатку необхідно враховувати потенційні дії зловмисника з метою компрометації системи і заздалегідь впроваджувати відповідні заходи. Зрештою, саме розробник несе відповідальність за визначення того, які дані повинні бути отримані і як вони повинні бути оброблені, а також які дані ніколи не повинні передаватися на сервер. Отже, доцільно створити модель зловмисника і модель загроз перед розробкою веб-додатку. Тим не менш, незалежно від стадії розробки, важливо захистити додаток і веб-сервер, на якому він розміщений.

Захист веб-сервера - це комплекс організаційних і технічних заходів. Перш ніж впроваджувати ефективну стратегію захисту веб-сервера, необхідно виявити вразливості, притаманні веб-додаткам і політиці безпеки самого веб-сервера. Аудит служить для виявлення помилок і оцінки безпеки веб-додатків, одночасно надаючи набір рекомендацій щодо посилення безпеки веб-сервера.

Одним з методів усунення вразливостей і одночасного запобігання їх виникненню є встановлення брандмауера веб-додатків (WAF). WAF - це програмний або апаратний брандмауер, призначений для використання з веб-додатками. Брандмауер веб-додатків (WAF) - це програмний або апаратний брандмауер для веб-додатків, який встановлюється перед веб-додатком і надає ширші можливості, ніж звичайні програмні брандмауери для системи. WAF може контролювати всі об'єкти, до яких мають доступ користувачі, такі як URL-адреси, дані, що вводяться, а також параметри GET і POST-запитів. Крім того, брандмауер запобігає запуску користувачами об'єктів, які не належать веб-ресурсу [6].

Тим не менш, як свідчать емпіричні дані, WAF часто виявляються неадекватними у виконанні покладених на них завдань. Це можна пояснити двома факторами: по-перше, внутрішніми обмеженнями технології, які не дозволяють їй забезпечити комплексний захист від усіх потенційних вразливостей, в тому числі пов'язаних з реалізацією конкретного брандмауера. Наприклад, було опубліковано кілька багів в mod_security, які дозволяють обійти захист. Крім того, були виявлені варіанти рядків, які дозволяють реалізувати SQL-ін'єкцію на веб-додатки з встановленими phpid. Крім того, уразливість в SecureSphere дозволяє віддаленому користувачеві провести XSS-атаку. Отже, встановлення брандмауерів не забезпечує комплексного вирішення проблеми [7]. Найбільш поширеними методами захисту веб-серверів є видалення зайвого програмного забезпечення, виявлення спроб обійти наявні заходи безпеки, виправлення недоліків у встановленому програмному забезпеченні, пом'якшення впливу атак на мережу та захист решти мережі в разі компрометації веб-сервера.

1.3 Вимоги до захисту інформації WEB – сторінки від несанкціонованого доступу

Документ НД ТЗІ 2.5-010-03 містить вичерпний перелік основних та більш складних вимог, що стосуються захисту інформації. Документ розроблено та впроваджено Головним управлінням технічного захисту інформації Департаменту спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України.

Затверджено Департаментом спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України 2 квітня 2003 року (наказ № 33), зі змінами, внесеними Адміністрацією Державної служби спеціального зв'язку та захисту інформації України 28 грудня 2012 року (наказ № 806).

Загальні вимоги:

1) Установа, під час створення WEB-сторінки та визначення операторів, вузли яких будуть використовуватися для підключення до мережі Інтернет, повинна керуватися законами України, іншими нормативно-правовими актами, що встановлюють вимоги з технічного захисту інформації.

2) WEB-сторінка установи може бути розміщена на власному сервері або на сервері, що є власністю оператора. Власник сервера зобов’язаний гарантувати власнику інформації рівень захисту у відповідності до вимог цього НД ТЗІ.

3) Функціонування WEB-сторінки забезпечується АС, за допомогою якої здійснюється актуалізація розміщених на WEB-сторінці інформаційних ресурсів та керування доступом до них. Для забезпечення захисту інформації WEB-сторінки в цій АС створюється КСЗІ, що є сукупністю організаційних і інженерно-технічних заходів, а також програмно-апаратних засобів, які забезпечують захист інформації.

4) Створення КСЗІ здійснюється відповідно до технічного завдання, розробленого згідно з НД ТЗІ 3.7-001.

5) КСЗІ підлягає державній експертизі у порядку, передбаченому Положенням про державну експертизу в сфері технічного захисту інформації.

6) Захист інформації на всіх етапах створення та експлуатації WEB сторінки здійснюється відповідно до розробленого установою плану захисту інформації, зміст якого визначено НД ТЗІ 1.4-001. План захисту затверджується керівником установи, а у випадку використання сервера оператора – погоджується з власником сервера.

7) Перелік інформації, призначеної для публічного розміщення на WEB сторінці, визначається з урахуванням вимог діючого законодавства та затверджується керівником установи, що є власником WEB-сторінки.

8) Організація робіт із захисту інформації та забезпечення контролю за станом її захищеності на WEB-сторінці в установі здійснюється відповідальним підрозділом або відповідальною особою (далі - службою захисту інформації, СЗІ).

9) У випадку користування послугами оператора щодо розміщення, експлуатації та адміністрування WEB-сторінки власник інформації укладає з оператором договір (угоду), яким визначаються права і обов’язки сторін, умови підключення, розміщення інформації та забезпечення доступу до неї, інші питання, що вимагають врегулювання між власником інформації WEB сторінки та оператором, виходячи з вимог законодавства у сфері захисту інформації та цього НД ТЗІ. Окремі питання із захисту інформації можуть оформлятися у вигляді додатків, які є невід’ємною частиною договору.

1.4 Висновки до першого розділу

У першому розділі було розглянуто основні елементи безпеки веб-ресурсів, зокрема політики безпеки веб-додатків і серверів, сучасні загрози та атаки, а також вимоги до інформаційної безпеки. З проведеного аналізу можна зробити наступні висновки:

Політики безпеки є основою захисту веб-додатків і серверів. Розробка та впровадження таких політик може слугувати зниженню ризику атак та забезпеченню безпеки даних.

Найбільш значущими сучасними загрозами для веб-ресурсів є ті, що використовують SQL-ін'єкції, XSS (міжсайтовий скриптинг), CSRF (підробка міжсайтових запитів) та інші методи. Їх необхідно враховувати при розробці системи безпеки.

Вимоги до інформаційної безпеки включають аутентифікацію, авторизацію, шифрування даних, а також впровадження захищених протоколів зв'язку та дотримання принципів мінімізації доступу. Використання таких підходів, як Content Security Policy (CSP), шифрування з'єднань SSL/TLS і регулярне оновлення програмного забезпечення, є обов'язковими елементами комплексного захисту веб-ресурсів.

2 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-ДОДАТКУ З ІМПЛЕМЕНТАЦІЄЮ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

2.1 HTML та CSS

2.1.1 HTML

Мова розмітки гіпертексту (HTML) є фундаментальною конструкцією Інтернету, що визначає зміст і структуру веб-контенту. Хоча HTML слугує основною мовою для визначення фундаментальної структури веб-сторінок, інші технології, такі як каскадні таблиці стилів (CSS) та JavaScript, широко використовуються для опису візуального представлення та функціональної поведінки веб-сторінок відповідно.

Термін «гіпертекст» використовується для опису посилань, які з'єднують веб-сторінки одна з одною, як в межах одного веб-сайту, так і між різними веб-сайтами. Посилання є фундаментальним аспектом Інтернету. Завантажуючи контент в Інтернет і пов'язуючи його зі сторінками, створеними іншими людьми, користувачі стають активними учасниками всесвітньої павутини.

HTML використовує "розмітку" для коментування тексту, зображень та іншого вмісту для відображення у веб-браузері. Розмітка HTML включає спеціальні "елементи", такі як <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <search>, <output>, <progress>, <video>, <ul>, <ol>, <li> та багато інших.

Елемент HTML відокремлюється від іншого тексту в документі за допомогою "тегів", які складаються з імені елемента, оточеного символами < і >. Ім'я елемента всередині тегу не залежить від регістру. Тобто його можна писати великими, малими літерами або їх комбінацією. Наприклад, тег <title> може бути записаний як <Title>, <TITLE> або будь-яким іншим чином. Однак, за домовленістю і рекомендованою практикою, теги слід писати малими літерами.

HTML - це мова розмітки, яка визначає структуру вашого контенту. HTML складається з низки елементів, які ви використовуєте, щоб вкласти або обернути різні частини контенту, щоб він виглядав певним чином або діяв певним чином. За допомогою тегів-обгорток можна зробити слово або зображення гіперпосиланням на інше місце, виділити слова курсивом, зробити шрифт більшим або меншим і так далі (див. рис. 2.1).

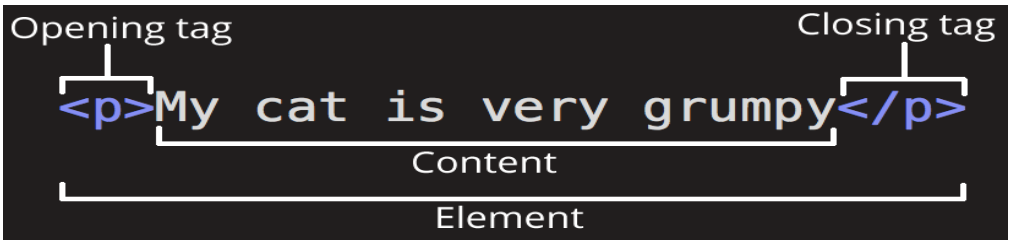

Рисунок 2.1 - Анатомія елемента HTML

Основні частини нашого елемента наступні:

Відкриваючий тег: Складається з назви елемента (у цьому випадку p), взятої у відкриваючу та закриваючу кутові дужки. Тут вказується, де елемент починається або починає діяти - у цьому випадку, де починається абзац.

Закриваючий тег: Це той самий тег, що й відкритий тег, за винятком того, що він містить пряму похилу риску перед назвою елемента. Він вказує, де закінчується елемент - у цьому випадку, де закінчується абзац. Відсутність закриваючого тегу є однією зі стандартних помилок початківців і може призвести до дивних результатів.

Вміст: Це вміст елемента, який в даному випадку є просто текстом.

Елемент: Відкриваючий тег, закриваючий тег і вміст разом складають елемент. Елементи також можуть мати атрибути, які виглядають наступним чином (див. рис. 2.2.):

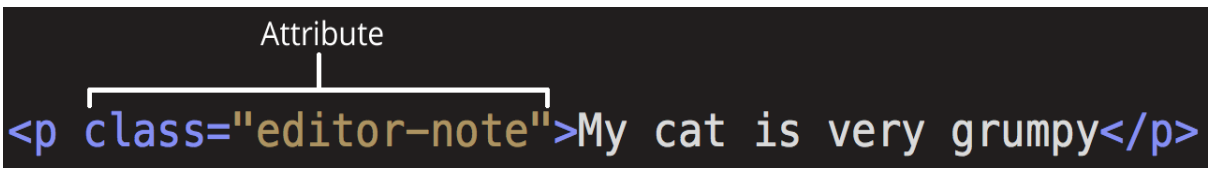

Рисунок 2.2 - Приклад вигляду атрибуту

Атрибути містять додаткову інформацію про елемент, яку ви не хочете показувати в реальному вмісті. Тут class - це назва атрибута, а editor-note - значення атрибута. Атрибут class дозволяє надати елементу неунікальний ідентифікатор, який можна використовувати для його (і будь-яких інших елементів з тим самим значенням класу) таргетування за допомогою інформації про стиль та інших речей. Деякі атрибути не мають значення, наприклад, required.

Атрибути, які задають значення, завжди мають значення:

- Пробіл між ним і назвою елемента (або попереднім атрибутом, якщо елемент вже має один або декілька атрибутів).
- Ім'я атрибута, за яким слідує знак рівності.
- Значення атрибуту, взяте у відкриваючі та закриваючі лапки

2.1.2 CSS

Каскадні таблиці стилів (CSS) - це мова таблиць стилів, яка використовується для опису представлення документа, написаного на HTML або XML (включаючи діалекти XML, такі як SVG, MathML або XHTML). CSS описує, як елементи повинні відображатися на екрані, на папері, в мовленні або на інших носіях.

CSS є однією з основних мов відкритого Інтернету і стандартизована у всіх веб-браузерах відповідно до специфікацій W3C. Раніше розробка різних частин специфікації CSS відбувалася синхронно, що дозволяло версіонувати найновіші рекомендації. Можливо, ви чули про CSS1, CSS2.1 або навіть CSS3. Ніколи не буде CSS3 або CSS4; скоріше, все тепер просто "CSS" з окремими модулями CSS, що мають номери версій.

Після CSS 2.1 обсяг специфікації значно збільшився, а прогрес у різних модулях CSS почав настільки відрізнятися, що стало більш ефективним розробляти і випускати рекомендації окремо для кожного модуля. Замість версій специфікації CSS, W3C тепер періодично робить знімок останнього стабільного стану специфікації CSS і прогресу окремих модулів. Модулі CSS тепер мають номери версій або рівні, наприклад, CSS Color Module Level 5.

Як і HTML, CSS не є мовою програмування. Це також не мова розмітки. CSS - це мова таблиць стилів. CSS - це те, що ви використовуєте для вибіркового стилю елементів HTML (див. рис. 2.3).

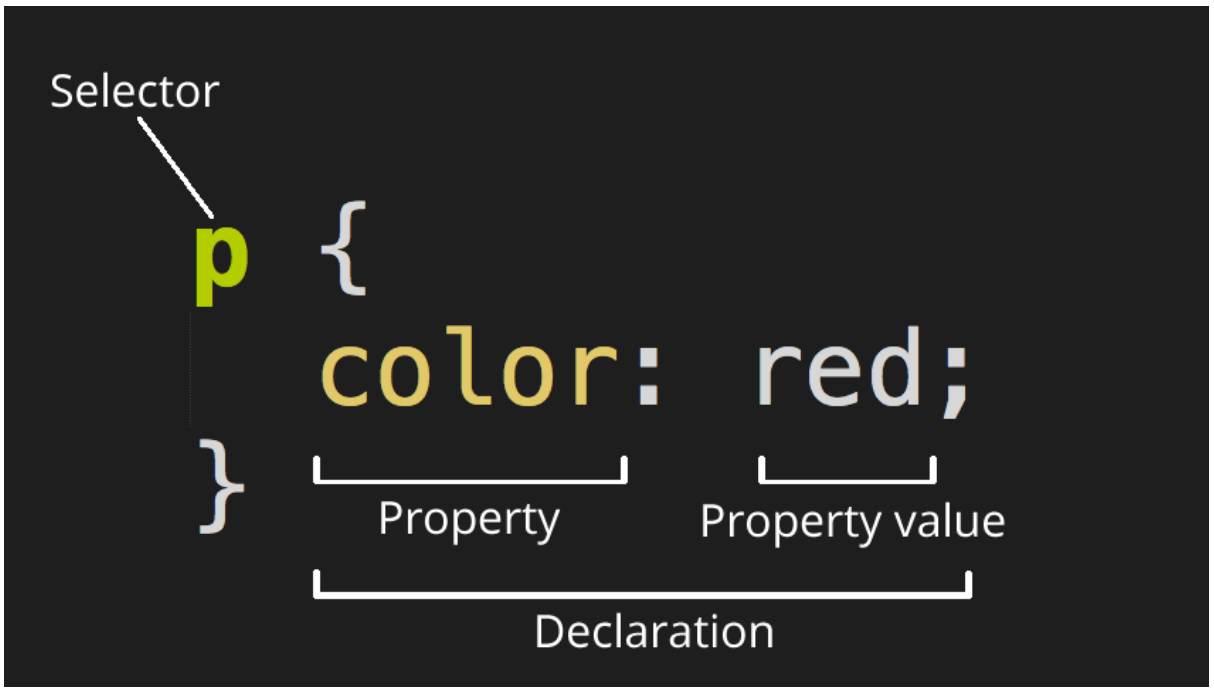

Рисунок 2.3 - Анатомія набору правил CSS

Вся структура називається набором правил. (Термін "набір правил" часто називають просто "правило".)

Селектор - Це ім'я HTML-елемента на початку набору правил. Вона визначає елемент(и), який(і) буде(уть) стилізовано (у цьому прикладі - елементи <p>). Щоб змінити стиль іншого елемента, змініть селектор.

Декларація - Це єдине правило, як color: red;. Воно визначає, яку з властивостей елемента ви хочете стилізувати.

Це способи, за допомогою яких ви можете змінити стиль елемента HTML. (У цьому прикладі колір є властивістю елемента <p>.)

Праворуч від властивості - після двокрапки - знаходиться значення властивості. Вибирається одне з багатьох можливих значень для даної властивості. (Наприклад, існує багато значень кольору, окрім червоного).

Окрім селектора, кожен набір правил має бути загорнутий у фігурні дужки. ({})

У кожній декларації ви повинні використовувати двокрапку (:), щоб відокремити властивість від її значення або значень.

Усередині кожного набору правил ви повинні використовувати крапку з комою (;), щоб відокремити одну декларацію від наступної.

Існує багато різних типів селекторів. У наведених вище прикладах використовуються селектори елементів, які вибирають всі елементи певного типу. Але ми також можемо робити більш специфічні вибірки. Ось деякі з найпоширеніших типів селекторів (див. таблицю 2.1):

Таблиця 2.1 – Типи селекторів

Назва селектора	Що він обирає	Приклад
Селектор елементів (іноді його називають тегом або селектором типу)	Всі HTML-елементи вказаного типу.	p selects <p>
Селектор ідентифікатора	Елемент на сторінці з вказаним ідентифікатором. На даній HTML-сторінці кожне значення ідентифікатора має бути унікальним.	#моя-ідентифікація вибирає <p id="my-id"> або <a id="my-id">
Вибір класу	Елемент(и) на сторінці з вказаним класом. На сторінці може бути декілька екземплярів одного класу.	.my-class вибирає <p class="my-class"> та <a class="my-class">
Селектор атрибутів	Елемент(и) на сторінці з вказаним атрибутом.	img[src] вибирає <img src="my-image.png">, але не <img>

Продовження таблиці 2.1

Псевдоселектор класів	Вказаний(і) елемент(и), але тільки коли він перебуває у вказаному стані. (Наприклад, коли курсор наведено на посилання).	a:hover вибирає <a>, але лише тоді, коли вказівник миші наведено на посилання.

2.2 JavaScript

JavaScript (JS) - це легка інтерпретована (або скомпільована просто в часі) мова програмування з першокласними функціями. Хоча вона найбільш відома як мова сценаріїв для веб-сторінок, багато небраузерних середовищ також використовують її, наприклад, Node.js, Apache CouchDB та Adobe Acrobat. JavaScript - це багатопарадигмальна, однопотокова, динамічна мова, що базується на прототипах, підтримує об'єктно-орієнтовані, імперативні та декларативні (наприклад, функціональне програмування) стилі.

Динамічні можливості JavaScript включають конструювання об'єктів під час виконання, списки параметрів змінних, змінні функцій, динамічне створення сценаріїв (за допомогою eval), самоаналіз об'єктів (за допомогою утиліт for...in і Object) і відновлення вихідного коду (функції JavaScript зберігають свій вихідний текст і можуть бути отримані за допомогою toString()).

Стандартами для JavaScript є специфікація мови ECMAScript (ECMA-262) та специфікація API інтернаціоналізації ECMAScript (ECMA-402).

JavaScript можна використовувати як для клієнтських, так і для серверних розробок. JavaScript є як імперативною, так і декларативною мовою. JavaScript містить стандартну бібліотеку об'єктів, таких як масив, дата та математичні функції, а також основний набір мовних елементів, таких як оператори, керуючі структури та оператори.

На стороні клієнта: надає об'єкти для керування браузером та його об'єктною моделлю документа (DOM). Наприклад, клієнтські розширення дозволяють додатку розміщувати елементи на HTML-формі та реагувати на події

користувача, такі як клацання мишею, введення даних у формі та навігація по сторінці. Корисними бібліотеками для клієнтської частини є AngularJS, ReactJS, VueJS та багато інших.

На стороні сервера: Робота на стороні сервера включає в себе такі речі, як зв'язок з базою даних, маніпулювання файлами та генерування відповідей. Завдяки середовищу виконання Node.js, яке дозволяє запускати JavaScript поза браузером, та фреймворкам, таким як Express.JS, JavaScript зараз широко використовується на стороні сервера.

Імперативна мова - у цьому типі мови нас найбільше цікавить, як це має бути зроблено. Вона просто контролює потік обчислень. Процедурний підхід до програмування, об'єктно-орієнтований підхід підпадає під цю категорію, оскільки в асинхронному очікуванні ми думаємо про те, що робити далі після виклику асинхронізації.

Декларативне програмування - у цьому типі мови нас цікавить, як це має бути зроблено, в основному тут потрібні логічні обчислення. Її основна мета - описати бажаний результат без прямої вказівки, як його отримати, як це робить функція зі стрілкою.

Для більш структурованого підходу до вивчення JavaScript наш курс пропонує повний посібник, від основ до розширених можливостей. Незалежно від того, чи ви тільки починаєте, чи хочете вдосконалити свої навички, цей курс допоможе вам оволодіти JavaScript.

JavaScript можна додати до HTML-файлу двома способами:

Внутрішній JS: Ми можемо додати JavaScript безпосередньо до нашого HTML-файлу, написавши код всередині тегу <script>. Тег <script> може бути розміщений всередині тегу <head> або <body> відповідно до вимог.

Зовнішній JS: Ми можемо написати код JavaScript в іншому файлі з розширенням js, а потім додати цей файл у тег <head> HTML-файлу, в який ми хочемо додати цей код.

JavaScript створив Брендан Айх у 1995 році. Він був інженером у компанії Netscape. Спочатку вона мала називатися LiveScript, але була перейменована. На

відміну від більшості мов програмування, мова JavaScript не має поняття введення або виведення. Вона призначена для запуску як мова сценаріїв у хост-середовищі, і саме хост-середовище забезпечує механізми для спілкування із зовнішнім світом. Найпоширенішим хост-середовищем є браузер. Будь ласка, зверніться до розділу Історія JavaScript для більш детальної інформації.

Згідно з нещодавнім опитуванням [18], проведеним Stack Overflow, JavaScript є найпопулярнішою мовою на землі. З розвитком технології браузерів та переміщенням JavaScript на сервер разом з Node.js та іншими фреймворками, JavaScript здатен на набагато більше. Ось декілька речей, які ми можемо робити за допомогою JavaScript:

JavaScript був створений в першу чергу для маніпуляцій з DOM. Раніше веб-сайти були переважно статичними, після створення JS з'явилися динамічні веб-сайти.

Функції в JS - це об'єкти. Вони можуть мати властивості та методи, як і інші об'єкти. Їх можна передавати як аргументи в інші функції.

Може обробляти дату та час. Виконує перевірку форми, хоча форми створено за допомогою HTML. Компілятор не потрібен.

Застосування JavaScript:

- Веб-розробка: Додавання інтерактивності та поведінки до статичних сайтів Для цього у 1995 році було винайдено JavaScript. За допомогою AngularJS цього можна досягти дуже легко.
- Веб-додатки: З розвитком технологій браузери вдосконалилися до такої міри, що для створення надійних веб-додатків знадобилася мова. Коли ми досліджуємо карту в Google Maps, нам потрібно лише клацнути і перетягнути мишу. Весь детальний перегляд знаходиться на відстані одного кліку, і це можливо лише завдяки JavaScript. Він використовує інтерфейси прикладного програмування (API), які надають додаткову потужність коду. Electron і React допомагають у цьому відділі.
- Серверні додатки: За допомогою Node.js JavaScript пройшов шлях від клієнта до сервера, і Node.js є найпотужнішим на серверній стороні.

- Ігри: Не лише на веб-сайтах, JavaScript також допомагає у створенні ігор для дозвілля. Поєднання JavaScript та HTML 5 робить JavaScript популярним і в розробці ігор. Для цього існує бібліотека Ease JS, яка надає рішення для роботи з насиченою графікою.
- Розумні годинники: JavaScript використовується у всіх можливих пристроях і додатках. Він надає бібліотеку Pebble JS, яка використовується в додатках для смарт-годинників. Цей фреймворк працює для додатків, які потребують Інтернету для свого функціонування.
- Мистецтво: Художники та дизайнери можуть створювати все, що завгодно, використовуючи JavaScript для малювання на полотні HTML 5, а щоб зробити звук більш ефективним, також можна використовувати бібліотеку p5.js.
- Машинне навчання: Ця бібліотека JavaScript ml5.js може бути використана у веб-розробці за допомогою машинного навчання.
- Мобільні додатки: JavaScript також можна використовувати для створення додатків для не-веб-контекстів. Можливості та використання JavaScript роблять його потужним інструментом для створення мобільних додатків. Це фреймворк для створення веб- та мобільних додатків за допомогою JavaScript. Використовуючи React Native, ми можемо створювати мобільні додатки для різних операційних систем. Нам не потрібно писати код для різних систем.
- Ризики для безпеки: JavaScript можна використовувати для отримання даних за допомогою AJAX або шляхом маніпулювання тегами, які завантажують дані, такими як <img>, <object>, <script>. Ці атаки називаються міжсайтовими скриптовими атаками. Вони впроваджують JS, який не є частиною сайту, в браузер відвідувача і таким чином отримують дані.
- Продуктивність: JavaScript не забезпечує такого ж рівня продуктивності, як багато традиційних мов, оскільки складна програма, написана на JavaScript, буде працювати порівняно повільно. Але оскільки JavaScript використовується для виконання простих завдань у браузері, то продуктивність не вважається великим обмеженням у його використанні.

- Складність: Щоб опанувати мову сценаріїв, програмісти повинні досконало знати всі концепції програмування, основні об'єкти мови, а також об'єкти на стороні клієнта і сервера, інакше їм буде складно писати складні сценарії з використанням JavaScript.
- Слабкі засоби обробки помилок та перевірки типів: Це динамічно типізована мова, оскільки немає необхідності вказувати тип даних змінної. Тому перевірка на неправильний тип не виконується під час компіляції.

2.3 TypeScript

TypeScript знаходиться у незвичних відносинах з JavaScript. TypeScript пропонує всі можливості JavaScript, а також додатковий рівень над ними: Система типів TypeScript.

Наприклад, JavaScript надає мовні примітиви, такі як рядок і число, але він не перевіряє, чи правильно ви їх присвоїли. TypeScript робить це.

Це означає, що ваш існуючий робочий JavaScript-код також є кодом TypeScript. Основна перевага TypeScript полягає в тому, що він може підсвічувати неочікувану поведінку у вашому коді, зменшуючи ймовірність появи помилок.

TypeScript знає мову JavaScript і в багатьох випадках буде генерувати типи для вас. Наприклад, при створенні змінної та присвоєнні їй певного значення, TypeScript буде використовувати це значення як її тип (див. рис. 2.4).

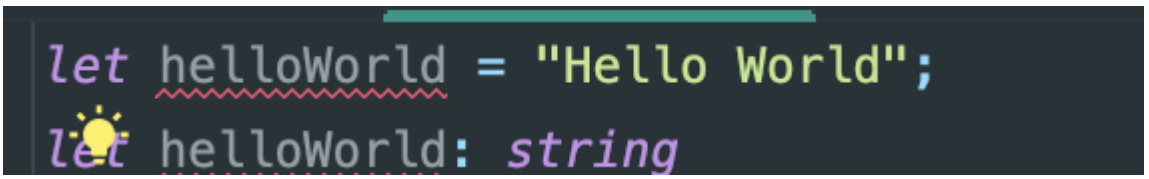

Рисунок 2.4 - Присвоєння значення та типу

Розуміючи, як працює JavaScript, TypeScript може створити систему типів, яка приймає код JavaScript, але має типи. Це дозволяє створити систему типів без необхідності додавати додаткові символи, щоб зробити типи явними.

У JavaScript можна використовувати широкий спектр шаблонів дизайну. Однак деякі шаблони ускладнюють автоматичне визначення типів (наприклад, шаблони, що використовують динамічне програмування). Для таких випадків

TypeScript підтримує розширення мови JavaScript, яке пропонує місця, де ви можете вказати TypeScript, якими мають бути типи.

Наприклад, щоб створити об'єкт з виведеним типом, який включає ім'я: рядок та ідентифікатор: число, можливо написати як наведено на рисунку 2.5

Рисунок 2.5 - Створення об'єкту з виведеним типом

Тепер є можливість явно описати форму цього об'єкта за допомогою оголошення інтерфейсу (див. рис. 2.6):

Рисунок 2.6 - Оголошення інтерфейсу User

Тепер є здатність оголосити, що об'єкт JavaScript відповідає формі вашого нового інтерфейсу, використовуючи синтаксис на кшталт : TypeName після оголошення змінної (див. рис. 2.7):

```
const user: User = {
    name: "Хейс",
    id: 0,
};
```

Рисунок 2.7 - Присвоєння інтерфейсу User до змінної user

Якщо надати об'єкт, який не відповідає наданому інтерфейсу, TypeScript попередить про це (див. рис. 2.8):

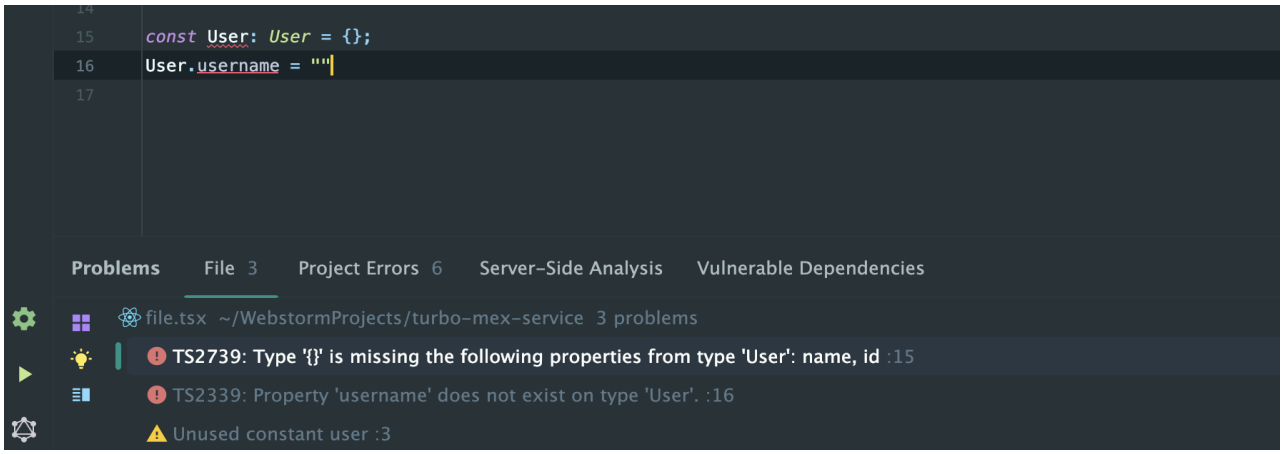

Рисунок 2.8 - Присвоєння змінної User хибного параметру

Об'єктний літерал може вказувати лише відомі властивості, а "ім'я користувача" не існує в типі "Користувач".

Об'єктний літерал може вказувати лише відомі властивості, а "ім'я користувача" не існує в типі "Користувач".

Оскільки JavaScript підтримує класи та об'єктно-орієнтоване програмування, TypeScript також підтримує їх. Можна використовувати оголошення інтерфейсу з класами (див. рис. 2.9):

```
interface User {  Show usages
    name: string;
    id: number;
}
class UserAccount {  Show usages
    name: string;
    id: number;

    constructor(name: string, id: number) {  Show usages
        this.constructor(name: string, id: number)
        this.name = name;
        this.id = id;
    }
}
    const user: User = new UserAccount("Murphy", 1);
```

Рисунок 2.9 - Оголошення інтерфейсу з класами

Користувач може використовувати інтерфейси для коментування параметрів і повернення значень функціям (див. рис. 2.10):

```
function deleteUser(user: User) {  no usages
    // ...
}

function getAdminUser(): User {  no usages
    //...
}
```

Рисунок 2.10 - Коментування параметрів і повернення значень функціям

У JavaScript вже існує невеликий набір примітивних типів: boolean, bigint, null, число, рядок, символ і невизначений, які розробник може використовувати в інтерфейсі. TypeScript розширює цей список ще кількома типами, такими як any (допускає все, що завгодно), unknown (переконайтеся, що той, хто використовує цей тип, декларує, що це за тип), never (неможливо, що цей тип може зустрітися) і

void (функція, яка повертає невизначене значення або не має значення, що повертається).

Можна побачити, що існує два синтаксиси для створення типів: Інтерфейси та Типи. Слід надавати перевагу інтерфейсу та використовувати тип, коли потрібні специфічні можливості.

За допомогою TypeScript розробник може створювати складні типи, комбінуючи прості. Існує два популярних способи: об'єднання та узагальнення.

Union

За допомогою об'єднання можна оголосити, що тип може бути одним з багатьох типів. Наприклад (див. рис. 2.11):

```
type MyBool = true | false;
```

Рисунок 2.11 - Оголошення типу MyBool

MyBool класифікується як boolean. Це властивість системи структурних типів.

Популярним випадком використання типів об'єднання є опис набору рядкових або числових літералів, якими може бути значення (див. рис. 2.12):

```
type WindowStates = "open" | "closed" | "minimized";
type LockStates = "locked" | "unlocked";
type PositiveOddNumbersUnderTen = 1 | 3 | 5 | 7 | 9;
```

Рисунок 2.12 - Оголошення складних типів

Об'єднання також дозволяють працювати з різними типами. Наприклад, у вас може бути функція, яка приймає масив або рядок (див. рис. 2.13):

```
function getLength(obj: string | string[]) {  no usages
    return obj.length;
}
```

Рисунок 2.13 - Оголошення пропса з різними параметрами

Щоб дізнатися тип змінної, використовуйте typeof:

- рядок - typeof s === "string"
- номер - typeof n === "число"
- boolean - typeof b === "boolean"
- невизначений - typeof undefined === "невизначений"
- функція - typeof f === "функція"
- масив - Array.isArray(a)

Наприклад, розробник може зробити так, щоб функція повертала різні значення залежно від того, чи передається їй рядок, чи масив (див. рис. 2.14):

```
function wrapInArray(obj: string | string[]) {  no usages
    if (typeof obj === "string") {
        {
            return [obj];
        }
        return obj;
    }
}
```

Рисунок 2.14 - Функція,яка перевіряє тип obj

Генерики

Типи надають змінні типам. Типовим прикладом є масив. Масив без узагальнень може містити будь-що. Масив з узагальненнями може описувати значення, які містить масив (див. рис. 2.15).

```
type StringArray = Array<string>;
type NumberArray = Array<number>;
type ObjectWithNameArray = Array<{ name: string }>;
```

Рисунок 2.15 - Демонстрація роботи генериків

Можна оголошувати власні типи, які використовують узагальнення (див. рис. 2.16):

```
interface Backpack<Type> {  no usages
    add: (obj: Type) => void;
    get: () => Type;
}
```

Рисунок 2.16 - Оголошення власного типу з узагальненням

// Цей рядок є ярликом для повідомлення TypeScript про те, що існує

// константу з назвою `backpack`, і не турбуватися про те, звідки вона взялася. (див. рис. 2.17)

```
const backpack: Backpack<string>;
```

Рисунок 2.17 - Оголошення змінної backpack

// об'єкт є рядком, тому що ми оголосили його вище як змінну частину Backpack. (див. рис. 2.18)

```
const object = backpack.get();
```

Рисунок 2.18 - Присвоєння object результату функції backpack.get()

// Оскільки змінна backpack є рядком, ви не можете передати число у функцію add. (див. рис. 2.19)

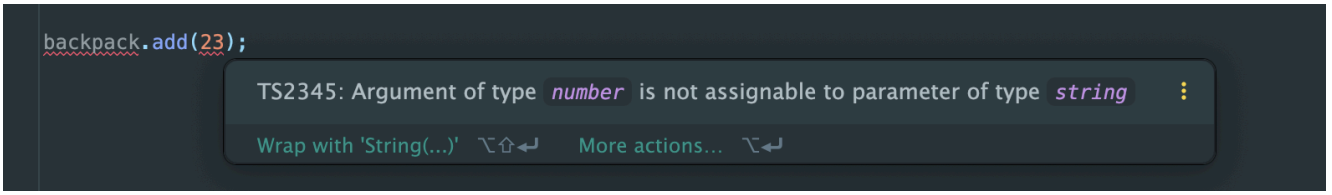

Рисунок 2.19 - Демонстрація помилки при присвоєнні помилкового типу

Один з основних принципів TypeScript полягає в тому, що перевірка типу фокусується на формі, яку мають значення. Іноді це називають "качиною типізацією" або "структурною типізацією".

У системі структурних типів, якщо два об'єкти мають однакову форму, вони вважаються одного типу (див. рис. 2.20).

```
interface Point {  Show usages
    x: number;
    y: number;
}

function logPoint(p: Point) {  Show usages
    console.log(`${p.x}, ${p.y}`);
}

// logs "12, 26"
const point = { x: 12, y: 26 };
logPoint(point);
```

Рисунок 2.20 - Демонстрація структурної типізації

Змінна point ніколи не оголошується як тип Point. Однак, TypeScript порівнює форму точки з формою типу Point при перевірці типу. Вони мають однакову форму, тому код проходить.

Для зіставлення за формою потрібно, щоб співпала лише підмножина полів об'єкта. (див. рис. 2.21)

```
const point3 = { x: 12, y: 26, z: 89 };
logPoint(point3); // виводить "12, 26"

const rect = { x: 33, y: 3, width: 30, height: 80 };
logPoint(rect); // виводить "33, 3"

const color = { hex: "#187ABF" };
logPoint(color);
```

Рисунок 2.21 - Зіставлення за формою

Аргумент типу '{ hex: string; }' не може бути присвоєний параметру типу 'Point'.

Типу '{ hex: string; }' не вистачає наступних властивостей типу 'Point': x, y

Аргумент типу '{ hex: string; }' не може бути присвоєний параметру типу 'Point'.

Типу '{ hex: string; }' не вистачає наступних властивостей типу 'Point': x, y

Немає різниці між тим, як класи та об'єкти відповідають формам (див. рис. 2.22):

```
class VirtualPoint {  Show usages
    x: number;
    y: number;

    constructor(x: number, y: number) {  Show usages
        this.x = x;
        this.y = y;
    }
}

const newVPoint = new VirtualPoint(13, 56);
logPoint(newVPoint); // виводить "13, 56"
```

Рисунок 2.22 - Робота класу VirtualPoint

Якщо об'єкт або клас має всі необхідні властивості, TypeScript повідомить, що вони збігаються, незалежно від деталей реалізації.

2.4 React

React (раніше React.js та ReactJS) - відкрита JavaScript бібліотека для створення користувацьких інтерфейсів. Вона була розроблена для вирішення проблем часткового оновлення вмісту веб-сторінок, які виникають при розробці односторінкових додатків. Розробляється компанією Meta (раніше Facebook) та спільнотою індивідуальних розробників.

Фреймворк React дозволяє розробникам створювати великі веб-додатки, які використовують дані, що можуть змінюватися з часом, без необхідності перезавантаження сторінки. Метою є досягнення швидкості, простоти та масштабованості. React - це бібліотека інтерфейсу користувача, яка обробляє інтерфейс користувача в додатках. Це відповідає представленню в патерні

модель-подання-контролер (MVC) і може використовуватися в поєднанні з іншими бібліотеками JavaScript або у великих MVC-фреймворках, таких як AngularJS. Він також може використовуватися з фреймворками на основі React-надбудов, щоб подбати про інтерфейсні частини створення веб-додатків. Як бібліотека користувацького інтерфейсу, React найчастіше використовується в поєднанні з іншими бібліотеками, такими як Redux.

Бібліотека була розроблена Джорданом Воком, інженером-програмістом у Facebook. Автор працював над проектом під впливом XHP, HTML-фреймворку для PHP. Вперше про реліз було оголошено у 2011 році на Facebook, а згодом у блозі Instagram. Фреймворк також був представлений як проект з відкритим вихідним кодом на конференції для розробників JSConf US, що відбулася в США в травні 2013 року. На конференції React.js Conf, організованій Facebook у березні 2015 року, проект був представлений як програмне забезпечення з відкритим вихідним кодом.

Властивості передаються рендерингу компонента у вигляді властивостей HTML-тегів. Компонент не може змінювати властивості, які були передані йому безпосередньо, але він може змінювати їх за допомогою функцій зворотного виклику. Цей механізм називається «властивості вниз, події вгору».

React використовує віртуальний DOM замість того, щоб покладатися лише на DOM браузера. Це дозволяє бібліотеці визначити, які елементи DOM були змінені, порівнюючи їх зі збереженою версією віртуального DOM, таким чином визначаючи оптимальний підхід для оновлення DOM браузера. (див. рис. 2.23)

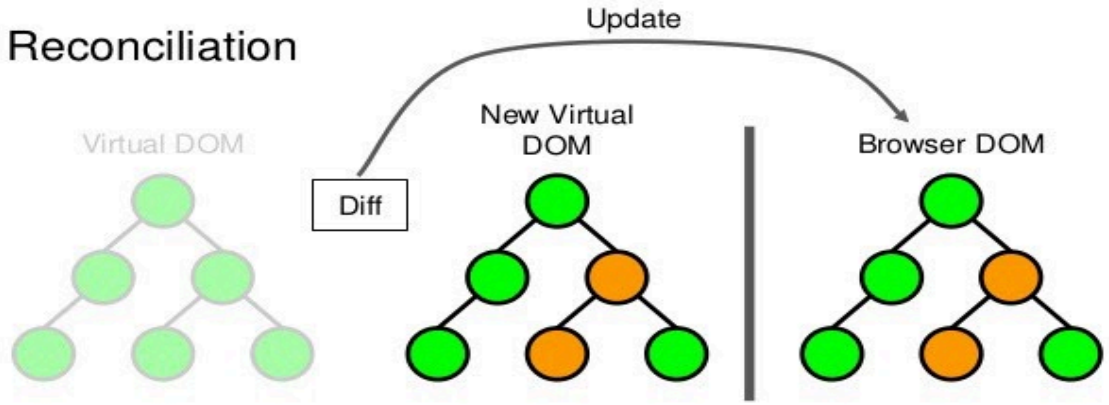

Рисунок 2.23 - Принцип роботи Virtual DOM

Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити. (див. рис. 2.24)

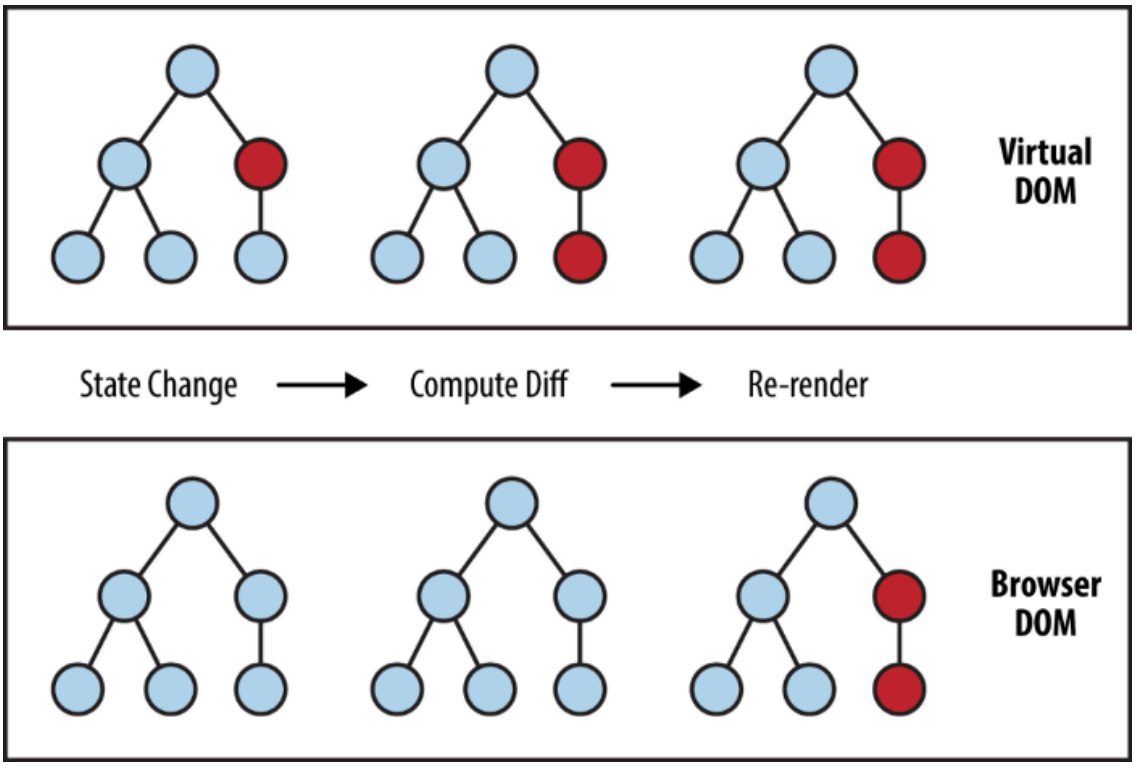

Рисунок 2.24 - Загальна схема заміни вузлів Virtual DOM

Компоненти React зазвичай пишуться на JSX, що є розширенням мови JavaScript. Код, написаний на JSX, компілюється у виклики методів бібліотеки

React. Розробники також можуть писати чистим JavaScript. JSX схожа на іншу мову, створену Facebook для розширення PHP, XHP..

Фреймворк React використовується для багатьох цілей, окрім простого відображення HTML у браузері. Наприклад, Facebook використовує динамічні графіки, які відображаються за допомогою тегів <canvas>, а Netflix та PayPal використовують ізоморфне завантаження для відображення ідентичного HTML на сервері та клієнті.

Методи життєвого циклу - це набір вбудованих функцій у фреймворку ReactJS, які дозволяють розробникам обробляти дані на різних етапах життєвого циклу React-додатку. Наприклад:

shouldComponentUpdate — Це метод життєвого циклу, який дозволяє JavaScript оновити компонент, керуючись логічними змінними.

componentWillMount — Це метод життєвого циклу, який інструктує JavaScript підготувати певні дані перед тим, як компоненти будуть змонтовані у віртуальний DOM.

componentDidMount — Це метод життєвого циклу, схожий на componentWillMount, але виконується після методу render. Його можна використовувати для завантаження JSON-даних, а також для визначення властивостей і станів компонента.

render є найважливішим методом життєвого циклу, необхідним у кожному компоненті. Він відповідає за з'єднання з JSX та відображення власного JSX-коду.

Кілька елементів на одному рівні повинні бути загорнутими в один елемент контейнера, наприклад елемент , або повернутий як масив. (див. рис. 2.25)

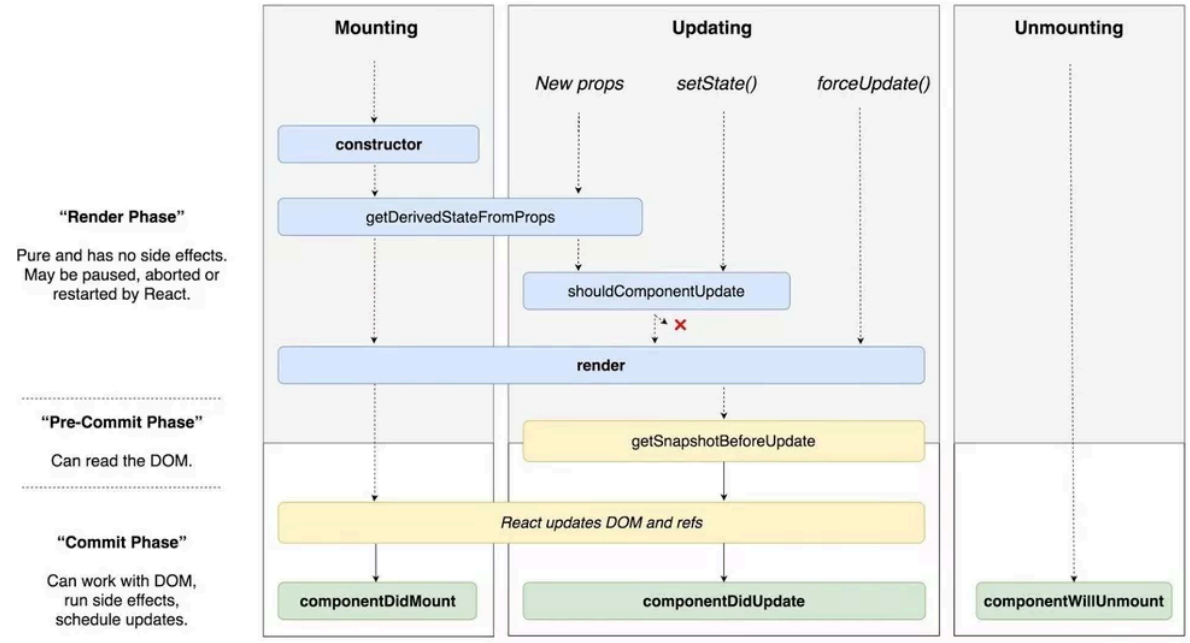

Рисунок 2.25 - Схема життєвого циклу React

JSX надає ряд атрибутів елементів, призначених для відображення тих, що надаються у форматі HTML. Користувацькі атрибути також можуть бути передані компоненту. Всі атрибути будуть отримані компонентом як реквізит.

2.5 Next.js

Next.js - це фреймворк React для створення повностекових веб-додатків. Ви використовуєте React-компоненти для створення користувацьких інтерфейсів, а Next.js - для додаткових функцій та оптимізацій.

Під капотом Next.js також абстрагується і автоматично налаштовує інструменти, необхідні для React, такі як бандлінг, компіляція тощо. Це дозволяє вам зосередитись на створенні вашого додатку замість того, щоб витрачати час на конфігурацію (див. таблицю 2.2).

Таблиця 2.2 – Можливості Next.js

Особливість	Опис
Маршрутизація	Маршрутизатор на основі файлової системи, побудований на основі серверних компонентів, який підтримує макети, вкладену маршрутизацію, стани завантаження, обробку помилок і багато іншого.

Продовження таблиці 2.2

Рендеринг	Рендеринг на стороні клієнта та сервера за допомогою клієнтських та серверних компонентів. Подальша оптимізація за допомогою статичного та динамічного рендерингу на сервері за допомогою Next.js. Потокова передача на Edge та Node.js.
Отримання даних	Спрощена вибірка даних за допомогою асинхронізації/очікування в серверних компонентах, а також розширений API вибірки для запам'ятовування запитів, кешування даних і повторної перевірки.
Стайлінг	Підтримка ваших улюблених методів стилів, включаючи CSS-модулі, Tailwind CSS та CSS-in-JS
Оптимізації	Оптимізація зображень, шрифтів і скриптів для покращення основних веб-показників вашого додатку та зручності користування.
Оптимізації	Оптимізація зображень, шрифтів і скриптів для покращення основних веб-показників вашого додатку та зручності користування.
TypeScript	Покращена підтримка TypeScript, з кращою перевіркою типів і ефективнішою компіляцією, а також користувацький плагін TypeScript і перевірка типів.

2.6 AJAX

Асинхронний JavaScript і XML (Ajax, або AJAX) - це техніка веб-розробки, в якій веб-додаток отримує вміст з сервера за допомогою асинхронних HTTP-запитів і використовує новий вміст для оновлення відповідних частин сторінки, не вимагаючи повного завантаження сторінки. Це може зробити сторінку більш адаптивною, оскільки запитуються лише ті частини, які потрібно оновити. (див. рис. 2.26)

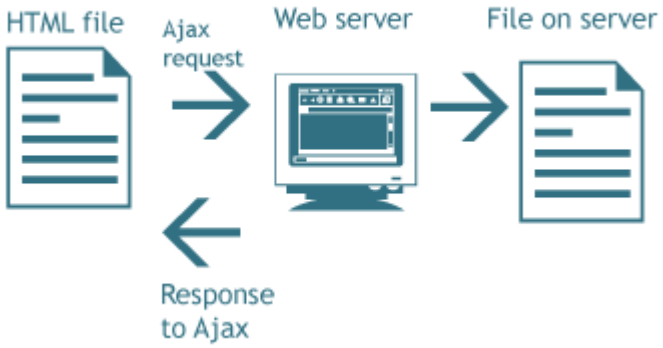

Рисунок 2.26 - Схема принципу роботи AJAX

Ajax можна використовувати для створення односторінкових додатків, в яких весь веб-додаток складається з одного документа, який використовує Ajax для оновлення свого вмісту за потреби.

Спочатку Ajax був реалізований за допомогою інтерфейсу XMLHttpRequest, але API fetch() більше підходить для сучасних веб-додатків: він потужніший, гнучкіший і краще інтегрується з фундаментальними технологіями веб-додатків, такими як сервісні працівники. Сучасні веб-фреймворки також надають абстракції для Ajax.

2.7 Radix UI

Radix UI - це потужний компонент інтерфейсу, який пропонує інтерактивні та адаптивні теми, шрифти, утиліти та настроювані варіанти стилів для швидшої розробки та кращої взаємодії з користувачем

Це бібліотека з відкритим вихідним кодом з підтримкою серверного рендерингу, яка широко використовується розробниками по всьому світу.

Radix UI - це потужна бібліотека інтерфейсу, яка використовується для швидкого і красивого створення тем, макетів, шрифтів, кольорів для вашої веб-сторінки. Деякі з основних можливостей Radix UI згадані нижче:

- За допомогою Radix UI ми можемо задавати кольори, типографіку, тіні, теми курсору та інші варіанти компонентів.
- Radix ui не має стилів за замовчуванням. Це забезпечує гнучкість у структуруванні інтерфейсу відповідно до потреб.
- Забезпечує вбудований доступ
- Radix - це модульна бібліотека, тому вам потрібно імпортувати лише ті компоненти, які ви будете використовувати, що зменшує розмір бібліотеки.
- Інтерактивні та адаптивні компоненти інтерфейсу, такі як діалогові вікна, картки, випадаюче меню, акордеон, спливаючі вікна, вкладки тощо.
- Radix UI також підтримує анімації та переходи, де розробники можуть створювати власні анімації, використовуючи інші бібліотеки та фреймворки.
- Він забезпечує рендеринг на стороні сервера (SSR) для веб-додатків.
- Radix UI пропонує хуки React для керування станом та поведінкою компонентів.

2.8 REST

Representational State Transfer (REST) - це архітектурний стиль для забезпечення стандартів між комп'ютерними системами в Інтернеті, тим самим полегшуючи взаємодію між системами. REST-сумісні системи, які часто називають RESTful-системами, відрізняються своєю бездержавною природою і розділенням функцій клієнта і сервера. (див. рис. 2.27).

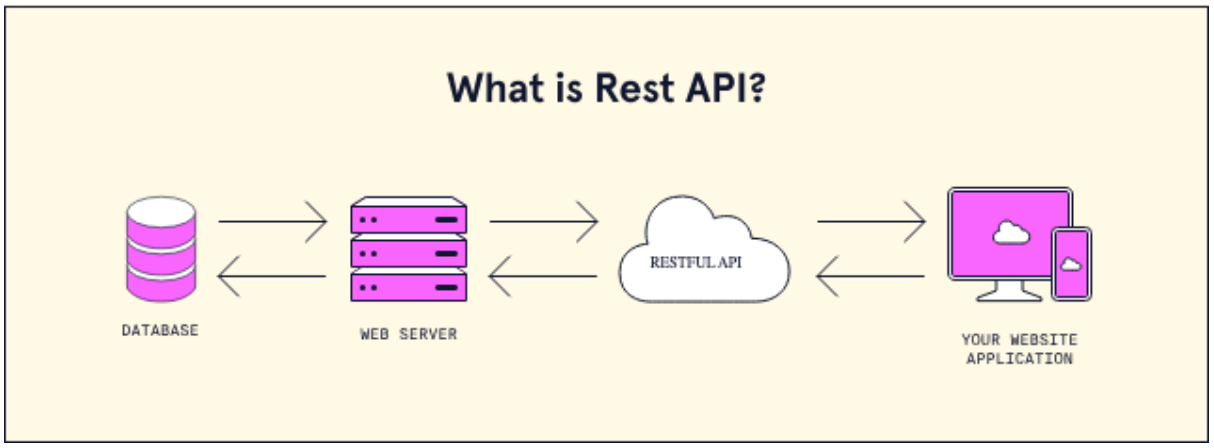

Рисунок 2.27 - Схема принципу роботи REST API

В архітектурному стилі REST клієнтська та серверна частини можуть виконуватися незалежно одна від одної, без необхідності взаємного інформування. Це означає, що код на стороні клієнта може бути змінений в будь-який час без наслідків для сервера, а код на стороні сервера може бути змінений без впливу на клієнта.

За умови, що кожна сторона знає відповідний формат для передачі повідомлень іншій стороні, вони можуть підтримувати модульність і відокремленість. Розмежовуючи проблеми користувацького інтерфейсу та зберігання даних, ми підвищуємо гнучкість користувацького інтерфейсу на різних платформах та покращуємо масштабованість за рахунок оптимізації серверних компонентів. Крім того, розділення дозволяє кожному компоненту розвиватися незалежно.

Інтерфейс REST дозволяє різним клієнтам отримувати доступ до ідентичних кінцевих точок REST, виконувати аналогічні дії та отримувати аналогічні відповіді.

Системи, які дотримуються парадигми REST (Representational State Transfer - передача представницького стану), є незалежними від стану, тобто серверу не потрібно знати про стан клієнта, і навпаки. Отже, і сервер, і клієнт можуть розуміти будь-яке отримане повідомлення без посилання на попередні повідомлення. Така бездержавність досягається завдяки використанню ресурсів

замість команд. Ресурси - це іменники в Інтернеті, що позначають будь-який об'єкт, документ або сутність, які можна зберігати або передавати іншим службам.

RESTful системи взаємодіють за допомогою стандартних операцій з ресурсами, усуваючи таким чином необхідність у реалізації інтерфейсів.

Ці обмеження сприяють надійності, швидкості та масштабованості RESTful-додатків, дозволяючи керувати, оновлювати та повторно використовувати компоненти, не впливаючи на систему в цілому, навіть під час її роботи.

В архітектурі REST клієнти надсилають запити на отримання або зміну ресурсів, а сервери надсилають відповіді на ці запити. У цьому розділі будуть розглянуті стандартні методи надсилання запитів і відповідей.

Архітектурний стиль Representational State Transfer (REST) передбачає, що клієнт ініціює запит до сервера з метою отримання або модифікації даних, що зберігаються на сервері. Запит зазвичай складається з трьох основних компонентів:

1) Дієслово HTTP, яке визначає тип операції, що має бути виконана.
2) Заголовок, який дозволяє клієнту передати інформацію про запит.
3) Шлях до ресурсу.
4) Необов'язкове тіло повідомлення, яке містить дані.
5) HTTP-дієслова

Існує чотири основних HTTP-дієслова, які використовуються в запитах для взаємодії з ресурсами в REST-системі:

1) Дієслово GET використовується для отримання конкретного ресурсу, або за ідентифікатором, або за колекцією.
2) Дієслово POST використовується для створення нового ресурсу.
3) Дієслово PUT використовується для оновлення певного ресурсу, також за ідентифікатором.
4) Дієслово DELETE використовується для видалення певного ресурсу, також за ідентифікатором.

У заголовку запиту клієнт надсилає тип вмісту, який він може отримати від сервера. Це поле називається "Прийняти", і воно гарантує, що сервер не надсилає

дані, які не можуть бути зрозумілі або оброблені клієнтом. Варіанти типів вмісту - це типи MIME (або багатоцільові розширення інтернет-пошти).

MIME-типи, які використовуються для визначення типів вмісту в полі Accept, складаються з типу і підтипу. Вони розділені косою рискою (/).

Наприклад, текстовий файл, що містить HTML, буде вказано з типом text/html. Якщо цей текстовий файл містить CSS, його буде вказано як text/css. Звичайний текстовий файл буде позначено як text/plain. Однак це значення за замовчуванням, text/plain, не є універсальним. Якщо клієнт очікує text/css, а отримує text/plain, він не зможе розпізнати вміст.

Інші типи та часто використовувані підтипи:

- зображення - image/png, image/jpeg, image/gif
- аудіо - audio/wav, audio/mpeg
- відео - video/mp4, video/ogg
- application - application/json, application/pdf, application/xml, application/octet-stream

Запити повинні містити шлях до ресурсу, на якому має бути виконана операція. У RESTful API шляхи повинні бути розроблені таким чином, щоб допомогти клієнту зрозуміти, що відбувається.

Умовно, перша частина шляху має бути у формі множини ресурсу. Це робить вкладені шляхи легкими для читання і розуміння.

Шлях типу fashionboutique.com/customers/223/orders/12 зрозумілий, на що він вказує, навіть якщо ви ніколи раніше не бачили цей конкретний шлях, тому що він ієрархічний і описовий. Ми бачимо, що отримуємо доступ до замовлення з ідентифікатором 12 для клієнта з ідентифікатором 223.

Шляхи повинні містити інформацію, необхідну для пошуку ресурсу з потрібним ступенем конкретизації. При посиланні на список або колекцію ресурсів не завжди потрібно додавати ідентифікатор. Наприклад, POST-запит до шляху fashionboutique.com/customers не потребує додаткового ідентифікатора, оскільки сервер згенерує ідентифікатор для нового об'єкта.

Якщо ми намагаємося отримати доступ до одного ресурсу, нам потрібно додати до шляху ідентифікатор.

Наприклад: GET fashionboutique.com/customers/:id - отримує елемент в ресурсі customers із зазначеним id. DELETE fashionboutique.com/customers/:id - видаляє елемент з ресурсу customers із зазначеним id.

У випадках, коли сервер надсилає клієнту корисне навантаження даних, сервер повинен включити тип контенту в заголовок відповіді. Це поле заголовка типу вмісту попереджає клієнта про тип даних, які він надсилає в тілі відповіді. Ці типи вмісту є MIME-типами, так само, як і в полі accept заголовка запиту. Тип контенту, який сервер надсилає у відповіді, має бути одним з варіантів, вказаних клієнтом у полі accept запиту.

Відповіді від сервера містять коди статусу, які сповіщають клієнта про успішність виконання операції (див. таблицю 2.3).

Таблиця 2.3 – Типові коди статусу відповідей запитів

Код стану	Опис
200 (ОК)	Це стандартна відповідь для успішних HTTP-запитів.
201 (СТВОРЕНО)	Це стандартна відповідь на HTTP-запит, який призвів до успішного створення елемента.
204 (БЕЗ ВМІСТУ)	Це стандартна відповідь для успішних HTTP-запитів, коли в тілі відповіді нічого не повертається.
400 (ПОГАНИЙ ЗАПИТ)	Запит не може бути оброблений через неправильний синтаксис запиту, надмірний розмір або іншу помилку клієнта.
403 (ЗАБОРОНЕНО)	Клієнт не має дозволу на доступ до цього ресурсу.
404 (НЕ ЗНАЙДЕНО)	Наразі ресурс не вдалося знайти. Можливо, він був видалений або ще не існує.

Продовження таблиці 2.3

500 (ВНУТРІШНЯ ПОМИЛКА СЕРВЕРА)	Загальна відповідь на випадок несподіваного збою, якщо немає більш конкретної інформації.

Для кожного HTTP-дієслова існують очікувані коди стану, які сервер повинен повернути в разі успіху:

GET - вертає 200 (OK)

POST - вертає 201 (CREATED)

PUT - вертає 200 (OK)

DELETE - вертає 204 (NO CONTENT) Якщо операція завершилася невдачею, поверніть максимально конкретний код стану, що відповідає проблемі, яка виникла.

2.9 NPM

NPM (Node Package Manager) — це менеджер пакетів для мови програмування JavaScript, який є за замовчуванням для середовища виконання Node.js. Він складається з клієнта командного рядка, також називаємого npm, а також з онлайн-реєстру публічних і приватних пакетів, який називається реєстром npm. Цей реєстр доступний через клієнт, а доступні пакети можна переглядати та шукати на веб-сайті npm. Менеджером пакетів та реєстром керує компанія npm, Inc.

Менеджер пакетів Node.js (npm) був розроблений Ісааком З. Шлютером (Isaac Z. Schlueter) як рішення для усунення недоліків існуючих модульних систем пакування. Шлютер створив npm як альтернативу системам PEAR (PHP) та CPAN (Perl).

Інсталятор Node.js включає npm як рекомендовану функцію. npm - це клієнт командного рядка, який взаємодіє з віддаленим реєстром. Він дозволяє користувачам використовувати та поширювати модулі JavaScript. Пакунки в реєстрі мають формат CommonJS і включають файли метаданих у форматі JSON. Основний реєстр npm наразі містить понад 477 000 пакунків. У реєстрі відсутня процедура верифікації, що може призвести до включення до нього пакунків

сумнівної якості або безпеки. Замість цього npm покладається на повідомлення користувачів для видалення пакунків, які порушують політику безпеки (наприклад, небезпечні, шкідливі або низької якості). npm надає статистику, включаючи кількість завантажень і пакунків, щоб допомогти розробникам оцінити якість пакунків.

Менеджер пакетів npm здатний керувати як локальними, так і глобальними залежностями JavaScript. При використанні в якості менеджера залежностей для локального проекту npm дозволяє встановити всі залежності проекту однією командою через файл package.json. Крім того, файл package.json дозволяє розробникам визначати діапазон допустимих версій для кожної залежності за допомогою семантичної схеми керування версіями. Це дозволяє розробникам автоматично оновлювати свої пакунки, уникаючи небажаних змін.

2.10 WebStorm

WebStorm - це інтегроване середовище розробки (IDE) для JavaScript, HTML і CSS від JetBrains, розроблене на платформі IntelliJ IDEA. Це спеціалізована версія PhpStorm, що пропонує підмножину його можливостей. WebStorm постачається з попередньо встановленими JavaScript плагінами (наприклад, для Node.js), які доступні для PhpStorm без додаткової оплати.

WebStorm підтримує мови JavaScript, CoffeeScript, TypeScript та Dart.

WebStorm забезпечує автозавершення, аналіз коду на льоту, навігацію по коду, рефакторинг, деприваційю та інтеграцію з системами контролю версій. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (включаючи рефакторинг JavaScript коду, що міститься в різних файлах і папках проекту, а також вбудованого в HTML). Підтримується множинна вкладеність (коли Javascript-скрипт вкладений в HTML-документ, який містить інший HTML-код, всередині якого вкладений JavaScript) - в таких структурах підтримується коректний рефакторинг.

Програмне забезпечення пропонує інтеграцію з різними системами контролю версій, включаючи Subversion, Git, GitHub, Perforce, Mercurial та CVS.

Ця інтеграція дозволяє користувачам створювати списки змін і змін, що очікують на затвердження.

Крім того, програмне забезпечення сумісне з різними системами відстеження помилок.

Підтримується віддалене розгортання за допомогою протоколів File Transfer Protocol (FTP), Secure File Transfer Protocol (SFTP), змонтованих мережевих дисків та інших подібних методів, а також можливість автоматичної синхронізації.

Можливості Zen Coding та Emmet

Підтримка:

- Веб - Angular, React, Vue.js
- Сервер - Node.js, Meteor
- Мобільні - Ionic, Cordova, React Native
- Робочий стіл - Електрон

2.11 Висновки до другого розділу

У другому розділі було проведено аналіз основних технологій, що використовуються для розробки сучасних веб-додатків. Розглянуті інструменти та бібліотеки забезпечують широкий спектр можливостей для створення функціональних, інтерактивних і зручних у користуванні застосунків.

HTML та CSS є фундаментальними технологіями для побудови структури та оформлення веб-сторінок, забезпечуючи їхній візуальний вигляд і доступність.

JavaScript як мова програмування дозволяє додавати інтерактивність до веб-додатків, створювати динамічний контент і працювати з елементами DOM.

TypeScript додає типізацію до JavaScript, що підвищує надійність і передбачуваність коду, сприяє зменшенню кількості помилок і полегшує розробку великих проектів.

React є популярною бібліотекою для створення компонентів користувацького інтерфейсу, що забезпечує повторне використання коду, оптимізацію продуктивності та ефективне управління станом додатку.

Next.js доповнює React, пропонуючи можливості серверного рендерингу, генерації статичних сайтів і покращення продуктивності завдяки оптимізації завантаження сторінок.

AJAX забезпечує асинхронний обмін даними між клієнтом і сервером, що дозволяє оновлювати частини веб-сторінки без її повного перезавантаження.

Radix UI надає набір готових до використання компонентів для створення інтуїтивно зрозумілого інтерфейсу, дотримуючись принципів доступності.

REST визначає архітектурний стиль взаємодії між клієнтом і сервером, забезпечуючи стандартизовані способи доступу до ресурсів.

NPM слугує менеджером пакетів для JavaScript, що спрощує встановлення та управління бібліотеками і залежностями у проектах.

WebStorm — інтегроване середовище розробки, яке забезпечує зручну роботу з кодом, підтримуючи підсвічування синтаксису, автоматичне завершення та інструменти для дебагу.

Таким чином, поєднання цих технологій дозволяє створювати сучасні, продуктивні та масштабовані веб-додатки, забезпечуючи зручність як для розробників, так і для кінцевих користувачів.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ З ІМПЛЕМЕНТАЦІЄЮ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

В рамках виконання досліджень було розроблено прототип тестової версії веб-сайту для продажу товару,а саме норійні ковші, кріплення норійні,скребки та інша продукція з полімерних матеріалів. Був розроблений веб-додаток для представлення контактної інформації компанії та власників. Також був створений тестовий зразок каталогу для цих товарів.

3.1 Концепція програми

Сайт-каталог — це веб-додаток, що дозволяє користувачам переглядати список товарів чи послуг з можливістю пошуку, сортування та фільтрації. У випадку SPA (Single Page Application) основний контент динамічно завантажується без повного перезавантаження сторінки, забезпечуючи плавний користувацький досвід.

Next.js дозволяє створювати SPA із SEO-оптимізацією та підтримкою серверного рендерингу (SSR), що підвищує швидкість завантаження і пошукову видимість.

3.2 Функціонал тестової версії дослідної програми

Головна сторінка (/): Презентація ключових категорій каталогу та пошук товарів.

Сторінка товарів та категорій (/products): Сторінка товарів та фільтрів по категоріям.

Сторінка інформації про компанію (/about-us/): Відображення інформації про компанію

Сторінка товару (/products/[productId]): Детальна інформація про вибраний товар.

Сторінка контактів (/contacts): Детальна інформація про вибраний товар.

3.3 Інтерфейс програми

Графічний інтерфейс веб-додатку складається з наступних вікон:

1) Головна сторінка з хедером,футером та основною інформацією (рис. 3.1 та 3.2):

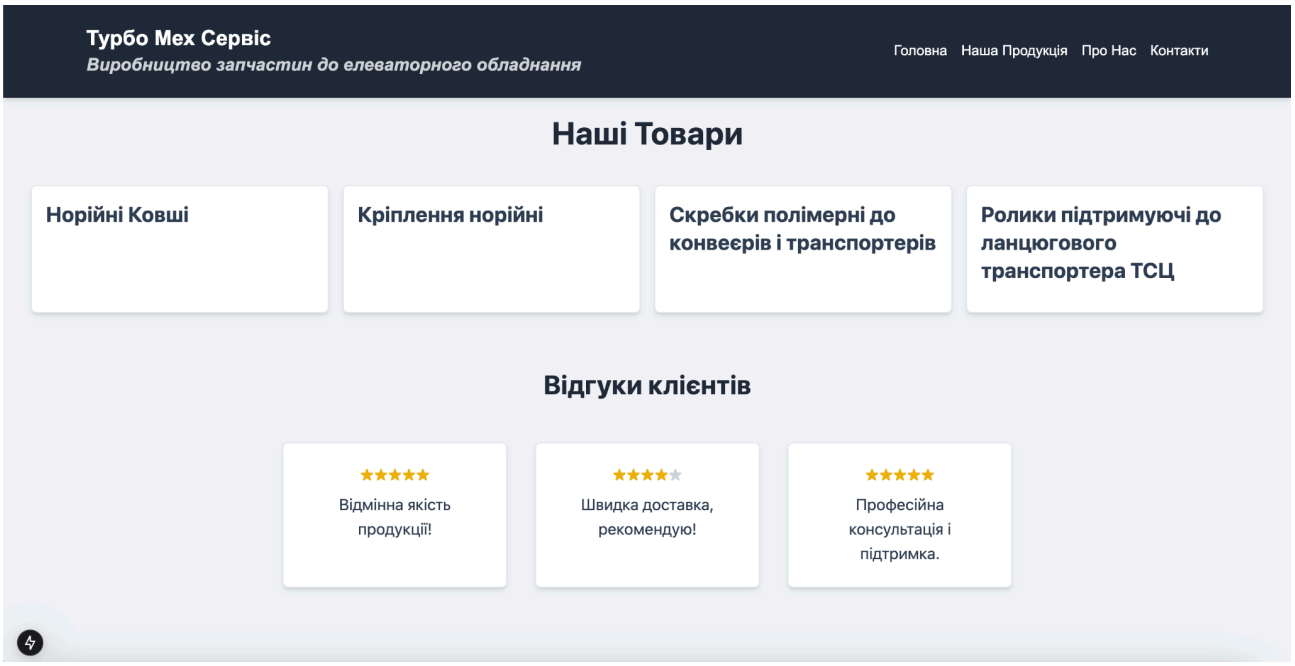

Рисунок 3.1 – Головна сторінка

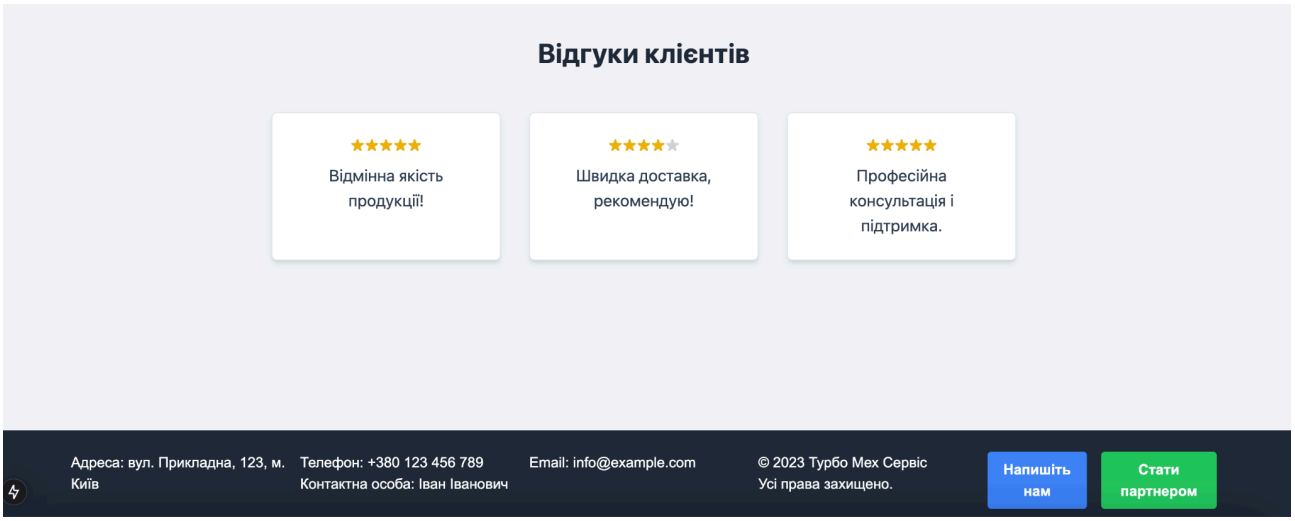

Рисунок 3.2 – Головна сторінка

2) Сторінка товарів компанії. Наявний фільтр по категоріям. Кожен елемент є посиланням на детальну сторінку товару (рис. 3.3 та 3.4):

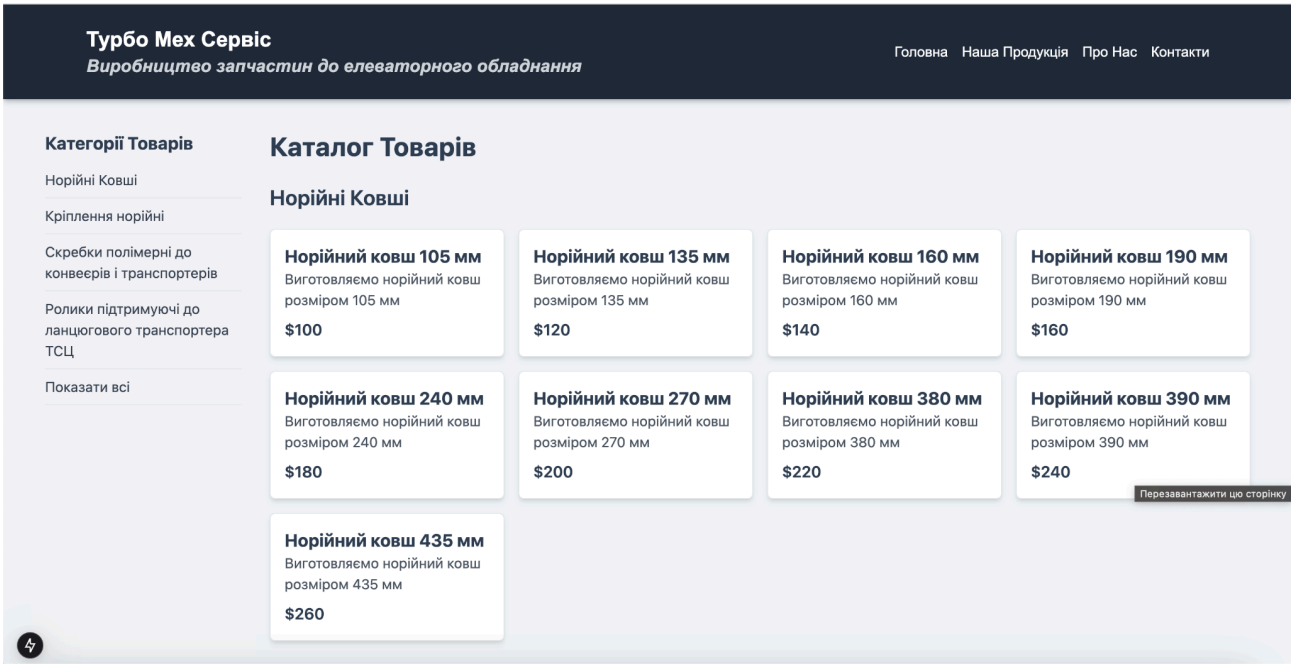

Рисунок 3.3 – Сторінка “Наша Продукція”

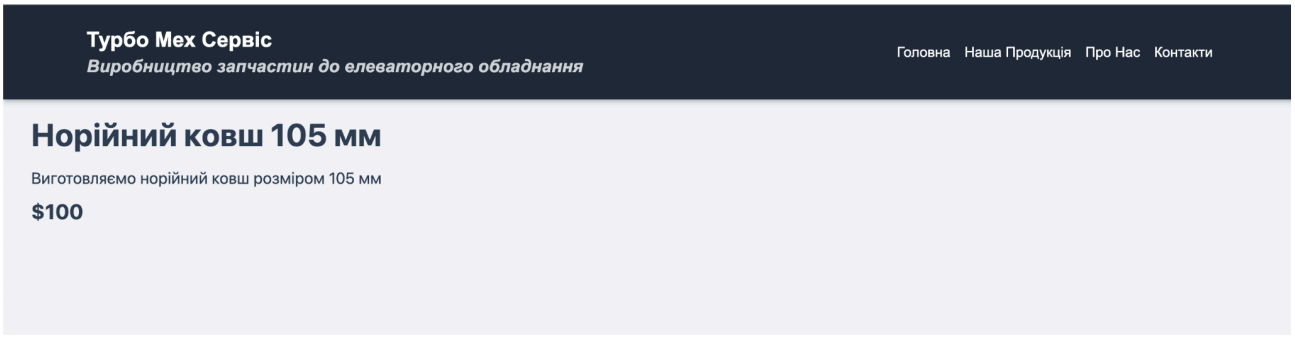

Рисунок 3.4 – Сторінка “Товару”

3) Сторінка “Про нас”. Містить інформацію про опис компанії,її оцінки своїх переваг та ризиків (рис. 3.5):

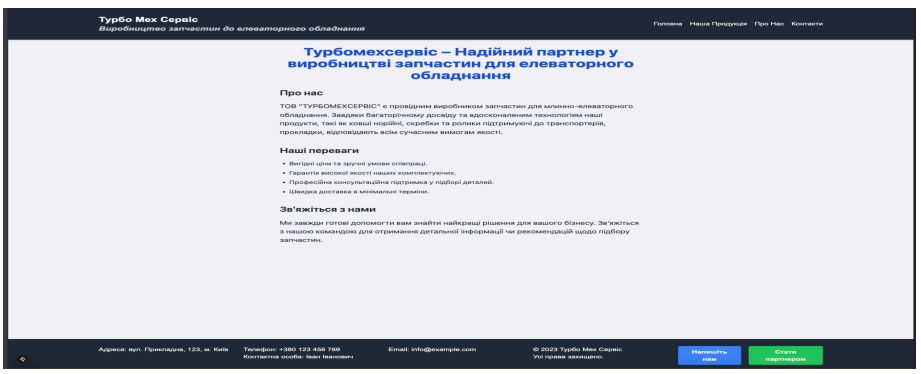

Рисунок 3.5 – Сторінка "Про Нас"

4) Сторінка “Контакти”. Містить основну інформацію про компанію,а саме телефон,адреса ,тощо. (рис. 3.6):

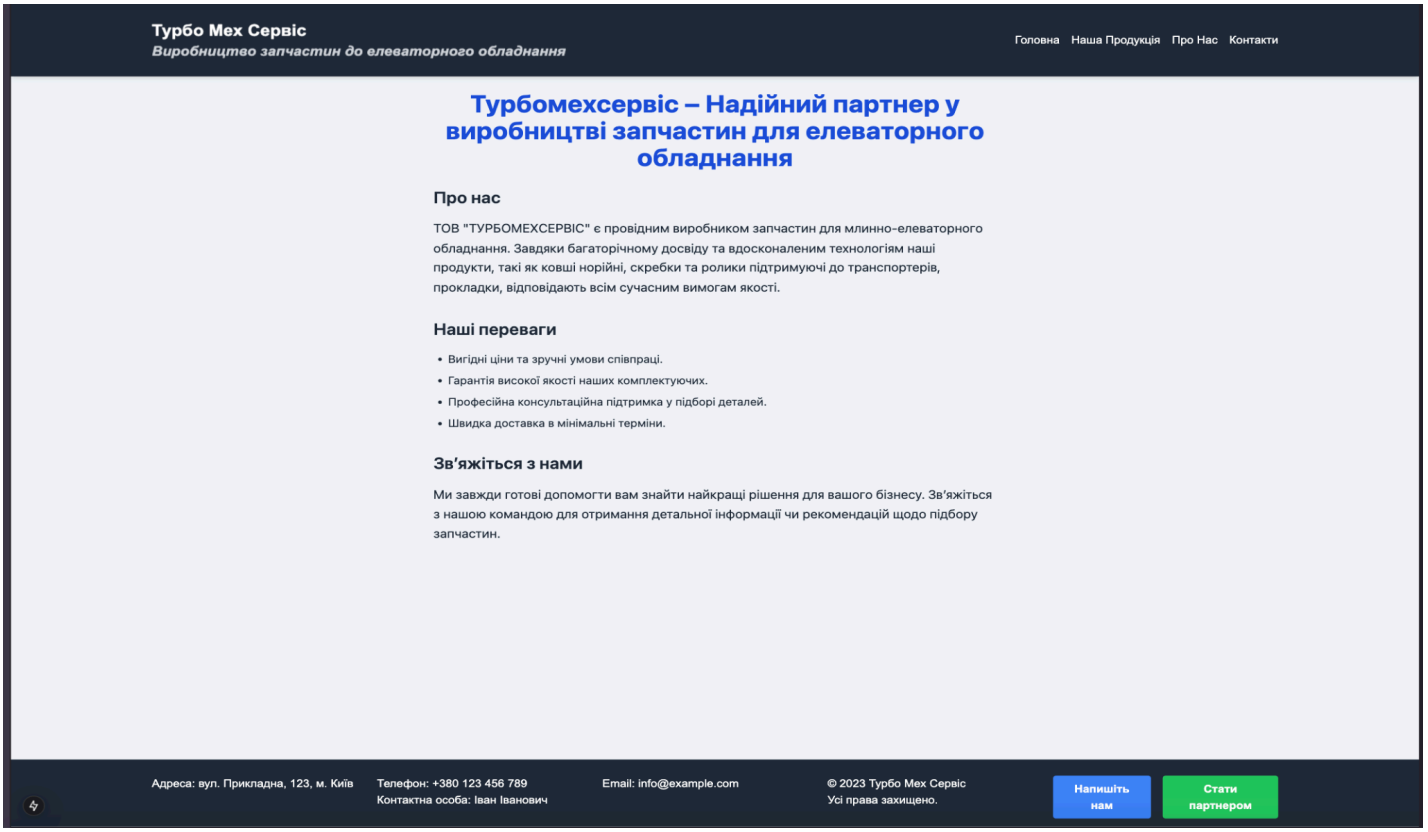

Рисунок 3.6 – Сторінка “Контакти”

3.4 Реалізація основних компонентів веб-додатку

Веб-додаток реалізований на мові програмування JavaScipt. Для розробки інтерфейсів використовувалась бібліотека React,Radix UI.

Для стилізації використовувались CSS та Tailwind CSS.

Проект був ініціалізований за допомогою Next.js,що спрощує роутинг та пропис метаданих товарів та сторінок. Next.js також оптимізовує рендеринг сторінок,оскільки всі сторінки по замовчуванню проходять рендер на сервері (окрім тих сторінок,де потрібна взаємодія зі станом ,наприклад “Наша продукція”, оскільки вона включає в себе фільтрацію елементів)

Програма складається з наступних компонентів представлена в діаграмі класів (див. рис. 3.7):

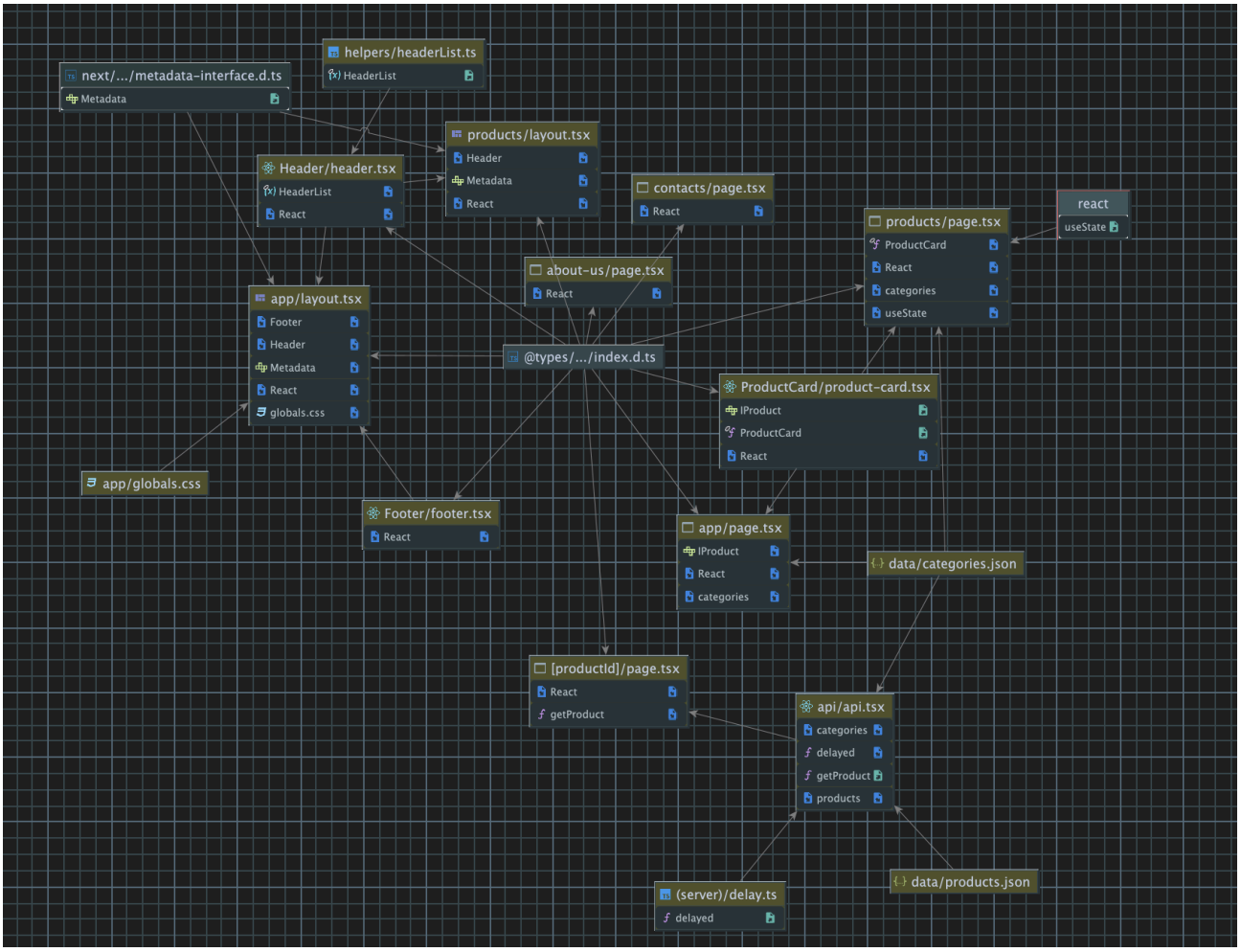

Рисунок 3.7 – Діаграма проекту

1) Компонент MainPage

Цей компонент є головною сторінкою веб-додатку. (див. рис. 3.8 - 3.11)

```
interface ICategory {  Show usages
    categoryName: string;
    products: IProduct[];
}

interface Testimonial {  Show usages
    id: number;
    text: string;
    image: string;
    rating: number;
}
```

Рисунок 3.8 – Інтерфейси компоненту реалізовані через TypeScript

```
const CategoryCard: React.FC<ICategory> = ({ categoryName, products }) => (  Show usages
    <div className="border rounded-md p-4 shadow-md bg-white w-full">
        <h2 className="text-2xl font-bold mb-4">{categoryName}</h2>
    </div>
);



const testimonials: Testimonial[] = [
    { id: 1, text: "Відмінна якість продукції!", image: "/images/client1.jpg", rating: 5 },
    { id: 2, text: "Швидка доставка, рекомендую!", image: "/images/client2.jpg", rating: 4 },
    { id: 3, text: "Професійна консультація і підтримка.", image: "/images/client3.jpg", rating: 5 },
];
```

Рисунок 3.9 – Початкові дані блоку “Відгуків” та елемент Category Card

```
const TestimonialSlider: React.FC = () => {  Show usages
    return (
        <div className="mt-16 w-full">
            <h2 className="text-3xl font-bold text-center text-gray-800 mb-8">Відгуки клієнтів</h2>
            <div className="flex space-x-8 p-4 overflow-hidden animate-scroll lg:justify-center">
                {testimonials.map(({ id, text, image, rating }) => (
                    <div key={id} className="flex-shrink-0 border rounded-md p-6 shadow-md bg-white w-64">
                        {/*<img src={image} alt={`client-${id}`} className="w-16 h-16 rounded-full mx-auto mb-4"
                        <div className="flex justify-center mb-2">
                            {[...Array(rating)].map((_, i) => (
                                <span key={i} className="text-yellow-500">★</span>
                            ))}
                            {[...Array(5 - rating)].map((_, i) => (
                                <span key={5 - i} className="text-gray-300">★</span>
                            ))}
                        </div>
                        <p className="text-lg text-center">{text}</p>
                    </div>
                ))}
            </div>
        </div>
    );
};
```

Рисунок 3.10 – Компонента Слайдеру,які відображають відгуки

```tsx
const Home: React.FC = () => {  Show usages
    return (
        <div className="min-h-screen p-8 pb-20 font-sans">
            <h1 className="text-4xl font-bold text-center text-gray-800 mb-10">
                Наші Товари
            </h1>
            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-4 mt-5 w-full">
                {categories.map((category: ICategory) => (
                    <CategoryCard key={category.categoryName} {...category} />
                ))}
            </div>
            <TestimonialSlider />
        </div>
    );
};

export default Home;  no usages
```

Рисунок 3.11 – Основна компонента Home Page

2) Реалізація імітації серверу (server)

Ця частина проекту створена для імітації роботи серверу та обробки даних бек-ендом. (див. рис. 3.12)

```tsx
import categories from '@/app/(server)/data/categories.json'
import products from '@/app/(server)/data/products.json'
import {delayed} from "@/app/(server)/delay";


export function getAllCategoriesWithProducts() {  no usages
    return delayed(categories, { timeout: 0 });
}

export function getAllProducts() {  no usages
    return delayed(products, { timeout: 0 });
}

export function getProduct(productId: number) {  Show usages
    const product = products.find((x) => x.id === productId);

    if (!product) {
        throw new Error("Product not found.");
    }


    return delayed(product);
}
```

Рисунок 3.12 – Тіло файлу api.tsx

В папці (server)/data - Знаходяться 2 файла у форматі JSON ,які представляють собою дані товарів для веб-додатку

В файлі (server)/delay.ts - Представлена імітація роботи REST API (див. рис. 3.13)

```
type DelayOptions =
    | { timeout: number; shouldFail?: boolean }
    | { timeout?: number; shouldFail: boolean };

export function delayed<T>(  Show usages
    data: T,
    options: DelayOptions = { shouldFail: false, timeout: 1500 },
) {
    const { shouldFail, timeout } = options;
    return new Promise<T>((resolve, reject) => {
        setTimeout(() => (shouldFail ? reject(data) : resolve(data)), timeout);
    });
}
```

Рисунок 3.13 – Тіло файлу delay.ts

3) Компонент About-Us

Цей компонент є сторінкою розширеної інформації про компанію.

Сторінка включає в себе

Рендер відбувається на сервері оскільки має лише статичні дані та задля інтеграції метаданих в сторінку та покращення результативності в пошуку в браузерах. (див. рис. 3.14)

```tsx
export const metadata: Metadata = {  no usages
    title: "Turbo-Mex-Service — About Us",
    description: "Турбомехсервіс — Надійний партнер у виробництві запчастин для елеваторного обладнання",
};

const Page = () => (  Show usages
    <div className="max-h-screen h-full p-8 font-sans  text-gray-800">
        <div className="max-w-3xl mx-auto h-full flex flex-col justify-between">
            <h1 className="text-4xl font-bold mb-6 text-center text-blue-700">
                Турбомехсервіс — Надійний партнер у виробництві запчастин для елеваторного обладнання
            </h1>
            <section className="mb-8">
                <h2 className="text-2xl font-semibold mb-4">Про нас</h2>
                <p className="text-lg leading-relaxed">
                    ТОВ "ТУРБОМЕХСЕРВІС" є провідним виробником запчастин для млинно-елеваторного обладнання. Завдяки багаторічному досвіду та
                </p>
            </section>
            <section className="mb-8">
                <h2 className="text-2xl font-semibold mb-4">Наші переваги</h2>
                <ul className="list-disc pl-5 space-y-2">
                    <li>Вигідні ціни та зручні умови співпраці.</li>
                    <li>Гарантія високої якості наших комплектуючих.</li>
                    <li>Професійна консультаційна підтримка у підборі деталей.</li>
                    <li>Швидка доставка в мінімальні терміни.</li>
                </ul>
            </section>
            <section>
                <h2 className="text-2xl font-semibold mb-4">Зв'яжіться з нами</h2>
                <p className="text-lg leading-relaxed">
                    Ми завжди готові допомогти вам знайти найкращі рішення для вашого бізнесу. Зв'яжіться з нашою командою для отримання деталі
                </p>
            </section>
        </div>
```

Рисунок 3.14 – Тіло файлу about-us/page.tsx

4) Компонент Contacts

Цей компонент є сторінкою для відображення контактних даних,завдяки яким клієнт може зв'язатися з компанією та інша контактна інформація. (див. рис. 3.15)

```tsx
const Page = () => (  Show usages
    <div className="min-h-screen p-8 font-sans bg-gray-50 text-gray-900">
        <div className="max-w-2xl mx-auto bg-white p-6 rounded-lg shadow-lg">
            <h1 className="text-3xl font-bold text-center mb-6 text-blue-700">Контакти</h1>
            <div className="mb-4">
                <h2 className="text-xl font-semibold mb-2">Наша адреса</h2>
                <p className="text-lg">
                    Україна, Полтавська обл., Полтава,<br />
                    вул. Вулична, 9, м. Полтава
                </p>
            </div>
            <div className="mb-4">
                <h2 className="text-xl font-semibold mb-2">Телефон</h2>
                <p className="text-lg">
                    <a href="tel:+3801232233" className="text-blue-500 hover:underline">
                        +3801232233
                    </a>
                </p>
            </div>
            <div className="mb-4">
                <h2 className="text-xl font-semibold mb-2">Email</h2>
                <p className="text-lg">
                    <a href="mailto:turbomech@ukr.net" className="text-blue-500 hover:underline">
                        turbomech@ukr.net
                    </a>
                </p>
            </div>
            <div className="mb-4">
                <h2 className="text-xl font-semibold mb-2">Сайт</h2>
                <p className="text-lg">
                    <a href="https://turbomech.prom.ua/ua/" target="_blank" rel="noopener noreferrer" className="text-blue-500
```

Рисунок 3.15 – Тіло файлу contacts/page.tsx

5) Компонент Products/[productId]/page.tsx

Цей компонент є динамічною сторінкою для кожного окремого елемента product. Параметром визначення елементу є ідентифікатор продукту. За допомогою цього ідентифікатора виконується запит getProduct(productId) (див. рис. 3.16)

```tsx
import React from 'react';
import {getProduct} from "@/app/(server)/api/api";



const Page = async (props:any) => {  Show usages
    const {productId} = await props.params;
        try {
            const product = await getProduct(Number(productId));
            return (
                <div className="min-h-screen p-8 font-sans">
                    <h1 className="text-4xl font-bold mb-4">{product.name}</h1>
                    <p className="text-lg mb-2">{product.description}</p>
                    <p className="text-2xl font-bold">{product.price}</p>
                </div>
            );

        }catch (e) {
            return (
            <div className="min-h-screen p-8 font-sans">
                <h1 className="text-4xl font-bold mb-4">Page not found</h1>
            </div>
            )
        }
};

export default Page;  no usages
```

Рисунок 3.16 – Тіло файлу products/[productId]/page.tsx

6) Компонент Products

Цей компонент є сторінкою для відображення категорій та товарів відфільтрованих по цим категоріям. Примітно те,що в цьому компоненті вже використовується режим рендеру на стороні клієнта,оскільки взаємодія зі станом та фільтрація елементів відбувається на стороні клієнта. (див. рис. 3.17)

```tsx
'use client'
import React, { useState } from 'react';
import { ProductCard } from "@/app/components/ProductCard/product-card";
import categories from '@/app/(server)/data/categories.json';

const CatalogPage: React.FC = () => {  Show usages
    const [selectedCategory, setSelectedCategory] = useState<string | null>('Норійні Ковші');

    const handleCategoryClick = (categoryName: string) => {  Show usages
        setSelectedCategory(categoryName);
    };


    const displayedCategories = selectedCategory
        ? categories.filter(category => category.categoryName === selectedCategory)
        : categories;

    return (
        <div className="flex">
            <aside className="bg-gray-100 p-4 w-64">
                <h2 className="text-xl font-bold mb-4">Категорії Товарів</h2>
                <ul className="space-y-2">
                    {categories.map((category, index) => (
                        <li key={index} className="border-b pb-2">
                            <button
                                onClick={() => handleCategoryClick(category.categoryName)}
                                className="text-gray-700 hover:text-blue-500 w-full text-left"
                            >
                                {category.categoryName}
                            </button>
                        </li>
                    ))}
                    <li className="border-b pb-2">
                        <button
                            onClick={() => setSelectedCategory(null)}
```

Рисунок 3.17 – Тіло файлу products/page.tsx

ВИСНОВКИ

1. Проаналізовано ключові аспекти архітектури та безпеки SPA-додатків на основі Next.js. Особливу увагу приділено поширеним атакам на веб-додатки, таким як XSS, CSRF та атакам через підроблені DNS-запити. Встановлено, що основна уразливість полягає у можливості отримання зловмисних відповідей від сервера або підроблених запитів, що підкреслює важливість захисту від зовнішніх загроз через правильну конфігурацію CORS, CSP (Content Security Policy) та DNS-фільтрацію.

2. Розглянуто сучасні методики підвищення безпеки додатків, серед яких: шифрування трафіку (HTTPS), використання httpOnly та secure cookies для захисту сесій, а також інтеграція веб-брендмауерів та систем реагування на загрози (WAF). Визначено, що перспективним напрямом для забезпечення адаптивного захисту є інтеграція технологій штучного інтелекту та машинного навчання для виявлення аномалій у поведінці користувачів та запитів.

3. Розроблено прототип SPA-додатку на Next.js, який дозволяє динамічно взаємодіяти з бекендом та забезпечує відображення даних із застосуванням методів захисту інформації. Додаток включає функції динамічного рендерингу контенту, управління станом та оптимізацію запитів через API Routes.

4. Використання прототипу дозволило оцінити продуктивність та безпеку різних методів рендерингу в Next.js (SSR, SSG, ISR), а також виявити аномалії у часі відповіді API-запитів та завантаженні компонентів. Було виявлено, що оптимізація мережевих запитів та кешування контенту суттєво впливає на загальну продуктивність додатку та користувацький досвід.

5. Результати дослідження можуть бути використані для оптимізації налаштувань додатків, вибору оптимальних методів захисту даних, а також моніторингу продуктивності та виявлення потенційних вразливостей. Інтеграція захисних механізмів, таких як CSP, шифрування запитів та багатофакторна

аутентифікація, дозволяє підвищити рівень безпеки SPA на Next.js та захистити користувацькі дані від сучасних кіберзагроз.

6. Перспективами подальшої розробки програми є удосконалення користувацького інтерфейсу, створення інструментів автоматичного аналізу вихідних даних та інтеграція технологій штучного інтелекту з метою виявлення аномалій у DNS-трафіку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. НОРМАТИВНИЙ ДОКУМЕНТ СИСТЕМИ ТЕХНІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ. Режим доступу: https://tzi.com.ua/downloads/2.5-010-03.pdf
2. Top 10 Web Application Security Risks. Режим доступу: https://owasp.org/www-project-top-ten/
3. Процес управління вразливостями в інформаційній системі органів державної статистики. Режим доступу: https://www.ukrstat.gov.ua/norm_doc/2024/195/prosec.pdf
4. Звіт по вебвразливостях 2021. Режим доступу: https://corewin.ua/news/acunetix-web-application-vulnerability-report-2021/
5. HTML: HyperText Markup Language. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML
6. HTML basics. Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics
7. CSS: Cascading Style Sheets. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS
8. CSS basics. Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics
9. JavaScript. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
10. Introduction to JavaScript. Режим доступу: https://www.geeksforgeeks.org/introduction-to-javascript/
11. TypeScript for JavaScript Programmers Risks. Режим доступу: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

12. Amler V. ReactJS by Example - Building Modern Web Applications with React / V. Amler, P. Sonpatki., 2016. – 280 c. – (2nd Edition).
13. Radix UI- What is Radix UI Used For? Режим доступу: https://pwskills.com/blog/radix-ui/
14. What is REST? Режим доступу: https://www.codecademy.com/article/what-is-rest
15. JSDoc comments | WebStorm Documentation. Режим доступу: https://www.jetbrains.com/help/webstorm/meet-webstorm.html
16. npm Docs, Documentation for the npm registry, website, and command-line interface. Режим доступу: https://docs.npmjs.com/about-npm
17. Banks A. Learning React: Learning React Functional Web Development with React and Redux / A. Banks, E. Porcello., 2017. – 350 c
18. Stack Overflow Developer Survey 2022: JavaScript — найпопулярніша мова, користування Docker зростає, AWS випереджає інші хмари. Режим доступу: https://dou.ua/forums/topic/38869/