МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

WEB-ПРОГРАМУВАННЯ (CLOUD COMPUTING).
Частина 1

Методичні вказівки до виконання лабораторних занять
для здобувачів першого (бакалаврського) рівня вищої освіти
освітньо-професійної програми «Комп'ютерні науки»
галузі знань F Інформаційні технології
спеціальності F3 Комп'ютерні науки
денної та заочної форм навчання

Луцьк 2026

УДК 004.8(07)
Т 47

До друку
Голова вченої ради факультету КІТ_____І.С.Кондіус

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ
Директор бібліотеки _____Н.П. Поліщук

Затверджено вченою радою факультету КІТ,
протокол № ___ від « » _____ 2026 року.

Розглянуто і схвалено на засіданні кафедри комп'ютерних наук ЛНТУ,
протокол № 7 від « 24 » 01 2026 року.
Завідувач кафедри КН _____ В.О. Ліщина

Укладач: _____ Ю.Й. Тулашвілі, доктор педагогічних наук, професор кафедри комп'ютерних наук ЛНТУ

Рецензент: _____ Н.М. Ліщина, кандидат технічних наук, доцент, завідувач кафедри інженерії програмного забезпечення ЛНТУ.

Відповідальний за випуск: _____ В.О. Ліщина, кандидат технічних наук, доцент, завідувач кафедри комп'ютерних наук ЛНТУ.

Т 47 **Web-програмування (Cloud Computing). Частина 1**: методичні вказівки до виконання лабораторних занять для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерні науки» галузі знань F Інформаційні технології спеціальності F3 Комп'ютерні науки денної та заочної форм навчання / уклад. Ю.Й. Тулашвілі, Луцьк: ЛНТУ, 2026. 60 с.

Видання містить необхідний матеріал, який дасть можливість студентам засвоїти навчальний матеріал з мінімальними витратами часу.

Призначене для студентів спеціальності F3 «Комп'ютерні науки» денної та заочної форми навчання.

ЗМІСТ

ПЕРЕДМОВА

Метою викладання навчальної дисципліни «Web-програмування (Cloud Computing)» є формування знань, вмінь та навичок з Web-програмування з використанням хмарних обчислень, опанування технологіями стеків проектування Web та прийомами їх застосування.

Завданнями вивчення навчальної дисципліни є: вивчити класифікацію Web; поглибити знання з мови розмітки гіпертексту HTML та мови стилів CSS; вивчити стеки проектування Web та їх застосування; вивчити стеки LAMP, MERN та методи програмування на мові JavaScript.

Найменування та опис компетентностей, формування котрих забезпечує вивчення дисципліни

Інтегральна компетентність

Здатність розв’язувати складні спеціалізовані завдання та практичні проблеми у галузі комп’ютерних наук або у процесі навчання, що передбачає застосування теорій та методів інформаційних технологій і характеризується комплексністю та невизначеністю умов.

Загальні компетентності

ЗК1. Здатність до абстрактного мислення, аналізу та синтезу.

ЗК2. Здатність застосовувати знання у практичних ситуаціях.

ЗК3. Знання та розуміння предметної області та розуміння професійної діяльності.

ЗК6. Здатність вчитися й оволодівати сучасними знаннями.

ЗК12. Здатність оцінювати та забезпечувати якість виконуваних робіт.

Фахові компетентності спеціальності

СК8. Здатність проектувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об’єктно-орієнтованого, функціонального, логічного, з відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмами управління.

СК9. Здатність реалізувати багаторівневу обчислювальну модель на основі архітектури клієнт-сервер, включаючи бази даних, знань і сховища даних, виконувати розподілену обробку великих наборів даних на кластерах стандартних серверів для забезпечення обчислювальних потреб користувачів, у тому числі на хмарних сервісах.

СК17. Здатність розробляти Web-додатки із застосуванням сучасних технологій та інструментів.

Передумови для вивчення дисципліни

Викладання курсу «Web-програмування (Cloud Computing)» базуються на знаннях, які студенти отримали при вивчені дисциплін: «Алгоритмізація та програмування», «Операційні системи», «Веб технології та веб дизайн», «Організація баз даних і знань».

Результати навчання

Після завершення вивчення дисципліни студент повинен:

ПР9. Розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв'язання задач в галузі комп'ютерних наук.

ПР10. Використовувати інструментальні засоби розробки клієнтсерверних застосувань, проектувати концептуальні, логічні та фізичні моделі баз даних, розробляти та оптимізувати запити до них, створювати розподілені бази даних, сховища та вітрини даних, бази знань, у тому числі на хмарних сервісах, із застосуванням мов web-програмування.

ПР17. Обирати певну парадигму web-програмування згідно з вимогами до якості при реалізації методів та алгоритмів розв'язання задач в галузі комп'ютерних наук, створювати дизайн web-додатків.

В результаті вивчення дисципліни студенти повинні:

знати:

– технології обробки, зберігання та передачі даних;

– архітектуру клієнт-сервер;

– бази даних;

– хмарні сервіси;

вміти:

– розробляти програмні моделі предметних середовищ;

– використовувати інструментальні засоби розробки клієнт-серверних застосувань;

– розробляти та оптимізувати запити до баз даних, створювати розподілені бази даних;

– обирати певну парадигму web-програмування згідно з вимогами до якості при реалізації методів та алгоритмів розв'язання задач в галузі комп'ютерних наук.

ЛАБОРАТОРНА РОБОТА 1. Написання технічного завдання на розробку web-додатка

Мета: навчитись складати технічне завдання на розробку web-додатка.

1. Теоретичні відомості

Написання технічного завдання (ТЗ) – це фундамент, на якому будується будь-який IT-продукт. Технічне завдання (Software Requirements Specification або SRS) – це документ, який є основним документом з якого розпочинається розробка проекту. ТЗ є точкою дотику між бізнес-ідеєю замовника та технічною реалізацією виконавця. ТЗ є не просто «список побажань», а юридично значущий протокол, що визначає обсяг робіт (Scope), терміни та стандарти якості.

Орієнтовна структура Технічного Завдання

Розділ 1 Загальні положення. Опис проєкту та бізнес-цілі

Містить:

- назва проєкту;
- опис основної цілі (наприклад, автоматизація обліку допомоги, зв'язок між волонтерами та нужденними);
- замовник/виконавець (дані сторін).

Розділ 2 Ролі користувачів

Цільова аудиторія. Хто буде користуватися (адміністратори, користувачі послуг, особи, що можуть переглядати).

Розділ 3 Функціональні вимоги (Functional Requirements)

Для Адміністратора:

- управління базами даних;
- управління контентом сайту;
- збір та подача статистики (текстові/візуальні звіти).

Для Користувачів:

- форма для подачі запиту на послуги;
- перегляд статусу запитів;
- форми для реєстрації та подачі пропозицій;
- можливість спілкування (чат з адміном/іншими користувачами).

Розділ 4 Архітектура

Схематична карта сайту (меню, кнопки).

Розділ 5 Дизайн

Посилання на макети або опис, що будуть розроблені.

Розділ 6 Технічні вимоги

Адаптивність, яка полягає у підтриці різних пристроїв (ПК, планшети, смартфони).

Backend, Frontend, БД – вказати бажані або залишити на розсуд розробника.

Інтеграція з платіжними системами, соцмережами, іншими сервісами.

Розділ 7 Безпека

Вимоги до захисту даних.

2. Послідовність виконання:

1. Ознайомитись із теоретичними вимогами.
2. Обрати прикладну галузь проекту.
3. Розробити ТЗ на проект.
4. Оформити звіт.

3. Приклад виконання

ТЕХНІЧНЕ ЗАВДАННЯ
на розробку інформаційної системи
«Web-додаток підтримки продажів брендового взуття»

Розділ 1 Загальні положення. Опис проєкту та бізнес-цілі

Призначення web-додатку

Головна мета проекту – створення функціонального додатку, орієнтованого на підтримку торгівельної діяльності магазину брендового взуття. Кінцевим продуктом розробки є інтернет-магазин з актуальним каталогом наявних моделей та можливістю фільтрування їх характеристик для зручного та швидкого вибору бажаного товару та подальшого онлайн замовлення через особистий кабінет клієнта.

Візія продукту

Індивідуальною особливістю та однією з найважливіших функцій додатку стане можливість для клієнта перевірити власну модель взуття на оригінальність та унікальність, якщо вона придбана у іншому магазині із сумнівною репутацією. Для цього необхідно лише завантажити фотоматеріали моделі взуття на перевірку менеджером-спеціалістом та зачекати необхідний проміжок часу. Web-додаток буде створено з метою підвищення якості обслуговування клієнтів, збільшення обсягів продажів і, як наслідок, зростання прибутку магазину.

Розділ 2 Ролі користувачів

Цільовою аудиторією розроблюваного додатку є люди, зацікавлені у придбанні рідкісних моделей брендового взуття від всесвітніх виробників за конкурентоспроможними цінами в Україні та хочуть бути впевненими у оригінальності на унікальності товарів, які вони купують.

Як [Адміністратор], я хочу [вносити зміни до контенту] та [генерувати звіт у PDF], щоб [зекономити час на підготовку аналітики].

Як [Відвідувач], я хочу [швидко отримати доступ до контенту].

Як [Клієнт-покупець], я хочу [зареєструватися через Google], щоб [швидко зробити замовлення].

Розділ 3 Функціональні вимоги (Functional Requirements)

Web-додаток підтримки продажів брендового взуття має бути завантажений на сервер через хостинг для користування в мережі Інтернет.

Функціонал продукту має забезпечувати наступні можливості:

– перегляд актуального списку моделей взуття та його характеристик;

– фільтрування товарів каталогу за характеристиками;

– реєстрацію облікового запису користувача та подальшої авторизації;

– сформувати та залишити запит на онлайн замовлення продукції;

– створити запит на онлайн перевірку оригінальності моделі.

Права доступу до його інформації розмежовуються на типи користувачів:

– адміністратор;

– клієнт-покупець;

– відвідувач.

Адміністратори повинні мати особливій обліковий запис, який містить панель з функціоналом для редагування інформації та налаштування додатку, а також функції надання зворотного зв’язку та обробки створених замовлень. Інформація про товари, наявні у каталозі додатку та їх характеристики, а також дані особистих кабінетів клієнтів та їх замовлення має зберігатися у базі даних, реалізованій засобами СУБД MySQL

Клієнту-покупцю необхідно створити особливий кабінет для отримання можливості зберігати до кошика бажані товари та замовляти їх, а також зберігати історію вже виконаних замовлень.

Відвідувач інтернет-магазину не повинен мати облікового запису. Він може вільно переглядати інформації у каталозі та фільтрувати її за характеристиками товару.

Навігація у web-додатку інтернет-магазину являє собою головне меню в шапці сторінок. Меню включає у себе пункти, необхідні для швидкого переміщення користувача по усім наявним розділам додатку. Воно

відображається у шапці кожної сторінки для легкої та швидкої навігації між ними у будь-який момент часу використання додатку.

Розділ 4 Архітектура

Структура додатку представлена набором взаємопов’язаних вебсторінок, які є пунктами головного меню інтернет-магазину.

Розроблюваний web-додаток складається з таких розділів:

– «Головна» – початкова сторінка, яка включає головне меню додатку, карусель фото з хітами продажів та акційними пропозиціями магазину;

– «Чоловікам» – розділ каталогу з чоловічими моделями брендового взуття;

– «Жінкам» – розділ каталогу з жіночими моделями брендового взуття;

– «Бренди» – перелік оригінальних закордонних брендів, чия продукція наявна та доступна для замовлення;

– «Контакти» – сторінка з контактною інформацією магазину та панеллю зворотного зв’язку;

– «Оплата і доставка» – розділ з інформацією про можливі способи оплати та доставки товарів при купівлі у інтернет-магазині;

– «Особистий кабінет» – розділ з панеллю реєстрації та авторизації облікового запису користувача.

Розділ 5 Дизайн

Інтерфейс web-додатку повинен забезпечувати безперешкодне та швидке сприйняття інформації користувачем. Для цього було реалізовано інтуїтивно зрозумілий та сучасний дизайн, основою якого став приємний для сприймання білий колір. Основою контенту мають бути власні фото високої якості, що виконані на студії. Розташування блоків інформації на головній сторінці додатку продемонстровано на рисунку 1.

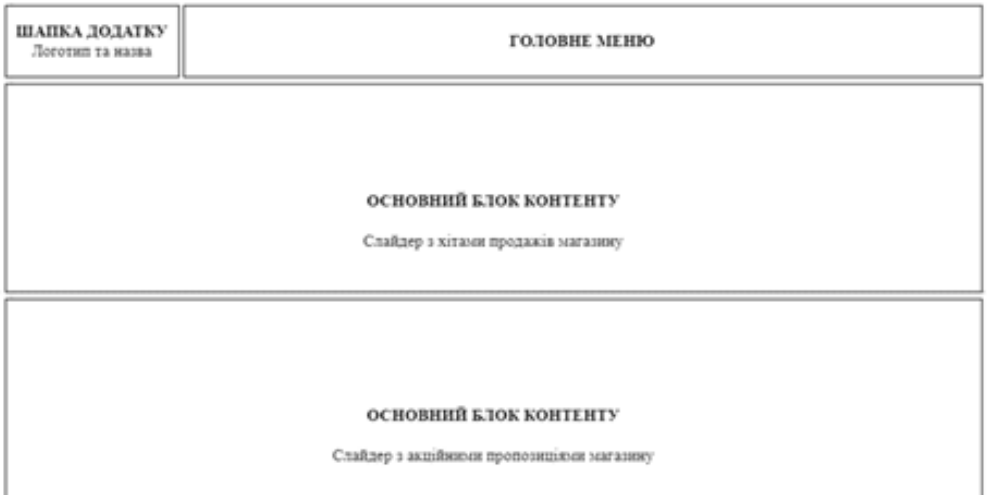

Рисунок 1 – Схема головної сторінки

Система навігації (карта web-додатку) зображена на рисунку 2.

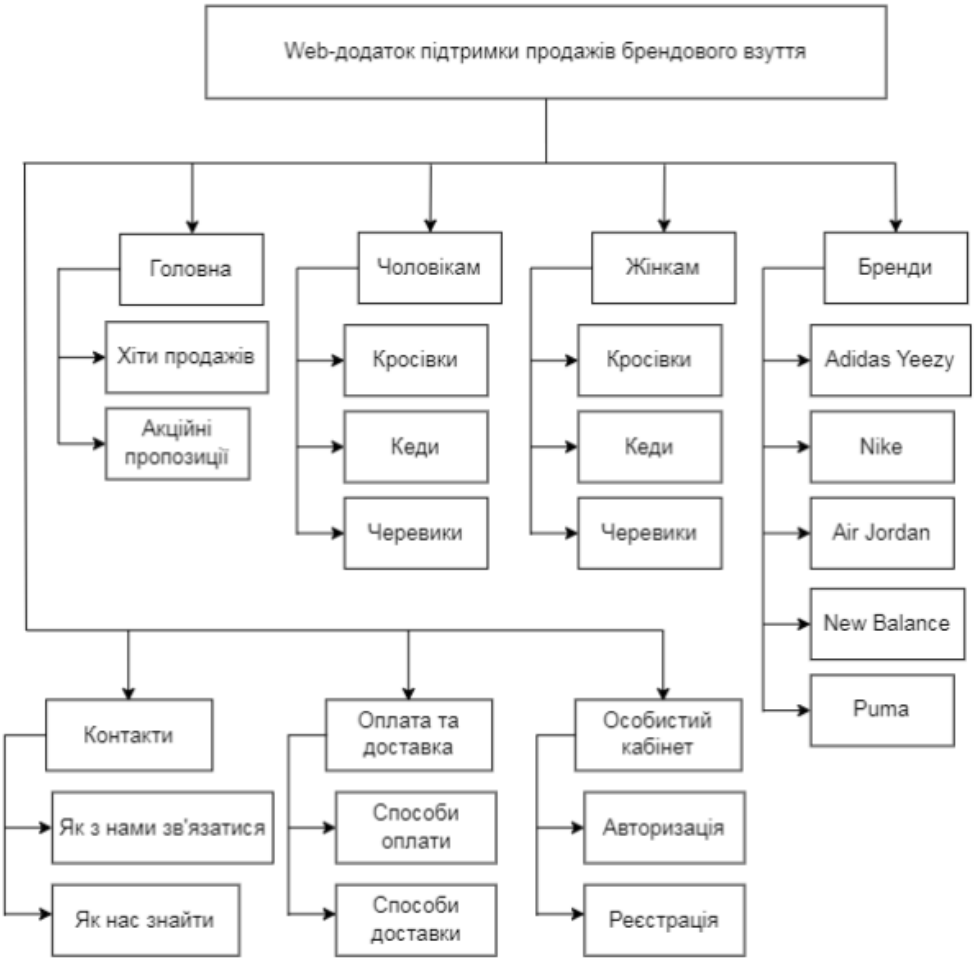

Рисунок 2 – Карта web-додатку

Розділ 6 Технічні вимоги

Реалізація web-додатку відбувається з використанням:

– PHP 7.4.4;

– MySQL 8.0;

– JavaScript.

Вимоги до лінгвістичного забезпечення – web-додаток має бути реалізовано українською мовою.

Вимоги до програмного забезпечення

Програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам – web-браузер: Firefox 3.5 і вище, або Opera 9.5 і вище, або Chrome 2 і вище.

Розділ 7 Безпека

SSL, захист форми зворотного зв’язку та даних клієнтів.

ЛАБОРАТОРНА РОБОТА 2. Використання сайтів-шаблонів вибраних з сайта-репозиторію сайтів-шаблонів templatemonster.com

Мета: навчити студентів знаходити шаблони головної Front-end сторонки web-додатку з використанням (https://www.templatemonster.com).

1. Теоретичні відомості

Front-end – являю собою ту частину web-додатку, яка є публічною та завжди знаходиться на стороні користувачів, надаючи їм потрібну інформацію та сервісну підтримку. Основними завданнями front-end є: відображення контент-інформації; надання функціоналу, що подається розробниками web-додатку на стороні клієнта; створення інтерфейсу для запитів користувачів на отримання потрібних послуг, що опрацьовуються за цими запитами на стороні back-end. Іншими словами, front-end – це все те, що є на web-сторінках web-додатку та те з чим користувач може взаємодіяти. Таким чином, web-додаток складається з двох частин – клієнт-серверного додатку, де клієнтом виступає браузер на стороні користувача; серверного додатку, коли web-сервер обслуговує запити користувача та реалізує інтерфейс сервер-браузер.

Front-end-розробник на початку створення своєї частини web-додатку повинен визначити таке: який дизайн матиме публічна частина web-додатку; які мови програмування будуть застосовані; які данні будуть відображатись, а які будуть передаватись на серверну частину; яким чином буде побудований інтерфейс взаємодії між браузером та сервером тощо.

Основними компетенціями якими повинен володіти front-end-розробник web-додатку на основі стеку LAMP є знання та розуміння з:

– HTML (HyperText Markup Language) – мова розмітки всіх елементів і документів на web-сторінці;

– CSS (Cascading Style Sheets) – мова стилізації зовнішнього вигляду web-сторінки. CSS визначає шрифти, кольори, розташування блоків сайту тощо;

– бібліотеки JavaScript, які на стороні браузера створює можливості інтерактивного реагування на дії користувача.

Web-розробнику використання шаблонів з репозиторіїв дає значну економію часу та ресурсів, прискорений старт проекту, гарантовану структуру й адаптивність (якщо шаблони якісні), а також можливість швидко прототипувати та розгортати рішення, отримуючи готовий функціонал та дизайн, який потім можна кастомізувати, що прискорює процес створення унікального сайту та знижує вартість розробки.

Ключові переваги роботи з шаблонами:

– забезпечує швидкість та ефективність процесу розробки. Замість розробки з нуля, розробник отримує готову основу з HTML/CSS/JS та структурою, що дозволяє зосередитись на кастомізації під потреби клієнта;

– консистентність дизайну. Готові шаблони забезпечують єдиний стиль, що полегшує підтримку та візуальну гармонію проекту;

– зменшення кількості помилок. Якісні шаблони часто вже протестовані, що знижує ризик базових помилок у верстці;

– економія коштів. Зменшується час розробки, а отже, і витрати на проект;

– розширюваність шаблону. Шаблони дозволяють додавати власну логіку.

2. Послідовність виконання:

1.Знайти шаблон сайту використавши сайт шаблонів templatemonster.com або інший.

2. Скачати архів шаблону.

3. Ознайомитись із шаблоном сайту з каскадним меню (видається викладачем).

4. Розібратися з вмістом файлів index.html та style.css використовуючи довідкові ресурси з Internet.

5. Програмувати зміни у скачаному шаблоні, а саме каскадне меню web-сторінки за зразком виданим викладачем та доопрацювати CSS.

6. Оформити звіт.

3. Приклад виконання

Доопрацюємо код файлу index.html щоб утворити каскадне меню. Вигляд web-сторінки з каскадним меню подано на рисунку 3.

Рисунок 3 – Сторінка проекту з каскадним меню

Лістинг 1 : Зміст файлу index.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="stylesheet" href="./style.css" type="text/css" media="screen">
</head>
<body>
<div class="example">
   <ul id="nav">
      <li class="current"><a href="http://www">Home</a></li>
          <li ><a href="http://www">Tutorials</a>
         <ul>
            <li><a href="http://www">HTML / CSS</a></li>
            <li><a href="http://www">JS / jQuery</a>
               <ul>
                  <li><a href="http://www">JS</a></li>
                   <li><a href="http://www">jQuery</a></li>
               </ul>
               </li>
            <li><a href="http://www">PHP</a></li>
            <li><a href="http://www">MySQL</a></li>
            <li><a href="http://www">XSLT</a></li>
            <li><a href="http://www">Ajax</a></li>
             </ul>
              </li>
        <li><a href="http://www">Resources</a></li>
       <li><a href="http://www">About</a></li>
   </ul>
</div>
      </body>
</html>
```

кінець лістингу 1

Запрограмуємо стилі оформлення сторінки у файлі style.css

Лістинг 2 : Зміст файлу style.css

```
body {
    background:#eee;
    margin:0;
    padding:0;
}
.example {
    background: url(./images/background.jpg);
    width:1000px;
    height:600px;
    border:3px #000 solid;
    margin:20px auto;
    padding:15px;
    border-radius:15px;
}
#nav {
    display:inline-block;
    width:100%;
    margin:0px auto;
    padding:0;
    background:#335599 repeat-x 0 -110px;
    border-radius:10px; /*some css3*/
    box-shadow:0 2px 2px rgba(0,0,0, .5);
}
#nav li {
      margin:10px;
    float:left;
    position:relative;
    list-style:none;
}
#nav a {
    font-weight:bold;
    color:#e7e5e5;
    text-decoration:none;
    display:block;
    padding:8px 20px;
```

```
    border-radius:10px; /*some css3*/
    -moz-border-radius:10px;
    -webkit-border-radius:10px;
    text-shadow:0 2px 2px rgba(0,0,0, .7);
}
/* selected menu element */
#nav .current a, #nav li:hover > a {
    background:#7788aa  repeat-x 0 -20px;
      color:#000;
    border-top:1px solid #f8f8f8;
    box-shadow:0 2px 2px rgba(0,0,0, .7); /*some css3*/
    -moz-box-shadow:0 2px 2px rgba(0,0,0, .7);
    -webkit-box-shadow:0 2px 2px rgba(0,0,0, .7);
    text-shadow:0 2px 2px rgba(255,255,255, 0.7);
}
#nav ul li:hover a, #nav li:hover li a {
    background:none;
    border:none;
    color:#000;
}
#nav ul li a:hover {
    background:#335599 repeat-x 0 -100px;
    color:#fff;
    border-radius:10px; /*some css3*/
    -moz-border-radius:10px;
    -webkit-border-radius:10px;
    text-shadow:0 2px 2px rgba(0,0,0, 0.7);
}
#nav ul li:first-child > a {
    -moz-border-radius-topleft:10px; /*some css3*/
    -moz-border-radius-topright:10px;
    -webkit-border-top-left-radius:10px;
    -webkit-border-top-right-radius:10px;
}
#nav ul li:last-child > a {
    -moz-border-radius-bottomleft:10px; /*some css3*/
    -moz-border-radius-bottomright:10px;
    -webkit-border-bottom-left-radius:10px;
    -webkit-border-bottom-right-radius:10px;
```

```
}
#nav li:hover > ul {
   opacity:1;
   visibility:visible;
}
#nav ul {
   opacity:0;
   visibility:hidden;
   padding:0;
   width:175px;
   position:absolute;
   background:#aabbcc  repeat-x 0 0;
   border:1px solid #7788aa;
   border-radius:10px; /*some css3*/
   -moz-border-radius:10px;
   -webkit-border-radius:10px;
   box-shadow:0 2px 2px rgba(0,0,0, .5);
   -moz-box-shadow:0 2px 2px rgba(0,0,0, .5);
   -webkit-box-shadow:0 2px 2px rgba(0,0,0, .5);
   -moz-transition:opacity .25s linear, visibility .1s linear .1s;
   -webkit-transition:opacity .25s linear, visibility .1s linear .1s;
   -o-transition:opacity .25s linear, visibility .1s linear .1s;
   transition:opacity .25s linear, visibility .1s linear .1s;
}
#nav ul li {
   float:none;
   margin:0;
}
#nav ul a {
   font-weight:normal;
   text-shadow:0 2px 2px rgba(255,255,255, 0.7);
}
#nav ul ul {
   left:160px;
   top:0px;
}
```

кінець лістингу 2

ЛАБОРАТОРНА РОБОТА 3. Проектування шаблонів форм для Web-сайтів. Валідація форми засобами HTML5

Мета: навчити студентів компонувати форми для Web-сайтів та здійснювати валідацію даних, що вводяться засобами HTML5.

1. Теоретичні відомості

Форми не використовуються самі по собі. Їх призначення – збирання інформації та передавання її для подальшого опрацювання та збереження різними засобами. Інформація, що введена користувачем, опрацьовується із застосуванням різних мов програмування (javascript, php тощо). Передавання інформації здійснюється у різних форматах обміну даними, наприклад, JSON - текстовому форматі, заснованому на JavaScript, або методом запиту POST, що призначений для запиту, при якому web-сервер приймає дані, вкладені в тіло повідомлення.

Теги form, input, textarea, select і option – базові теги для форм в HTML.

Тег <form> формує такий собі «бланк». Якщо використовується для користувача форма для відправки даних, то потрібно описати атрибут action для вказівки, куди контент буде відправлений.

Атрибут method вказує формі, як дані будуть відправлятися на сервер, також має дефолтний значення get, а також post, що практично непомітно передає інформацію про форму. Значення атрибута method не залежить від регістра.

Метод get є одним з найпоширеніших і призначений для отримання необхідної інформації і передачі даних в адресному рядку. Пари «ім'я = значення» приєднуються в цьому випадку до адреси після знаку питання і розділяються між собою амперсандом (символ &). Зручність використання методу get полягає в тому, що адреса з усіма параметрами можна використовувати неодноразово, зберігши його, наприклад, в закладки браузера, а також змінювати значення параметрів прямо в адресному рядку.

Метод post посилає на сервер дані в запиті браузера. Це дозволяє відправляти більшу кількість даних, ніж є методом get, оскільки у нього встановлено обмеження в 4 Кб. Великі обсяги даних використовуються у форумах, поштових службах, заповненні бази даних, при пересиланні файлів і ін.

Тег <input> – є найважливішим у формах. Він може приймати величезну кількість значень, найпоширеніші з яких є:

<input type = "text"> або просто <input> – стандартне текстове поле. Також може мати атрибут value, що перетворює вихідний текст в textbox;

<input type = "password"> – схожий на textbox, проте символи приховані від користувача;

<input type = "checkbox"> – кнопка з прапорцем, користувач може задати режим вкл / викл;

<input type = "checkbox" checked> – робить прапорець «включеним»;

<input type = "radio"> – схожий на checkbox, користувач може вибрати тільки одну радіокнопку з групи. Також може мати атрибут checked;

<input type = "submit" value = "Click"> – кнопка, що відправляє форму. Користувач може змінювати вихідний текст форми через атрибут value.

Зверніть увагу на те, що тег input як і img, і br не має закриття тега.

Тег <textarea> – по суті, велика многострочное текстове поле. Через атрибути rows і cols задається число рядків і стовпців відповідно, хоча можна керувати розміром поля через CSS.

<textarea rows = "5" cols = "20">

Однією з особливостей HTML5 є можливість перевірки більшості даних користувача без використання скриптів. Це робиться за допомогою атрибутів перевірки елементів форми, які дозволяють вам вказувати правила введення форми, наприклад, чи потрібно заповнювати значення, мінімальна і максимальна довжина даних, чи повинно це бути число, адреса електронної пошти, адреса або щось ще, і шаблон, якому це має відповідати. Якщо введені дані відповідають всім цим правилам, дані вважаються валідними; якщо немає – невалидность.

Коли елемент валідний:

– елемент відповідає CSS псевдо-класу: valid; це дозволяє вам застосувати конкретний стиль до дійсним елементам;

– якщо користувач намагається надіслати дані, браузер відправить форму, якщо немає нічого, зупинить відправку (наприклад, JavaScript).

Якщо елемент невалідний:

– елемент відповідає CSS псевдо-класу: invalid; це дозволяє вам застосувати конкретний стиль до невалідним елементам;

– якщо користувач намагається надіслати дані, браузер заблокує форму і видасть повідомлення про помилку.

2. Послідовність виконання:

1.Ознайомитись з теоретичними даними.

2. Згідно з ТЗ та обраним шаблоном проекту розробити шаблон форми.

3. Програмувати файл index.html для виклику форми.

4. Програмувати файл форми form.html.

5. Додати стилі для форми у файл style.css.

6. Запрограмувати валідацію даних засобами HTML5.

7. Оформити звіт.

Приклад виконання

Створимо реєстраційну форму для введення даних користувача я додано на рисунку 4.

Рисунок 4 – Вигляд form.html

Лістинг 3 : Зміст файлу form.html

```
<html>
<head>
  <title>Form</title>
  <link href='style.css' rel='stylesheet' type='text/css'>
</head>
<body>
  <form method='post' class='reg-form'>
        <div class='form-row'>
              <label for='form_fname'>First Name: </label>
              <input type='text' id='form_fname' name='first_name'>
        </div>
        <div class='form-row'>
              <label for='form_sname'>Second Name: </label>
              <input type='text' id='form_sname' name='second_name'>
        </div>
        <div class='form-row'>
```

```
            <label for='form_email'>Email: </label>
            <input type='email' id='form_email' name='email'>
        </div>
        <div class='form-row'>
            <label for='form_phone'>Phone: </label>
            <input type='text' id='form_phone' name='phone'>
        </div>
        <div class='form-row'>
            <label for='form_about'>About myself: </label>
            <textarea id='form_about' name='about'></textarea>
        </div>
        <div class="form-row">
            <input type="submit" value='Go'>
        </div>
  </form>
</body>
</html>
```

кінець лістингу 3

Тепер задаємо властивості для зазначених класів і тегів. Почнемо з текстових полів, розміру блоку форми і рядків форми

Лістинг 4 : Зміст файлу style.css

```
.form-row {
  margin-bottom: 8px;
  overflow: hidden;
}
.form-row input {
  float: right;
  height: 19px;
  line-height: 19px;
  margin: 0px;
  padding: 0px 5px;
  width: 287px;
  border: 1px solid #ABADB3;
}
.reg-form {
```

```
  width: 416px;
  margin: 0 auto;
}
```

кінець лістингу 4

Тут важливо звернути увагу на CSS-властивість float: right; в form-row input і overflow: hidden; в form-row. Підключивши стильовий файл отримаємо форму з елементами які притискаються до правого її краю. Файл style.css вміщує стилі оформлення форми.

Лістинг 5 : Доопрацювання файлу style.css

```
.form-row textarea {
    height: 90px;
    line-height: 16px;
    font-size: 12px;
    padding: 0px 5px;
    width: 404px;
    border: 1px solid #ABADB3;
}
.form-row input [type=submit] {
  padding:5px;
  height:25px;
  width:100%;
  line-height: 10px;
  background: #337AB7;
  color: white;
}
.form-row input [type=submit]:hover {
  background:#286090;
}
* {
  font-family: arial;
  font-size:10pt;
}
```

кінець лістингу 5

Запрограмуємо форму у якій буде здійснюватися валідація даних. Найпростішою функцією перевірки даних в HTML5 є використання атрибуту required. Цей атрибут призначений для перевірки обов'язковості введення даних. Якщо цей атрибут встановлений, форма не буде відправлятися, а буде відображати повідомлення про помилку, а пусте поле input також буде вважатися невалідним.

Тепер розглянемо деякі функції HTML5, які можна використовувати для перевірки <input> елементів. Почнемо з простого прикладу - input, який дозволяє вам вибирати ваш улюблений плід між бананом і вишнею. Він включає простий текст <input> відповідний ярлик label і відправку submit тегом <button>.

Рисунок 5 – Вигляд form_valid.html

Лістинг 6 : Зміст файлу form_valid.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Favorite fruit start</title>
```

```
    <link href='style_valid.css' rel='stylesheet' type='text/css'>
    </head>
  <body>
     <form>
    <p>
    <fieldset>
     <legend>Title<abbr title="This field is mandatory">*</abbr></legend>
<input type="radio" required name="title" id="r1" value="Mr"><label for="r1">Mr.</label>
<input type="radio" required name="title" id="r2" value="Ms"><label for="r2">Ms.</label>
    </fieldset>
  </p>
  <p>
    <label for="n1">How old are you?</label>
    <!-- Атрибут шаблону pattern реалізують тип введення номеру  -->
    <input type="number" min="12" max="120" step="1" id="n1" name="age"
          pattern="\d+">
  </p>
  <p>
<label for="t1">What's your favorite fruit?<abbr title="This field is mandatory">*</abbr></label>
    <input type="text" id="t1" name="fruit" list="l1" required
      pattern="[Bb]anana|[Cc]herry|[Aa]pple|[Ss]trawberry|[Ll]emon|[Oo]range">
    <datalist id="l1">
      <option>Banana</option>
      <option>Cherry</option>
      <option>Apple</option>
      <option>Strawberry</option>
      <option>Lemon</option>
      <option>Orange</option>
    </datalist>
  </p>
  <p>
    <label for="t2">What's your e-mail?</label>
    <input type="email" id="t2" name="email">
  </p>
  <p>
    <label for="t3">Leave a short message</label>
```

```
    <textarea id="t3" name="msg" maxlength="140" rows="5"></textarea>
  </p>
  <p>
    <button>Submit</button>
  </p>
</form>
</body>
</html>
```

кінець лістингу 6

Елемент <fieldset> призначений для групування елементів форми. Таке групування полегшує роботу з формами, що містять велику кількість даних. Наприклад, один блок може бути призначений для введення текстової інформації, а інший – для прапорців.

Дуже поширеною функцією перевірки є атрибут pattern, який очікує Regular Expression в якості значення.

Всі текстові поля, створені з допомогти тегів <input> або <textarea> можуть бути обмежені за розміром, використовуючи атрибути minlength і maxlength. Поле невалидне якщо його значення коротше ніж minlength або значення довше значення maxlength. Браузери часто не дозволяють користувачеві вводити більш довге значення, ніж очікувалося, в текстові поля в будь-якому випадку, але корисно мати цей елемент управління.

Для числових полів, наприклад <input type = "number">, атрибути min і max також забезпечують обмеження валідації. Якщо значення поля менше атрибута min або більше атрибута max, поле буде невалідним.

Лістинг 7 : Зміст файлу style_valid.css

```
    body {
  font: 1em sans-serif;
  padding: 0;
  margin : 0;
 }
 form {
  max-width: 200px;
  margin: 0;
  padding: 0 5px;
 }
```

```
p > label {
  display: block;
}
input[type=text],
input[type=email],
input[type=number],
textarea,
fieldset {
  -webkit-appearance: none;
  width : 100%;
  border: 1px solid #333;
  margin: 0;
   font-family: inherit;
  font-size: 90%;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
input:invalid {
  box-shadow: 0 0 5px 1px red;
}
input:focus:invalid {
  outline: none;
}
input:invalid {
 border: 2px dashed red;
     }
input:valid {
border: 2px solid black;
     }
```

кінець лістингу 7

Зверніть увагу на CSS

```
input: invalid {
  border: 2px dashed red;
}
input: valid {
  border: 2px solid black;
}
```

В цьому випадку до input буде застосовуватися яскраво-червоний пунктирний border, коли він невалідний, і більш тонка чорна межа, коли він валідний.

ЛАБОРАТОРНА РОБОТА 4. Валідація форми із використанням перевірки обмежень засобами JAVASCRIPT

Мета: навчити студентів програмувати методи валідації даних при введенні до форми для Web-сайтів із використанням перевірки обмежень засобами JAVASCRIPT.

1. Теоретичні відомості

JavaScript – це мова програмування, який додає інтерактивності до вашого web-сайту (наприклад: реакція при натисненні кнопок або при введенні даних у форми).

API обмежень валідації дає потужний інструмент для перевірки форми, дозволяючи вам отримати контроль над призначеним для користувача інтерфейсом більше і краще того, що ви можете робити тільки за допомогою HTML і CSS.

Властивості API для повідомлень про помилки форм зазвичай включають в JavaScript це властивості об'єкта помилки типу name та message для локальної валідації за допомогою HTML/JS Constraint Validation API, що дозволяють перевіряти форми (наприклад, checkValidity()).

Для валідації на стороні клієнта (у браузері) за допомогою вбудованих можливостей HTML5 та методів JavaScript Constraint Validation API:

– validity – об'єкт елемента форми з властивостями на кшталт valid, typeMismatch, valueMissing, tooLong тощо, що описують стан валідації;

– validationMessage – властивість, що повертає відповідне повідомлення про помилку валідації;

– checkValidity() – метод для перевірки, чи проходить елемент або вся форма валідацію;

– reportValidity() – метод для відображення стандартного повідомлення про помилку валідації.

Для загальних помилок в JavaScript (наприклад, при викликах fetch або роботі з помилками в try...catch):

– name – назва типу помилки (наприклад, TypeError, ReferenceError);

– message – текстове повідомлення про помилку.

2. Послідовність виконання:

1.Ознайомитись з теоретичними даними.

2. Розробити або обрати шаблон форми.

3. Програмувати файл index.html

4. Додати стилі для форми у файл style.css.

5.Запрограмувати валідацію даних засобами API обмежень валідації JAVASCRIPT.

6. Оформити звіт.

Приклад виконання

Створимо реєстраційну форму для введення даних користувача із використанням перевірки обмежень засобами JAVASCRIPT. ЇЇ вигляд подано на рисунку 6.

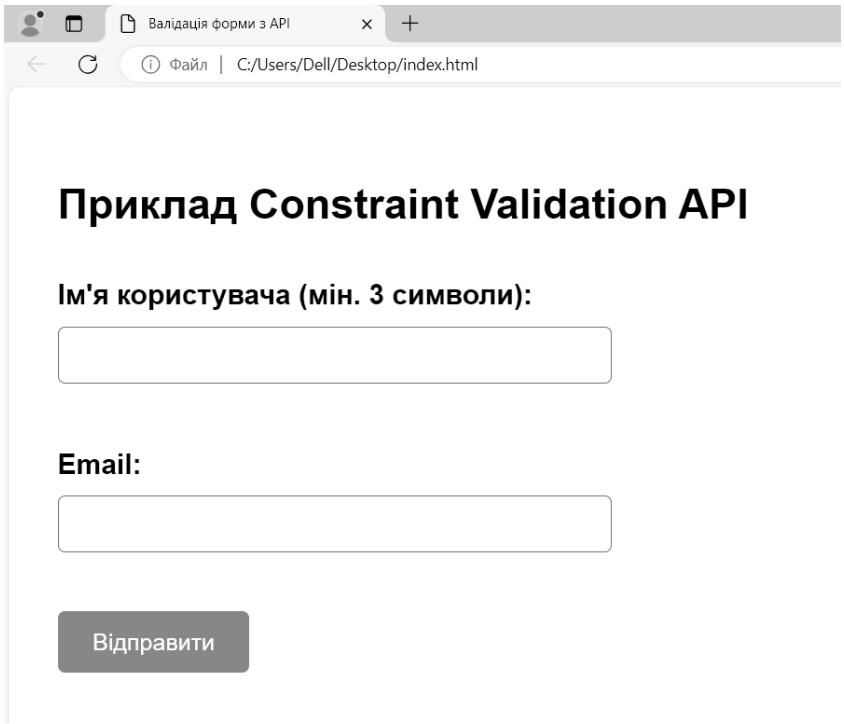

Рисунок 6 – Вигляд form_validAPI.html

Лістинг 8 : Зміст файлу form_validAPI.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Валідація форми з API </title>
    <style>
        body { font-family: sans-serif; padding: 20px; line-height: 1.6; }
        .form-group { margin-bottom: 15px; }
        label { display: block; margin-bottom: 5px; font-weight: bold; }
        input { padding: 8px; width: 300px; border: 1px solid #ccc; border-radius: 4px;
}
        /* Стилізація за допомогою псевдокласів валідації */
        input:invalid { border-color: #ff4d4d; }
        input:valid { border-color: #2ecc71; }
        .error-message {
            color: #ff4d4d;
            font-size: 0.85em;
            margin-top: 5px;
            min-height: 1em;
        }
        button { padding: 10px 20px; cursor: pointer; background: #3498db; color:
white; border: none; border-radius: 4px; }
    </style>
</head>
<body>
    <h2>Приклад Constraint Validation API</h2>
    <form id="registrationForm" novalidate>
        <!-- novalidate вимикає валідацію браузера -->
        <div class="form-group">
            <label for="username">Ім'я користувача (мін. 3 символи):</label>
            <input type="text" id="username" name="username" required
minlength="3">
            <div id="usernameError" class="error-message"></div>
        </div>
        <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
            <div id="emailError" class="error-message"></div>
        </div>
        <button type="submit">Відправити</button>
    </form>
    <script>
        const form = document.getElementById('registrationForm');
```

```
const usernameInput = document.getElementById('username');
const emailInput = document.getElementById('email');
// Функція для відображення помилок через властивості API
function updateValidationMessage(input, errorDisplayId) {
  const errorDisplay = document.getElementById(errorDisplayId);
  if (input.validity.valid) {
    errorDisplay.textContent = ''; // Очищуємо текст, якщо поле валідне
  } else {
// Використовуємо властивість validationMessage для отримання тексту помилки
// або перевіряємо конкретні стани об'єкта validity
    if (input.validity.valueMissing) {
      errorDisplay.textContent = 'Це поле є обов’язковим для заповнення.';
    } else if (input.validity.tooShort) {
      errorDisplay.textContent = `Занадто коротко. Має бути мінімум
${input.minLength} символи.`;
    } else if (input.validity.typeMismatch) {
 errorDisplay.textContent = 'Будь ласка, введіть коректну e-mail адресу.';
    } else {
      errorDisplay.textContent = input.validationMessage;
    }
  }
}
// Слухаємо подію введення для миттєвої валідації
usernameInput.addEventListener('input', () =>
updateValidationMessage(usernameInput, 'usernameError'));
emailInput.addEventListener('input', () =>
updateValidationMessage(emailInput, 'emailError'));
// Кастомна перевірка перед відправкою
form.addEventListener('submit', (event) => {
  // Перевірка через метод checkValidity()
  if (!form.checkValidity()) {
    event.preventDefault(); // Зупиняємо відправку
    updateValidationMessage(usernameInput, 'usernameError');
    updateValidationMessage(emailInput, 'emailError');
    alert('Форма містить помилки!');
  } else {
    alert('Форма успішно пройшла валідацію !');
  }
});
```

```
      // Приклад setCustomValidity: заборонимо слово "admin"
      usernameInput.addEventListener('input', () => {
        if (usernameInput.value.toLowerCase() === 'admin') {
          usernameInput.setCustomValidity('Ім’я "admin" заборонене.');
        } else {
// Важливо скинути кастомну помилку порожнім рядком
          usernameInput.setCustomValidity('');
        }
      });
  </script>
</body>
</html>
```

кінець лістингу 8

ЛАБОРАТОРНА РОБОТА 5. Валідація форми із використанням перевірки обмежень засобами JAVASCRIPT без вбудованого API

Мета: навчити студентів програмувати методи валідації даних при введенні до форми для Web-сайтів із використанням перевірки обмежень засобами JAVASCRIPT без вбудованого API.

1. Теоретичні відомості

Скасування стандартної поведінки: Атрибут novalidate у формі вимикає стандартні підказки браузера, дозволяючи JS повністю контролювати процес.

Регулярний вираз (Regex): Для e-mail використано шаблон, який перевіряє наявність символу @, крапки та домену.

Метод trim(): Обов'язковий при перевірці імені, щоб користувач не міг пройти валідацію, просто ввівши кілька пробілів.

Регулярний вираз (regex) - це шаблон (позначається в коді - /a|b/) який використовується для відповідності комбінацій символів у текстових рядках, тому він ідеально підходить для перевірки форми (а також для багатьох інших цілей в JavaScript).

Нижче наведені деякі приклади, щоб розкривають принцип застосування regex:

a - відповідає одному символу a (ні b, ні aa, і т.д.)

abc - відповідає a, далі b, далі c.

a * - відповідає символу a, 0 або більше разів (+ відповідає символу один або кілька разів).

[^ a] - відповідає одному символу, що не a.

a | b - відповідає одному символу a або b.

[abc] - відповідає одному символу a, b, або c.

[^ abc] - відповідає одному символу крім a, b, або c.

[a-z] - відповідає одному символу в діапазоні a-z, тільки в нижньому регістрі (ви можете використовувати [A-Za-z] для великих і малих літер, і [A-Z] тільки для великих літер).

a.c - відповідає a, за ним слід будь-який елемент, за ним слід c.

a {5} - відповідає a, 5 разів.

a {5,7} - відповідає a, від 5 до 7 разів, але не більше і не менше.

Ви також можете використовувати числа та інші символи в цих виразах, наприклад:

[-] - відповідає пробілу або тире.

[0-9] - відповідає будь цифрі в діапазоні від 0 до 9.

Ви можете комбінувати їх практично, як хочете, вказуючи різні частини, одну за одною:

[Ll]. * k - один символ L, в верхньому або нижньому регістрі, за ним йде ні одного або більше символів будь-якого типу за якими слідує k в нижньому регістрі.

[A-Z] [A-Za-z ' -] + - один символ верхнього регістру, за яким слід один або кілька символів, які представляють собою велику літеру або нижнього регістру, тире, пробіл або апостроф. Це можна використовувати для перевірки назви міст / міст англомовних країн, які повинні починатися з великої літери, але не містити інших символів. Приклади з UK включаючи Manchester, Ashton-under-lyne, і Bishop's Stortford.

[0-9] {3} [-] [0-9] {3} [-] [0-9] {4} - внутрішній телефонний номер США - три цифри, потім пробіл або тире, потім три цифри, потім пробіл або тире, потім чотири цифри.

2. Послідовність виконання:

1.Ознайомитись з теоретичними даними.

2. Розробити або обрати шаблон форми.

3. Програмувати файл index.html

4. Додати стилі для форми у файл style.css.

5. Запрограмувати валідацію даних із використанням перевірки обмежень засобами JAVASCRIPT без API.

6. Оформити звіт.

Приклад виконання

Створимо валідацію форми із використанням перевірки обмежень засобами JAVASCRIPT без вбудованого API. Фома подана на рисунку 7.

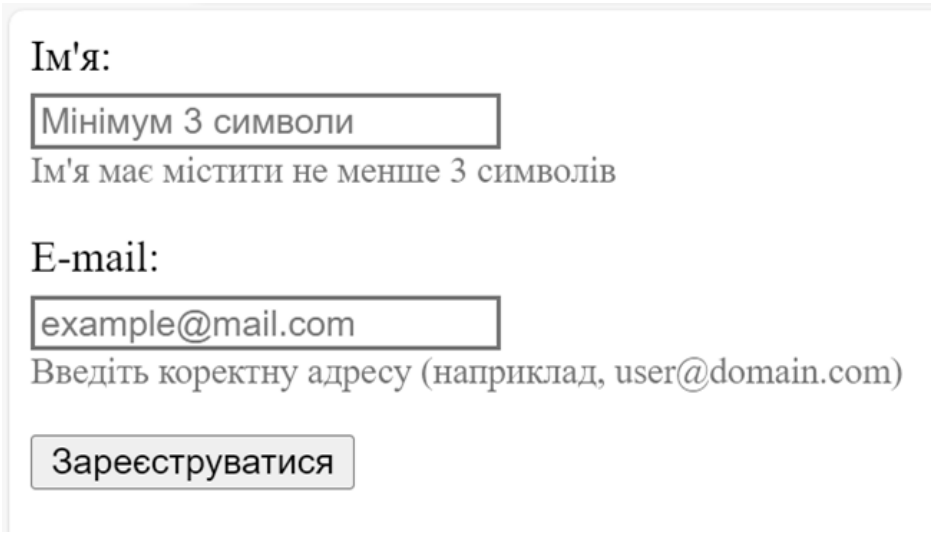

Рисунок 7 – Вигляд form_validNOAPI.html

Лістинг 9 : Зміст файлу form_validNOAPI.html

```
<!DOCTYPE html>
<html">
<head>
   <meta charset="UTF-8">
   <title>Валідація форми</title>
   <style>
      .form-group { margin-bottom: 15px; }
      .error { color: #ff4d4d; font-size: 0.85em; display: none; }
      input.invalid { border: 2px solid #ff4d4d; outline: none; }
      label { display: block; margin-bottom: 5px; }
   </style>
</head>
<body>
<form id="myContextForm" novalidate>
   <div class="form-group">
      <label for="name">Ім'я:</label>
      <input type="text" id="name" placeholder="Мінімум 3 символи">
```

```
<span id="nameError" class="error">Ім'я має містити не менше 3 символів</span>
    </div>
    <div class="form-group">
        <label for="email">E-mail:</label>
        <input type="email" id="email" placeholder="example@mail.com">
 <span id="emailError" class="error">Введіть коректну адресу (наприклад, user@domain.com)</span>
    </div>
    <button type="submit">Зареєструватися</button>
</form>
<script>
    const form = document.getElementById('myContextForm');
    const nameInput = document.getElementById('name');
    const emailInput = document.getElementById('email');
    // Регулярний вираз для перевірки e-mail (стандарт 2026)
    const emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
    const validateField = (input, condition, errorId) => {
        const errorElement = document.getElementById(errorId);
        if (condition) {
            input.classList.remove('invalid');
            errorElement.style.display = 'none';
            return true;
        } else {
            input.classList.add('invalid');
            errorElement.style.display = 'block';
            return false;
        }
    };
    form.addEventListener('submit', (e) => {
        e.preventDefault(); // Зупиняємо відправку для перевірки
        // Виконуємо перевірки
const isNameValid = validateField(nameInput, nameInput.value.trim().length >= 3,
'nameError');
        const isEmailValid = validateField(emailInput,
emailPattern.test(emailInput.value), 'emailError');

        if (isNameValid && isEmailValid) {
            alert('Форму успішно пройшла валідацію!');
            // Тут можна викликати form.submit()
```

```
        }
    });
// Динамічна перевірка під час введення (після першої спроби відправки)
    [nameInput, emailInput].forEach(input => {
        input.addEventListener('input', () => {
            if (input.id === 'name') {
                validateField(nameInput, nameInput.value.trim().length >= 3, 'nameError');
            } else {
                validateField(emailInput, emailPattern.test(emailInput.value), 'emailError');
            }
        });
    });
</script>
</body>
</html>
```

кінець лістингу 9

ЛАБОРАТОРНА РОБОТА 6. Патерни для пошуку в контенті Web-сайту засобами JAVASCRIPT

Мета: навчити студентів застосовувати патерни для пошуку в контенті Web-сайту засобами JAVASCRIPT.

1. Теоретичні відомості

Клас для роботи з рядками String має в своєму арсеналі досить великий набір властивостей і функцій, за допомогою яких ви можете здійснювати різні маніпуляції з рядками.

Довжина рядка

Властивість length дозволяє задавати довжину рядка. Дана властивість повертає число:

var hello1 = "привіт світ";

document.write ("У рядку '" + hello + "'" + hello1.length + "символів");

Пошук у рядку

Для того щоб знайти в рядку деякий підрядок застосовуються функції indexOf() (повертає індекс першого входження підрядка) та lastIndexOf()

(повертає індекс останнього входження підрядка). Ці функції приймають два аргументи:

1. Підрядок, який власне і треба знайти

2. Необов'язковий аргумент, який показує, з якого символу треба проводити пошук підрядка в рядку

Обидві ці функції повертають число, яке є індексом символу, з якого в рядку починається підрядок.

У разі якщо підрядок не знайдено, то повернеться число -1. Тому ці функції використовуються в логічних операторах, тому що як правило треба просто перевірити містить рядок підрядок чи ні, то в цьому випадку результат роботи цих функцій порівнюється з -1.

```
var str1 = "Привіт Вася!";
var podstr = "Петя";
if (str.indexOf (podstr) == -1) {
  document.write ( "Підрядок не знайдено.");
}
else {
  document.write ( "Підрядок знайдено.");
}
```

У прикладі буде виведено повідомлення «Підрядок не знайдено», оскільки рядок «Петя» не міститься в рядку «Привіт Вася!».

Вибір підрядка

Для вирізання з рядка підрядка, використовуються такі функції як substr() і substring().

Функція substring() приймає 2 аргументи:

1. стартова позиція символу в рядку, починаючи з якого буде проведена обрізка рядки

2. кінцева позиція до якої треба обрізати рядок

```
var hello1 = "Привіт світ. До побачення світ";
var world1 = hello1.substring (7, 11);     // з 7-го по 10-й індекс
document.write (world1);        // виведеться -  світ
```

Функція substr() також в якості 1-го параметра приймає стартовий індекс підрядка, а ось як 2-го - довжину підрядка:

```
var hello1 = "Привіт світ. До побачення світ";
var bye1 = hello1.substr (13, 2);
document.write (bye1);    //  виведеться -   До
```

Якщо 2-ий параметр не вказувати, то рядок буде обрізана до кінця:

```
var hello1 = "Привіт світ. До побачення світ";
var bye1 = hello1.substr (16);
```

```
document.write (bye1); //  виведеться -   побачення світ
```

Управління регістром літер

Для зміни регістру літер, тобто щоб зробити всі букви маленькими або великими використовуються функції toLowerCase() (для перекладу символів в нижній регістр, тобто всі букви будуть маленькими) і toUpperCase() (для перекладу символів у верхній регістр, тобто всі букви будуть великими).

```
var hello1 = "Привіт Джим";
document.write (hello1.toLowerCase () + "<br/>");  // привіт джим
document.write (hello1.toUpperCase () + "<br/>");  // ПРИВІТ ДЖИМ
```

Отримання символу за його індексом

Для того щоб знайти певний символ в рядку по його індексу, застосовуються функції charAt() і charCodeAt(). Обидві ці функції як аргумент приймають індекс символу:

```
var hello1 = "Привіт Джим";
document.write (hello1.charAt (3) + "<br/>");     // в
document.write (hello1.charCodeAt (3) + "<br/>");      // 1080
```

Але ось тільки якщо в якості результату своєї роботи функція charAt() поверне сам символ, то функція charCodeAt() поверне числовий Юнікод код цього символу.

Видалення пробілів

Для видалення пробілів у стоці використовується функція trim():

```
var hello1 = "  Привіт Джим  ";  //  по 2 пробіла з переду та позаду
var beforeLen = hello1.length;
hello1 = hello1.trim ();
var afterLen = hello1.length;
document.write ( "Довжина рядка до:" + beforeLen + "<br/>");   // 15
document.write ( "Довжина рядка після:" + afterLen + "<br/>");    // 10
```

Об'єднання рядків

Функція concat() дозволяє об'єднати 2 рядки:

```
var hello1 = "Привіт";
var world1 = "Вася";
hello1 = hello1.concat (world1);
document.write (hello);     //Привіт Вася
```

Заміна підрядка

Функція replace() дозволяє замінити один підрядок на інший:

```
var hello1 = "Добрий день";
hello1 = hello1.replace ( "день", "вечір");
document.write (hello1);                  //Добрий вечір
```

Перший аргумент функції вказує, який саме підрядок треба замінити, а 2-ий аргумент - на який власне підрядок потрібно замінити.

Поділ рядка на масив підрядків

Функція split() дозволяє розділити рядок на масив підрядків, використовуючи певний роздільник. Для поділу рядка його треба передавати в метод:

```
var mes = "Сьогодні була чудова погода";
var stringArr = mes.split ( "");
for (var str1 in stringArr)
   document.write (stringArr [str1] + "<br/>");
```

Використання регулярного виразу для пошуку

Патерни і прапори регулярних виразів

Якісний код - це перш за все швидкі програми і сайти без зайвого сміття. Зрештою, швидка отрисовка сторінок і поліпшене реагування принесуть вашим користувачам більш позитивний досвід взаємодії.

У мові програмування JavaScript регулярні вирази представлені об'єктом RegExp і вбудовані в методи рядків. За допомогою регулярних вирази можна знаходити і замінювати різні фрагменти тексту.

Регулярний вираз, як правило складається з шаблону і необов'язкових прапорів.

Розглянемо як створюються регулярні вирази:

```
var reg = new RegExp(/\d/, "gim");
```

Але є й інша форма запису (це шаблон всередині слешів «/»):

```
var reg = /[a-z]/; // без прапорів
var reg = /[a-z]/gmi; // с прапорами gmi
```

Слеші «/» говорять движку JavaScript про те, що це регулярний вираз. Це як лапки при створенні рядків.

Патерни і прапори регулярних виразів

Основа регулярного виразу - це патерн або шаблон. Це власне рядок, який можна розширити символами, для того щоб пошук був набагато могутніше.

Розглянемо найпростіший випадок, без прапорів і спеціальних символів:

```
var str1 = "Я люблю Регулярні вирази!"; // будемо шукати в цьому рядку
var reg = /лю/;
alert (str1.search (reg));    // знайдено збіг на позиції 2
```

Для порівняння ось звичайний пошук:

```
var str1 = "Я люблю регулярні вирази!";
var substr1 = "лю";
alert (str1.indexOf (substr1)); // знайдено збіг на позиції 2
```

Коди дещо подібні, однак потрібно зауважити, що для регулярного виразу був використаний метод search – який якраз працює з регулярними виразами, а для пошуку підрядка - indexOf.

Для простого пошуку застосування регулярного виразу не одразу дає помітний результат. Для прикладів, що мають складний код з регулярними виразами, можна побачити всю могутність застосування регулярного виразу.

Для початку розглянемо прапори.

У регулярних виразах можуть зустрічатися такі прапори, що впливають на пошук (в JavaScript їх всього три):

i - якщо цей прапорець встановлений, то регулярний вираз здійснює незалежний пошук від регістру символу, тобто немає відмінностей між великими і маленькими літерами A і a.

g - якщо цей прапорець встановлений, то регулярний вираз буде шукати всі збіги, якщо багато немає то тільки перший.

m - якщо цей прапорець встановлений, то пошук здійснюється в багаторядковому тексті.

Самий звичайно простий з цих прапорів - це i.

Приклад його використання:

```
var str1 = "Я люблю регулярні вирази!";  // шукаємо в цьому рядку
alert (str1.search (/ЛЮ/));
// повернув -1, що буде означати «не знайдено»
alert (str1.search (/ЛЮ/ i));  // знайдено збіг на позиції 2
```

З шаблном /ЛЮ/ виклик нам повернув -1, що буде означати що «не знайдено» (як і власне в indexOf),

З шаблоном /ЛЮ/ і виклик знайшов збіг на позиції 2, тому що встановлений прапор i, а це значить «лю» теж підходить.

Можна зробити висновок, що прапори і спец. символи можуть зробити код пошуку набагато могутніше.

Регулярні вирази складаються з паттерна і прапорів g, i та m.

Метод рядки str.search (regexp) повертає індекс, на якому знайдено збіг.

2. Послідовність виконання:

1.Ознайомитись з теоретичними даними.

2. Розробити шаблон сайта з елементом пошуку.

3. Виконати пошук у рядку із застосуванням функції indexOf().

4. Виконати пошук у рядку із застосуванням патернів і прапорів регулярних виразів.

5. Виконати пошук у рядку із застосуванням JQuery.

6. Оформити звіт.

Приклад виконання

Застосуємо патерн для пошуку в контенті web-сайту засобами JAVASCRIPT. Приклад реалізації патерну подано на рисунку 8.

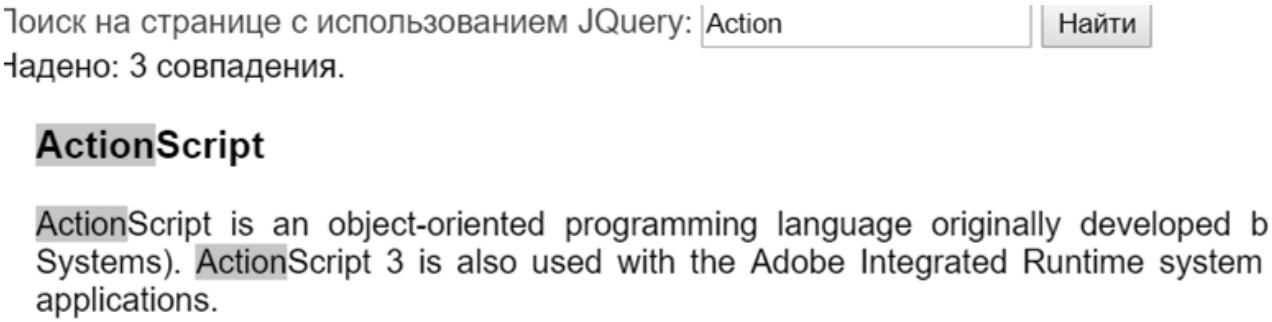

Рисунок 8 – Вигляд form_validNOAPI.html

Лістинг 10 : Зміст файлу form_validNOAPI.html

```
<body>
  <div id="body">
  <div id="findBody">
 <span style='color: red'>Поиск на странице с использованием JQuery:</span>
<input type="text" name="term" id="term" />
<input type="button" name="submit1" id="submit1" value="Найти" />
</div>
<div class="results"></div>
<div class="main">
<h3 id="actionscript">ActionScript</h3>
<p id="p1">ActionScript is an object-oriented programming language originally
developed by Macromedia Inc. (now owned by Adobe Systems).
ActionScript 3 is also used with the Adobe Integrated Runtime system for the
development of desktop and mobile applications.
 </p>
</div>
</div>
</body>
<style>
body {
font-family: "ApercuRegular",Helvetica,"Helvetica Neue",Arial,sans-serif;
margin:0px;
```

```
padding:0px;
}
.main {
margin:0px;
padding: 0px 20px;
text-align:justify;
}
.highlight {background:#4CFF00;}
</style>
<script type="text/javascript">
$(document).ready(function(){
        $("#submit1").click(function(){
                var term = "";
                var n = "0";
$('body').removeHighlight();       // метод видаляє виділення
$("div.results").hide().empty();    // empty() видаляє вміст з обраних елементів
 term = $('#term').attr('value');   //  attr встановлює значення атрибута jQuery
                if($('#term').val() == ""){
                        $("div.results").fadeIn().append("Введіть текст для пошуку");
  // Метод fadeIn() – плавна поява елемента.
  //  Метод .append() додає текст в кінець елемента
                        return false;
                }else{
                        $('.main').highlight( term );
                        n = $("span.highlight").length;
                if(n == 0){
                                $("div.results").fadeIn().append("Немає співпадінь!");
                        }else{
        $("div.results").fadeIn().append('Знайдено: '+n+' співпадінь.');
                        }
                        return false;
                }
        });
    }
);
</script>
```

кінець лістингу 10

ЛАБОРАТОРНА РОБОТА 7. Програмування патернів проектування Web-сайту засобами JAVASCRIPT

Мета: навчити студентів програмуванню патернів проектування Web-сайту засобами JAVASCRIPT.

1. Теоретичні відомості

Патерн проектування «поведінки» (behavior). Шаблон проектування «поведінку» дозволяє задавати обробники на елементах декларативно, установкою спеціальних HTML-атрибутів і класів.

Прийом проектування «поведінки» складається з двох частин:

1. Елементу ставиться атрибут, що описує його поведінку.

2. За допомогою делегування ставиться обробник на документ, який ловить все кліки і, якщо елемент має необхідний атрибут, виробляє потрібну дію.

приклад

Наприклад, додамо «поведінку» для всіх елементів у яких є атрибут data-counter, який при кліці збільшує значення на 1:

Лічильник:

```
<button data-counter>1</button>
```

Ще лічильник:

```
<button data-counter>2</button>

<script>
  document.onclick = function(event) {
    if (!event.target.hasAttribute('data-counter')) return;
    var counter = event.target;
    counter.innerHTML++;
  };
</script>
```

Якщо запустити HTML-код вище, то при кліці на кожну кнопку - її значення буде збільшуватися.

Патерн «Модуль» - це популярна реалізація патерну, що інкапсулює приватну інформацію, стан і структуру, використовуючи замикання.

Користуючись патерном «Модуль» можна управляти областю видимості змінних в JavaScript. Його мета - приховати внутрішні деталі реалізації скрипта. У тому числі: тимчасові змінні, константи, допоміжні міні-функції і т.п.

Для того щоб створити приватну зону видимості, можна скористатися замиканням. Як відомо, функції створюють власні області видимості, вміст яких відокремлено від глобальному контексті:

```
(Function () {
 // тут знаходиться приватна область видимості
}) ();
```

Перед нами - так звана самовикликаюча функція (IIFE, Immediately-Invoked Function Expression, негайно викликається функціональний вираз). Така функція виконується відразу ж після її оголошення. Подібні функції зручно використовувати для того, щоб вирішити якусь задачу, яку потрібно вирішити лише один раз, не залишаючи при цьому нічого зайвого в глобальному контексті. Усередині цієї функції (як, втім, і всередині інших функцій) створюється приватна область видимості, недоступна ззовні. Тобто, якщо оголосити всередині цієї області видимості іншу функцію, то, після того, як IIFE виконається, доступ до неї отримати не вдасться.

```
(function () {
  var myFunction = function () {
    // виконуємо тут якісь дії
  };
}) ();
```

Спробуємо тепер звернутися до функції myFunction з основного тексту програми:

```
myFunction ();     // Uncaught ReferenceError: myFunction is not defined
```

Як бачите, що цілком очікувано, цей виклик призводить до помилки. Це говорить нам про те, що дана функція в тій області видимості, з якої ми намагаємося до неї звернутися, недоступна. Розглянуті приклади потрібні нам лише для того, щоб підготуватися до розбору патерну «Модуль».

Модуль за допомогою замикань - це обгортання пакету функціоналу в єдину зовнішню функцію, яка тут же виконується. «модуль» - це всього лише функція-обгортка для приховування змінних.

Приватний змінні і функції, що зберігаються в замиканні. Наприклад, це можуть бути якісь допоміжні функції, що забезпечують роботу внутрішніх механізмів модуля.

Це можуть бути тимчасові змінні, або змінні, які відіграють роль сховищ деяких даних, доступ до яких ми хочемо жорстко контролювати. Нас цікавить такий пристрій модуля, коли зовнішнього світу є лише те, що повинно бути доступно, а все інше виявляється прихованим. Власне кажучи, приватним стане все те, що, в нашому прикладі, буде оголошено за межами об'єкта, що повертається з IIFE.

```
var Module = (function () {
  var privateMethod = function () {
  };
  return {
    publicMethod: function () {
    }
  };
}) ();
```

Метод publicMethod з цього прикладу можна викликати ззовні, а функцію privateMethod - ні, так як вона знаходиться в приватній області видимості, в замиканні. Саме подібні функції, недоступні ззовні, можуть виконувати роль допоміжних механізмів модулів. Вони можуть використовуватися для управління внутрішніми структурами даних, для виконання якихось викликів до якихось сервісів, і в інших ситуаціях.

При роботі з подібними функціями потрібно враховувати, що до них можна звертатися з інших функцій, оголошених в тій же області видимості, в тому числі - і з методів повернутого з IIFE об'єкта, причому, навіть після того, як виконана команда return, що повертає цей об'єкт . Тобто, загальнодоступні методи мають доступ до приватних функцій, вони можуть з ними взаємодіяти, але в глобальному контексті ці приватні функції недоступні.

```
var Module = (function () {
  var privateMethod = function () {
  };
  return {
    publicMethod: function () {
// у цього методу є доступ до privateMethod, ми можемо викликати його тут так:
      // privateMethod ();
    }
  };
}) ();
```

Завдяки цьому ми можемо захищати код від несанкціонованого втручання і захищати глобальну область видимості від забруднення. Якщо цього не робити, то, з одного боку, робота внутрішніх механізмів модулів може бути випадково або цілеспрямовано порушена, через те, що зовнішній код звертається до функцій або змінним, до яких він звертатися не повинен. З іншого боку, якщо не користуватися описаним тут підходом, в глобальну область видимості потрапляє багато непотрібного, що може, наприклад, привести до конфліктів імен.

Ось приклад об'єкта, що повертається з IIFE, який містить загальнодоступні методи і може звертатися до приватних функцій:

```
var Module = (function () {
  var myModule = {};
  var privateMethod = function () {
  };
  myModule.publicMethod = function () {
  };
  myModule.anotherPublicMethod = function () {
  };
  return myModule; // повертає об'єкт з загальнодоступними методами
}) ();
// використання модуля
Module.publicMethod ();
```

2. Послідовність виконання:

1.Ознайомитись з теоретичними даними.
2. Запрограмувати патерн проектування «поведінки» (behavior).
3. Запрограмувати патерн «Модуль».
4. Запрограмувати патерн «Модуль» з поверненнямоб'єкта з IIFE і API модул.
5. Оформити звіт.

Приклади виконання

Патерн проектування «поведінки» (behavior).

Елементів data-counter може бути скільки завгодно. Нові можуть додаватися в HTML в будь-який момент. За допомогою делегування ми, фактично, додали новий «псевдо-стандартний» атрибут в HTML, який додає елементу нову можливість («поведінку»).

Зробимо так, що при кліці на елемент з атрибутом data-toggle-id буде ховатися/показуватися елемент із заданим id:

```
<button data-toggle-id="subscribe-mail">
  Показати форму підписки
</button>
<form id="subscribe-mail" hidden>
  Ваша пошта: <input type="email">
</form>

<script>
  document.onclick = function(event) {
    var target = event.target;
    var id = target.getAttribute('data-toggle-id');
```

```
  if (!id) return;
  var elem = document.getElementById(id);
  elem.hidden = !elem.hidden;
 };
</script>
```

Тепер для того, щоб додати приховування-розкриття будь-якого елементу - навіть не треба знати JavaScript, можна просто написати атрибут data-toggle-id.

Це буває дуже зручно - не потрібно писати JavaScript-код для кожного елемента, який повинен служити такий кнопкою. Просто використовуємо поведінку.

Зверніть увагу: обробник поставлений на document, клік на будь-якому елементі сторінки пройде через нього, так що поведінка визначена глобально.

Для своїх цілей ми можемо використовувати в HTML будь-які атрибути, але стандарт рекомендує для своїх цілей називати атрибути data- *.

У обробнику document.onclick ми могли б перевіряти не атрибут, а клас або щось ще, але з атрибутом - простіше і зрозуміліше всього.

Також для додавання обробників на document рекомендується використовувати addEventListener, щоб можна було додати більше одного обробника для типу події.

Шаблон «поведінка» зручний тим, що як завгодно складне JavaScript-поведінку можна «навісити» на елемент одним лише атрибутом. А можна - декількома атрибутами на пов'язаних елементах.

Патерн «Модуль».

Розглянемо приклад коли до функції, що оголошена всередині іншої функції, можна звертатися.

```
// Оголосимо модуль
var Module = (function () {
 return {
  myMethod: function () {
   console.log ( 'myMethod has been called.');
  }
 };
}) ();
// Викличемо функцію як метод об'єкта
Module.myMethod ();
```

Можна помітити, що тут використовується таке ж IIFE, як раніше, але з функції тепер повертається об'єкт з методом, до якого можна звернутися з глобальному контексті. Сам по собі, природно, цей метод викликати не можна.

Треба відзначити, що в цьому прикладі можливостями замикання ми не користуємося.

Об'єкт, що повертається з IIFE - це звичайний об'єкт, у якого може бути безліч методів і властивостей.

```
// Оголосимо модуль
var Module = (function () {
 return {
  myMethod: function () {
  },
  someOtherMethod: function () {
  }
 };
}) ();
// Викличемо функцію як метод об'єкта
Module.myMethod ();
Module.someOtherMethod ();
```

Другий приклад «Модуля»за допомогою замикань.

Визначимо найпростіший модуль:

```
let foo = (function () {
   let obj = {greeting: "hello"};
      return {
      display: function () {
         console.log (obj.greeting);
      }
   }
}) ();
foo.display (); // hello
```

Тут визначена змінна foo, яка представляє результат анонімної функції. Всередині подібної функції визначено об'єкт obj з деякими даними.

Сама анонімна функція повертає об'єкт, який визначає функцію display. Повертається об'єкт визначає загальнодоступний API, через який ми можемо звертатися до даних, певним всередині модуля.

```
return {
      display: function () {
         console.log (obj.greeting);
      }
   }
```

Така конструкція дозволяє закрити (обгорнути) деякий набір даних в рамках функції-модуля і опосредовать доступ до них через певний API - повертаються внутрішні функції.

Складніший приклад:

```
// Оголосимо модуль
let  calcul = (function(){
   let   data = {number: 0};
     return {
       sum: function (n) {
         data.number += n;
              },
      subtract: function (n) {
                 data.number -= n;
      },
    display: function () {
          alert ( data.number);
      }
   }
}) ();
<p>
<input id="clickMe" type="button" value="10" onclick="calcul.sum (10);" />
<input id="clickMe" type="button" value="3" onclick="calcul.sum (3);" />
<input id="clickMe" type="button" value="=" onclick="calcul.display (); " /> </p>
<p>
Після натикування на кнопки  почерзі 10,  3 та = буде виведений результат 13.
<input id="clickMe" type="button" value="- 4" onclick="calcul.subtract (4);" />
<input id="clickMe" type="button" value="=" onclick="calcul.display (); " />
</p>
```

Після натикування на кнопки почерзі -4 та = буде виведений результат 9.

Даний модуль представляє примітивний калькулятор, який виконує три операції: додавання, віднімання і виведення результату.

Всі дані вміщені в об'єкті data, який зберігає результат операції. Всі операції представлені трьома функціями, що повертаються: sum, subtract і display. Через ці функції ми можемо управляти результатом калькулятора ззовні.

ЛАБОРАТОРНА РОБОТА 8. Docker. Контейнеризація web-додатка

Мета: навчитись встановлювати засіб контейнеризації Docker та запускати образи з Docker Hub.

1. Теоретичні відомості

Docker – це програмне забезпечення, яке призначене для автоматизації розгортання та управління додатками в середовищах з підтримкою контейнеризації. Docker дозволяє користувачеві «упакувати» додаток з усім його оточенням та залежностями у контейнер, який можна розгорнути на будь-якій Linux-системі з підтримкою контрольних груп у ядрі, а також надання набору команд для керування цими контейнерами.

Docker раціоналізує життєвий цикл розробки, що дозволяє розробникам працювати в стандартизованому середовищі через локальні контейнери, що надають додатки та сервіси. Контейнери відмінно підходять для робочих процесів безперервної інтеграції та безперервної доставки (continuous integration/continuous delivery, CI/CD).

Платформа, заснована на контейнерах, дозволяє легко портувати додатки. Контейнери можуть запускатися на локальній машині розробника, у фізичних або віртуальних дата-центрах, в публічних провайдерах або змішаних середовищах.

Docker використовує клієнт-серверну архітектуру (рисунок 9). Клієнт (Docker client) звертається до демона (Docker daemon), який приймає (збирає), запускає та розподіляє контейнери. Клієнт і демон можуть бути запущені в одній системі або клієнт може бути підключений до видаленого демона. Клієнт і демон спілкуються через REST APIповерхню UNIX-сокетовабо мережевий інтерфейс. Іншим клієнтом є Docker Compose, що дозволяє працювати з додатками, що складаються з кількох контейнерів.

Docker використовує безпосереднє керування контейнерами шляхом взаємодії з функціями ядра Linux. Це дозволяє Docker ефективно створювати, запускати та керувати контейнерами, не покладаючись на зовнішні середовища виконання контейнерів. Docker використовує кілька функцій ядра Linux для забезпечення своєї функціональності (дивись рис.9).

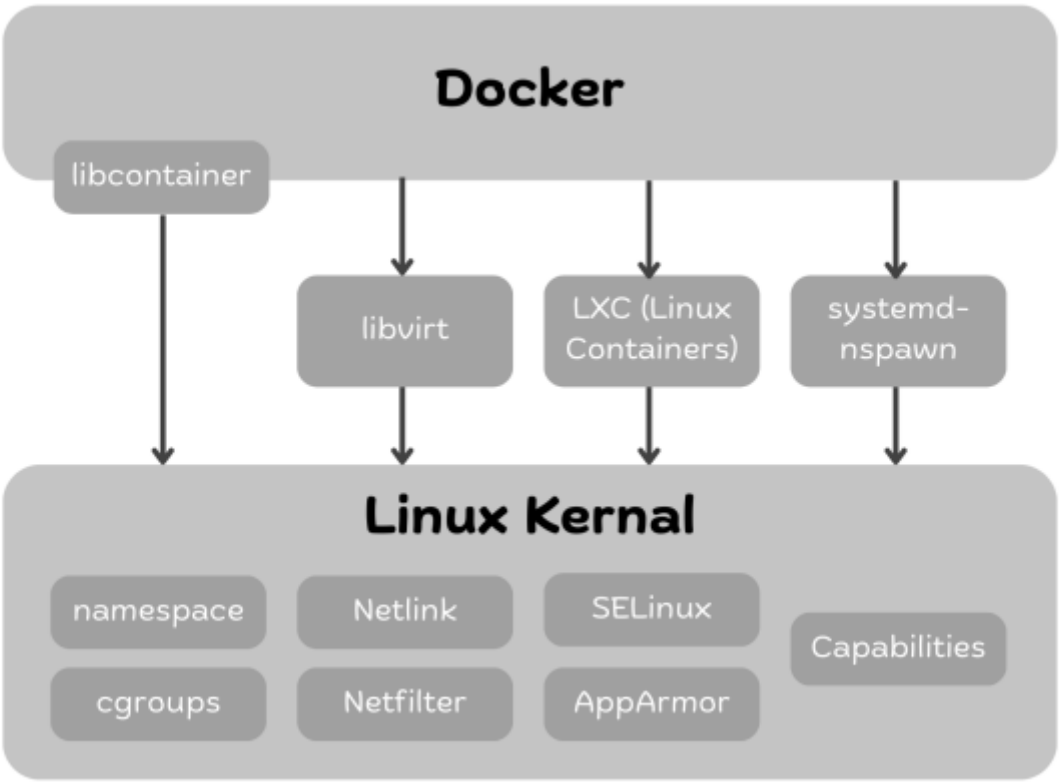

Рисунок 9 – Інтеграція Docker з Linux

Демон (dockerd daemon) реєструє (слухає) запити, які обробляють Docker API, і керує такими об’єктами, як образи, контейнери, мережі та томи. Демон може спілкуватися з іншими демонами для управління сервісами (дивись рисунок 10.

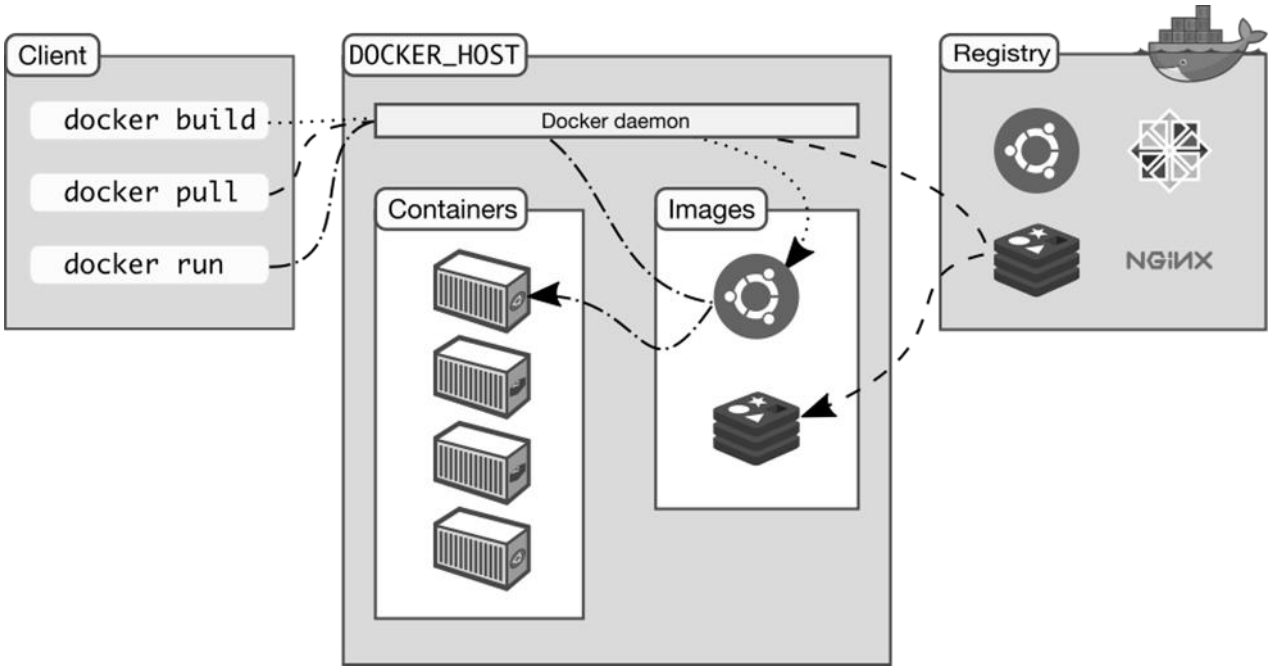

Рисунок 10 – Конфігурація Docker

Клієнт (Client docker) – основний спосіб комунікації з Docker. При виконанні такої команди, як docker run, клієнт відправляє цю команду демона, який, власне, цю команду і виконує.

Демон Docker знаходиться або встановлюється на локальному комп'ютері (hosts). Це мозок Docker. Він керує всіма образами та контейнерами. Вся його робота полягає в отриманні команд від хостів за допомогою CLI та виконанні дій на Docker за допомогою Docker API. Команда docker використовує Docker API. Клієнт може спілкуватися з кількома демонами.

Docker Desktop – це настивний додаток для Mac, Windowsi Linux, що дозволяє створювати та розподіляти контейнерні додатки та мікросервіси. Docker Desktop включає в себе демона, клієнта, Docker Compose, Docker Content Trust, Kubernetesi Credential Helper.

Docker Compose – це інструментальний засіб, який входить до складу Docker. Воно призначене для вирішення завдань, пов'язаних із розгортанням проектів. Простіше кажучи, цей засіб дозволяє за допомогою однієї команди запустити відразу безліч сервісів.

В реєстрі (реєстрі) хранятся образы контейнеров. Docker Hub – це публічний реєстр, який (за замовчуванням) використовується Docker для отримання образів. Кожен розробник має можливість створення приватних (закритих) реєстрів.

При виконанні таких команд, як docker pull або docker run, необхідні образи завантажуються з вбудованого реєстру. А при виконанні команди docker push образ завантажується в реєстр.

При використанні Docker ми створюємо і використовуємо образи, контейнери, мережі, томи, плагіни та інші об'єкти. Розглянемо деякі з них.

Образи (Images) – це доступні тільки для читання шаблони з інструкціями по створенню контейнера. Можна створювати свої образи або використовувати образи, що створені та опубліковані в реєстрі. Для створення образу використовується Dockerfile, що містить інструкції по створенню образу та його запуску. Ряд інструкцій Dockerfile призводить до створення в образі нового шару (раніше новий шар створювався для кожної інструкції). При зміні Dockerfile та повторному зборі образа перебираються тільки модифіковані шари. Це робить образи легкими, маленькими і швидкими.

Контейнери (Containers) – запускає екземпляр образу. Ми створюємо, запускаємо, переміщуємо та видаляємо контейнери за допомогою Docker API або CLI (інтерфейс командного рядка, інтерфейс командної строки).

За замовчуванням контейнери добре ізольовані від інших контейнерів і хосту. Однак ми можемо керувати тими та будь-якою підсистемою контейнера.

Команда docker run запускає контейнер ubuntu, інтерактивно підключається до локального сеансу командних рядків і виконується в цій команді /bin/bash:

docker run -i -t ubuntu /bin/bash

При виконанні цієї команди відбувається наступне:

– оскільки на машині розробника немає образу ubuntu, Docker завантажує його з реєстру;

– Docker створює новий контейнер (те же саме робить команда docker container create);

– у якості останнього шару Docker виділяється контейнер файлової системи для читання та запису. Це дозволяє створювати запущений контейнер і модифікувати файли та каталоги в його локальній файловій системі;

– оскільки не було вказано мережевих налаштувань, Docker створюється мережевий інтерфейс для підключення контейнера до дефолтної мережі. Це включає в себе присвоєний контейнер IP-адреса. Контейнери можуть підключатися до зовнішніх мереж через мережеве підключення хоста;

– Docker запускає контейнер і виконує /bin/bash. Якщо контейнер запущено в інтерактивному режимі та підключено до терміналу (завдяки прапорцям -ii -t) то можемо у подальшому вводити команди та отримувати результати в терміналі;

– виконання команди exit призводить до припинення виконання /bin/bash. Контейнер залишається, але не видаляється. Його можна запустити знову або видалити.

Основні команди для роботи з контейнерами (Containers):

- docker run [image] – створює та запускає контейнер з образу;
- -d – запуск у фоновому режимі (detached mode);
- -p 80:8080 – прокидання портів (зовнішній:внутрішній);
- --name [name] – призначення власного імені контейнеру;
- docker ps – список запущених контейнерів (додайте -a, щоб побачити всі, включаючи зупинені);
- docker stop [id/name] – зупинка контейнера;
- docker start [id/name] – запуск раніше зупиненого контейнера;
- docker rm [id/name] – видалення контейнера (додайте -f для примусового видалення);
- docker exec -it [id/name] bash – підключення до термінала всередині запущеного контейнера.

Основні команди для роботи з образами (Images)

– docker build -t [name:tag] – створення образу з Dockerfile у поточній директорії;
– docker images – список усіх завантажених образів на хості;
– docker pull [image] – завантаження образу з Docker Hub;
– docker push [image] – завантаження вашого образу в репозиторій;
– docker rmi [id/name] – видалення образу;

2. Послідовність виконання:

1.Ознайомитись з теоретичними даними.
2. Встановити Docker Desktop.
3. Запустити Docker Desktop.
4. Знайдіть образ MySQL з Docker Hub.
5. Запустити образ MySQL.
6. Оформити звіт.

Приклад виконання

Завантаження та встановлення Docker Desktop. Цю дію виконуємо після ознайомлення з офіційним сайтом з інструкціями пр встановленню Docker: https://docs.docker.com/desktop/setup/install/windows-install/#system-requirements.

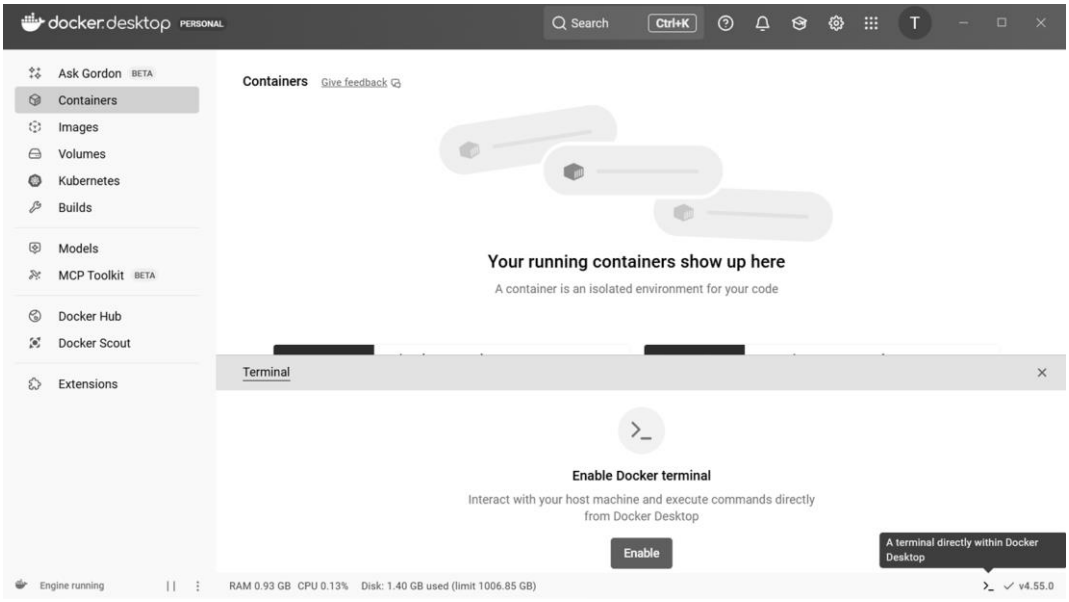

Рисунок 11 – Вікно Docker Desktop після встановлення

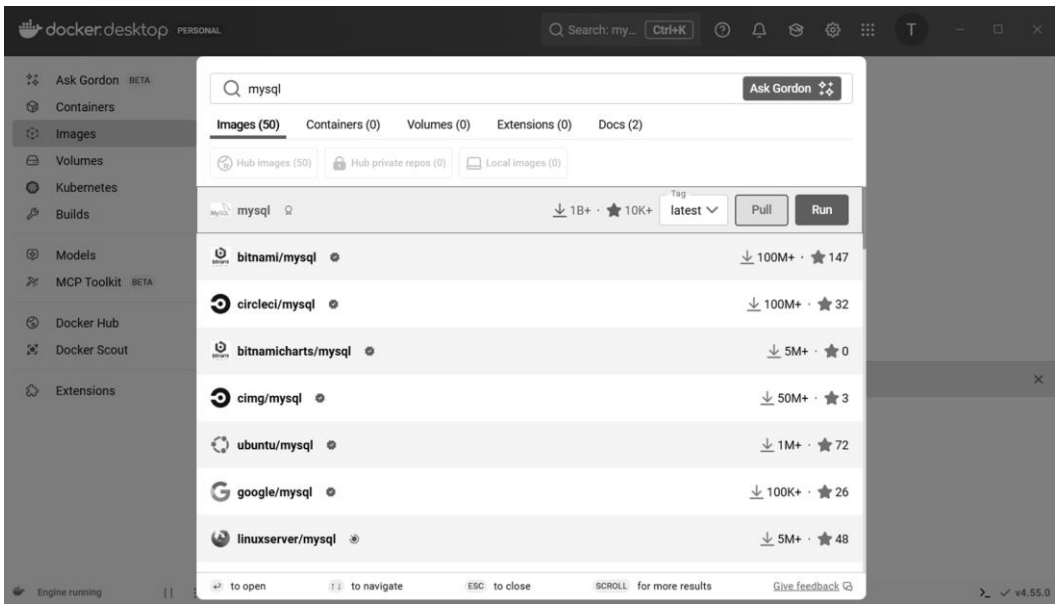

Рисунок 12 – Конфігурація Docker

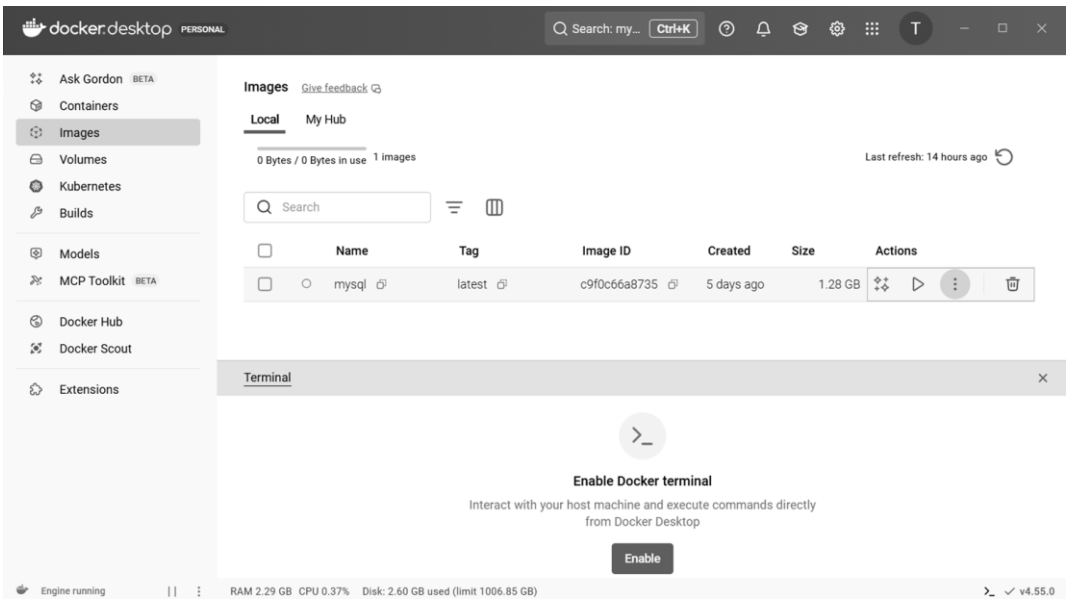

Рисунок 13 – Конфігурація Docker

Щоб відобразити MySQL з Docker Hub у браузері, вам потрібно запустити контейнер з MySQL, прокинути порт із контейнера на ваш хост і використовувати web-інтерфейс phpMyAdmin) для підключення до MySQL по цьому порту в браузері або локально.

Docker Hub – це просто репозиторій образів, а чи не web-інтерфейс для бази даних.

Кроки для запуску та доступу до MySQL

Запустіть контейнер MySQL:

docker run --name my-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -p 3306:3306 -d mysql:latest

--name my-mysql: Ім'я контейнера.

-e MYSQL_ROOT_PASSWORD=my-secret-pw: Встановлює логин root и пароль my-secret-pw

-p 3306:3306: Прокидає порт 3306 з контейнера на ваш локальний хост (перший 3306 – хост, другий – контейнер)

-d: Запуск у фоновому режимі.

mysql:latest: Використовує останній образ MySQL

Встановіть та запустіть web-інтерфейс (наприклад, phpMyAdmin):

docker run --name phpmyadmin -d -p 8080:80 --link my-mysql:db -e PMA_HOST=db phpmyadmin/phpmyadmin

-p 8080:80: Прокидає порт 8080 на ваш хост

--link my-mysql:db: Зв'язує phpMyAdmin з контейнером MySQL

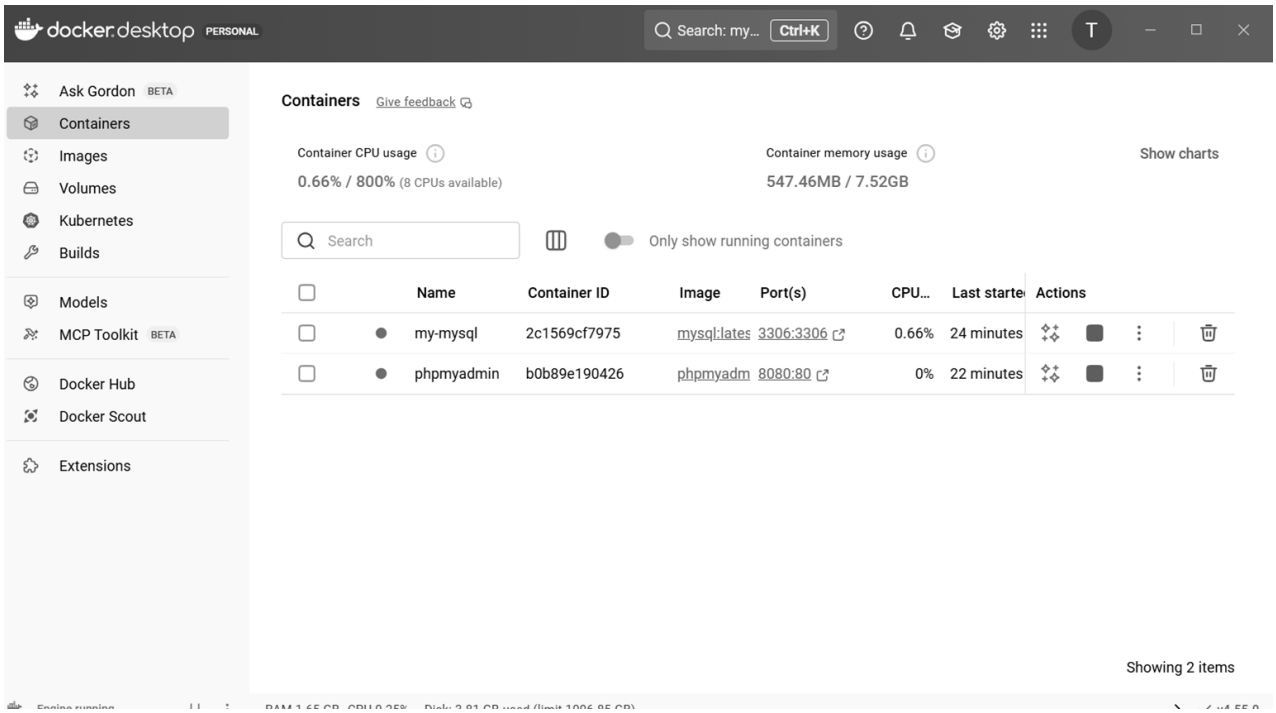

Рисунок 14 – Конфігурація Docker

Відкриваємо у браузері.

Для phpMyAdmin потрібно відкрити http://localhost:8080 у Chrome.

Рисунок 15 – Авторизація phpMyAdmin

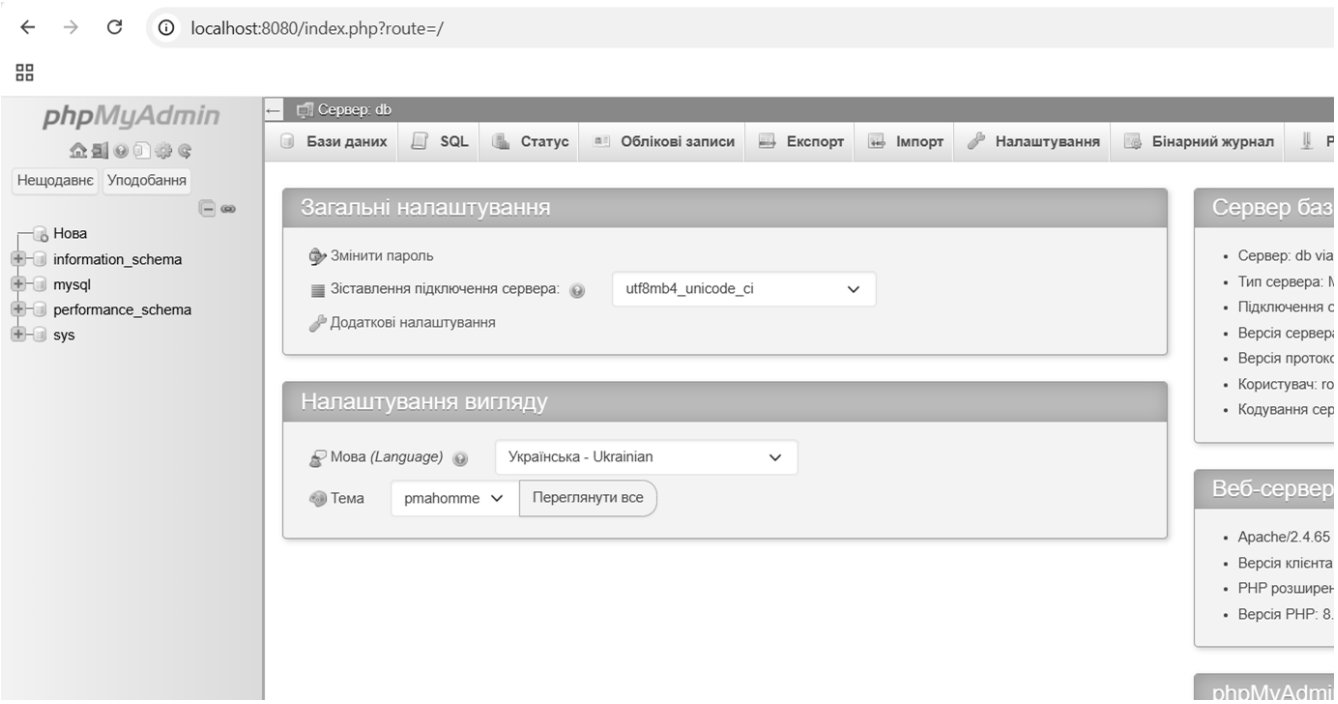

Рисунок 16 – Редактор phpMyAdmin

ЛАБОРАТОРНА РОБОТА 9. Docker Compose. Контейнеризація та запуск власного проекту

Мета: навчитись виконувати контейнеризацію та запуск власного проекту з використанням Docker Compose.

1. Теоретичні відомості

Docker Compose – це інструментальний засіб, який входить до складу Docker. Його головна роль полягає в автоматизації розгортання цілого стека сервісів за допомогою одного файлу конфігурації. Docker Compose відіграє роль інструменту для визначення, конфігурації та запуску багатоконтейнерних Docker-додатків, спрощуючи управління всією інфраструктурою проєкту через один YAML-файл (docker-compose.yml), що дозволяє координувати роботу сервісів, мереж і томів та запускати їх однією командою.

Docker Compose – це інструмент, який спрощує одночасний запуск кількох контейнерів. Він дозволяє визначити всі контейнери, мережі та томи для вашої програми в одному файлі.

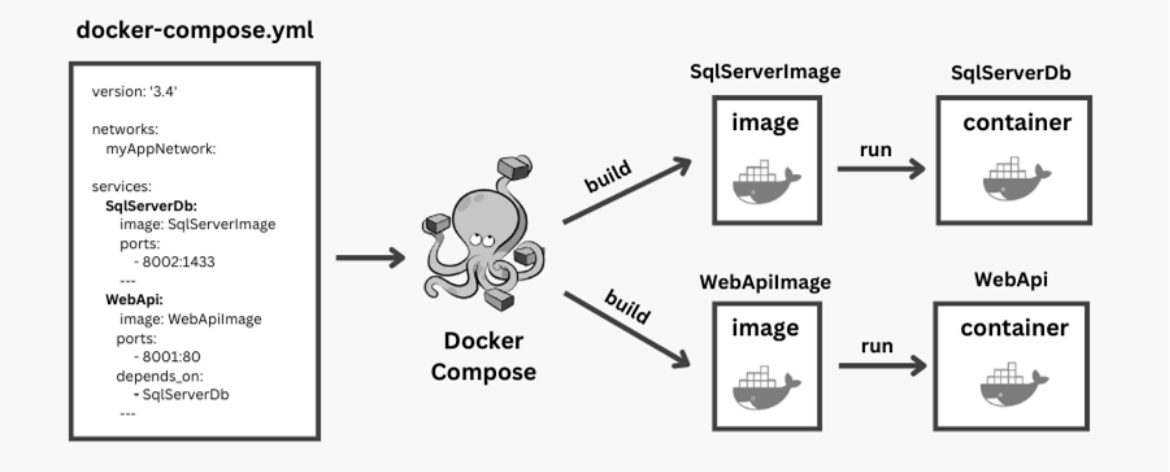

Рисунок 17 – Конфігурація Docker Compose

Основні функції та роль Docker Compose:

– оркестрація на одному ноді. Це дозволяє описати всі компоненти програми (бази даних, бекенд, фронтенд тощо) у єдиному файлі docker-compose.yml;

– забезпечує спрощене керування. Тобто, замість того, щоб запускати кожен контейнер окремо з довгими командами docker run, ви використовуєте одну команду – docker-compose up – для запуску всієї системи;

– ізоляція середовищ. Compose створює окрему віртуальну мережу для кожного проєкту, що дозволяє контейнерам одного стека спілкуватися між собою, залишаючись ізольованими від інших;

– робота з томами (Volumes) та мережами. Compose автоматично створює та підключає необхідні сховища даних і мережеві конфігурації, описані в YAML-файлі;

– забезпечує ефективність локальної розробки та тестування, оскільки дозволяє швидко відтворити ідентичне середовище на будь-якій машині.

Основні команди для роботи з Docker Compose (керування стеком):

- docker compose up – запуск усіх сервісів, описаних у docker-compose.yml;
- docker compose down – зупинка та видалення контейнерів, мереж та образів проєкту;
- docker compose logs -f – перегляд логів усіх сервісів у реальному часі.

Для контейнеризації маємо проект з HTML CSS

project/
├── docker-compose.yml
└── .htaccess
└── index.html
└── styles.css

Де

.htaccess – це файл конфігурації Apache, який дозволяє керувати поведінкою сайту без доступу до головного конфігу сервера.

docker-compose.yml – це YAML-файл, у якому описується:

які контейнери потрібні

з яких образів

які порти відкривати

які файли/папки підключати

як вони взаємодіють між собою

docker-compose.yml потрібен для зручного керування кількома Docker-контейнерами як одним застосунком.

Docker Compose читає цей файл і автоматично піднімає все середовище.

Тобто замість довгих команд docker run використовується коротка форма

Наприклад, без Compose:

```
docker run -d -p 80:80 -v ./html:/var/www/html apache
docker run -d -p 3306:3306 mysql
```

Тоді як з Compose:

```
docker compose up -d
```

Розберемо шаблон коду docker-compose.yml

```
services:
  web:
    image: php:8.1-apache
    ports:
      - "8080:80"
    volumes:
      - ./html:/var/www/html
```

У коді:
web – назва сервісу
php:8.1-apache – образ
8080:80 – порт
./html – папка з index.html

2. Послідовність виконання:

1.Знайти шаблон сайту використавши сайт шаблонів.

2. Скачати архів шаблону.

3. Ознайомитись із шаблоном сайту з каскадним меню (видається викладачем).

4. Розібратися з вмістом файлів index.html та style.css використовуючи довідкові ресурси з Internet.

5. Програмувати зміни у скачаному шаблоні, а саме каскадне меню web-сторінки за зразком виданим викладачем
– зробити каскадне меню;
– доопрацювати CSS.

6. Оформити звіт.

Приклад виконання

Відкриваємо проект HTML CSS у Visual Studio Code

```
project/
├── docker-compose.yml
└── .htaccess
└── index.html
└── styles.css
```

Лістинг 11 : Зміст файлу docker-compose.yml

```
services:
  web:
    image: httpd:2.4
    container_name: apache_html
    ports:
      - "8080:80"
    volumes:
      - ./:/usr/local/apache2/htdocs
    restart: unless-stopped
```

кінець лістингу 11

Лістинг 12 : Зміст файлу .htaccess

```
DirectoryIndex index.html
```

кінець лістингу 12

Створення образу
docker compose up -d -- build
Запуск docker
docker compose up -d
Запуск проекту у браузері
http://localhost:8080

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Баран С.В. Основи web-програмування: навч. посіб. Кривий Ріг: Державний університет економіки і технологій, 2023. 316 с.
2. Двірничук К.В., Вацек Д.О. Веб-програмування та веб-дизайн : навч. посіб. Чернівці : Чернівец. нац. ун-т ім. Ю. Федьковича, 2022. 472 с.
3. Влнссидес, Джон. Застосування шаблонів проектування. Додаткові штрихи. URL: http://khizha.dp.ua/library/John_Vlissides-_Pattern%20Hatching_ Design_Patterns_ Applied_ru.pdf (дата звернення: 19.12.2025).
4. Технології розробки програмного забезпечення. Методології та засоби розробки: комп'ютерний практикум. URL: https://ela.kpi.ua/bitstream/123456789/43482/1/MRPZ.pdf (дата звернення: 19.12.2025).
5. Керівництво по Docker. Введення, Docker CLI. URL: https://my-js.org/docs/guide/docker (дата звернення: 19.12.2025).
6. Тулашвілі Ю.Й. Комплекти web-порталів та мобільних додатків для цифрової трансформації. Матеріали XIV міжнародної науково-практичної конференції «Інформаційні технології і автоматизація – 2021». Одеса, 21-22 жовтня 2021 р. - Одеса, Видавництво ОНАХТ, 2021 р. – С.273-276.
7. Веб програмування. Частина 1. Конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерні науки» галузі знань 12 Інформаційні технології спеціальності 122 Комп'ютерні науки денної та заочної форм навчання . Уклад. Ю.Й. Тулашвілі. Луцьк : Луцький НТУ, 2020. 71 с.

Web-програмування (Cloud Computing). Частина 1: методичні вказівки до виконання лабораторних занять для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерні науки» галузі знань F Інформаційні технології спеціальності F3 Комп'ютерні науки денної та заочної форм навчання / уклад. Ю.Й. Тулашвілі, Луцьк: ЛНТУ, 2026. 60 с.

Комп'ютерний набір Ю.Й. Тулашвілі
Редактор Ю.Й. Тулашвілі

Підп. до друку «__» _____ 2026 р. Формат 60x84/57. Папір офс.
Гарн. Таймс. Ум. друк. арк. 6,3.
Тираж 50 прим.

Відділ іміджу та промоції
Луцького національного технічного університету
43018 м. Луцьк, вул. Львівська, 75
Друк – ВІП Луцького НТУ