

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

"INTERNET ТЕХНОЛОГІЇ"

Частина 2

для студентів денної та заочної форм навчання
напряму 6.050102 - «Комп'ютерна інженерія»

Електронне видання

ЗАТВЕРДЖЕНО
Кафедрою ЕОМ
Протокол № 1
від “30” серпня 2013р.

ХАРКІВ 2014

Методичні вказівки до лабораторних робіт з дисципліни «Internet технології» (частина 2) для студентів денної та заочної форм навчання напрямку 6.050102 - «Комп'ютерна інженерія» [Електронне видання] / Упоряд: Лебьодкіна А.Ю., Саранча С.М. - Харків: ХНУРЕ, 2014. - 109 с.

Упорядники: А.Ю. Лебьодкіна
 С.М. Саранча

Рецензент М.М. Корабльов, д. т. н., професор кафедри ЕОМ

ВМІСТ

ВСТУП.....	5
1 ПРОГРАМНІ ЗАСОБИ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ.....	6
ВХІДНИХ ДАНИХ	6
1.1 Мета роботи	6
1.2 Методичні вказівки з організації самостійної роботи студентів.....	6
1.3 Опис лабораторної установки.....	6
1.4 Порядок виконання роботи і методичні вказівки з її виконання	6
1.4.1 Валідація даних	7
1.4.2 Методи шифрування даних.....	20
1.5 Варіанти завдань.....	25
1.6 Зміст звіту.....	27
1.7 Контрольні запитання і завдання.....	28
2 ВИВЧЕННЯ РОЗШИРЕННЯ PDO ДЛЯ ЗАБЕЗПЕЧЕННЯ АБСТРАКЦІЇ ДОСТУПУ ДО БАЗ ДАНИХ.....	29
2.1 Мета роботи	29
2.2 Методичні вказівки з організації самостійної роботи студентів.....	29
2.3 Опис лабораторної установки.....	29
2.4 Порядок виконання роботи і методичні вказівки з її виконання	29
2.5 Варіанти завдань.....	45
2.6 Зміст звіту.....	52
2.7 Контрольні запитання і завдання.....	53
3 ПРОГРАМНІ МЕХАНІЗМИ ШАБЛОНІЗАЦІЇ WEB-ДОДАТКІВ	54
3 ВИКОРИСТАННЯМ TWIG	54
3.1 Мета роботи	54
3.2 Методичні вказівки з організації самостійної роботи студентів.....	54
3.3 Опис лабораторної установки.....	54
3.4 Порядок виконання роботи і методичні вказівки з її виконання	54
3.5 Приклад виконання лабораторної роботи	66
3.6 Варіанти завдань.....	70
3.7 Зміст звіту.....	70
3.8 Контрольні запитання і завдання.....	70
4 АСИНХРОННИЙ ОБМІН ДАНИМИ З СЕРВЕРОМ НА ОСНОВІ ТЕХНОЛОГІЇ AJAX.....	72
4.1 Мета роботи	72
4.2 Методичні вказівки з організації самостійної роботи студентів.....	72
4.3 Опис лабораторної установки.....	72
4.4 Порядок виконання роботи і методичні вказівки з її виконання	72
4.5 Приклад використання технології AJAX.....	76
4.6 Варіанти завдань.....	82
4.7 Зміст звіту.....	82

4.8 Контрольні запитання і завдання.....	82
5 ПОВНОДУПЛЕКСНИЙ ОБМІН ДАНИМИ МІЖ БРАУЗЕРОМ ТА ВЕБ-СЕРВЕРОМ НА ОСНОВІ ТЕХНОЛОГІЇ WEBSOCKET	83
5.1 Мета роботи	83
5.2 Методичні вказівки з організації самостійної роботи студентів.....	83
5.3 Опис лабораторної установки.....	83
5.4 Порядок виконання роботи і методичні вказівки з її виконання	83
5.5 Варіанти завдань.....	92
5.6 Зміст звіту.....	94
5.7 Контрольні запитання і завдання.....	94
ПЕРЕЛІК ПОСИЛАНЬ	95

ВСТУП

Другий семестр дисципліни «Internet технології» присвячений завданням реалізації багаторівневих асинхронних інтернет-додатків з насиченим, функціональним інтерфейсом, а також підтримкою безпеки переданих даних і незалежним доступом до них.

Лабораторні роботи з дисципліни «Internet технології» призначено для закріплення студентами знань, одержаних на лекційних і практичних заняттях, а також набуття експериментальних навичок перевірки коректності даних і забезпечення безпеки web додатків, застосування розширення об'єктів даних PDO для забезпечення абстракції доступу до баз даних, освоєння механізмів шаблонізації Twig, а також вивчення методів передачі даних із застосуванням технології AJAX і протоколу передачі даних WebSockets.

До виконання лабораторних робіт допускаються студенти, що пройшли інструктаж з техніки безпеки і успішно пройшли контрольне опитування. Звіт з лабораторної роботи складається кожним студентом окремо. Захист виконаної роботи відбувається під час наступного заняття. Під час перебування в лабораторії студенти повинні суворо дотримуватися вимог техніки безпеки щодо роботи з комп'ютерною технікою. Інструктаж з техніки безпеки проводить викладач на початку лабораторних зайнять, про що кожен студент і викладач засвідчують в лабораторному журналі.

Лабораторні роботи виконуються в комп'ютерних класах з використанням такого програмного забезпечення побудови багаторівневих інтернет-додатків як середовище для розробки додатків Notepad++ 6.4.5, веб-сервер хампрр 2.4.4, веб-інтерфейс адміністрування баз даних phpMyAdmin, бібліотека phpseclib, шаблонізатор Twig, серверна платформа NODE.js, пакет socket.io.

Вимоги до комп'ютера: IBM з процесором Intel Core 2 Quad Q8200, 2,33 ГГц, не менше 20 MB вільного простору на жорсткому диску, операційна система Microsoft Windows XP або пізніші версії, маніпулятор типу мишка, кольоровий монітор стандарту VGA і вище.

На лабораторних роботах студент повинен показати грамотне використання програмних засобів розробки програм та навички кваліфікованого програмування. При реалізації програмного додатка особливу увагу необхідно приділити застосуванню коректних алгоритмічних рішень і шаблонів проектування.

У кінці кожної роботи наведені контрольні запитання, відповіді на які дозволяють визначити ступінь готовності студентів до виконання лабораторної роботи.

1 ПРОГРАМНІ ЗАСОБИ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ВХІДНИХ ДАНИХ

1.1 Мета роботи

Вивчення клієнтських і серверних програмних засобів безпеки даних, що вводяться користувачем, з метою побудови надійних і стійких інтернет-додатків.

1.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до виконання лабораторної роботи слід повторити засоби створення форми з використанням стандарту HTML5 та CSS3 [1], ознайомитися з синтаксисом регулярних виразів, загальними принципами валідації даних на клієнтському і серверному боках, а також роботи з DOM API HTML5 для валідації даних [2], вивчити програмні методи хешування і шифрування даних [3,4].

1.3 Опис лабораторної установки

При виконанні лабораторної роботи використовується ПЕОМ під керуванням операційної системи Windows XP і старше, веб-сервер хатрр 2.4.4, бібліотека phpseclib, середовище розробки Notepad++ 6.4.5.

1.4 Порядок виконання роботи і методичні вказівки з її виконання

Поряд із загальними питаннями безпеки, уразливості в Web додатках виникають через недостатню перевірку введених даних і застосування ненадійних методів хешування, що реалізуються на рівні додатків.

Метою реалізації надійних і стійких web-додатків розробнику рекомендується виконати ряд перевірок:

- визначення ступеня довіри до даних;
- типізація всіх не довірених даних;
- валідація всіх типізованих недовірених даних;
- фільтрація, екранування та санітаризація даних;
- хешування та/або шифрування даних.

1.4.1 Валідація даних

Регулярні вирази дають вам можливість побудувати шаблони, використовуючи набір спеціальних символів. Ці шаблони потім можуть зіставлятися з текстом файлу, введеної в додаток датою або даними з форми, заповненої користувачем на сайті. Залежно від того, чи є збіг, чи ні, приймається відповідна дія і викликається відповідний програмний код.

Наприклад, одне з найпоширеніших додатків регулярних виразів - перевірка того, вірний чи ні введений у форму користувачем адрес електронної пошти; якщо так, форма буде прийнята, а якщо ні, з'явиться повідомлення, що просить користувача виправити помилку. Тут регулярні вирази відіграють велику роль у процедурах інтернет-додатків, які приймають рішення, хоча, як ви побачите пізніше, вони також можуть мати величезний ефект у складних операціях пошуку та заміни.

Синтаксис регулярних виразів.

Найпростіший регулярний вираз можна записати так: **"abc"**. Цей вираз відповідає будь-якому рядку, яка містить підрядок "abc".

Існує таке поняття, як вираз у квадратних дужках. Квадратні дужки обмежують пошук тими символами, які в них укладені: **"[abc]"**.

Цьому регулярному виразу відповідає будь-який рядок, що містить abc або разом, або кожен з них окремо.

Припустимо, потрібно створити регулярний вираз, відповідний всім буквам російського алфавіту. У цьому випадку можна, звичайно, перелічити всі ці букви в регулярному виразі. Це допустимо, але утомливо і неелегантно. Більш коротко такий регулярний вираз можна записати наступним чином: **"[a-Я]"**.

Цей вираз відповідає всім буквам російського алфавіту, оскільки будь-які два символи, що розділяються дефісом, задають відповідність діапазону символів, що знаходяться між ними. Зауважте, що регулярний вираз **"[a-Я]"** описує символи як нижнього, так і верхнього реєстрів, тому більш детально цей вираз можна записати так: **"[a-я A-Я]"**.

Точно таким же чином задаються регулярні вирази, що відповідні числах: **"[0-9]"** або **"[0123456789]"**.

Обидва ці вирази еквівалентні та відповідають будь-якій цифрі.

При створенні регулярних виразів часто зручно користуватися груповим символом крапки **"."**, який об'єднує два одиночних символи, за винятком символу **\ n**. Наприклад, **".ok"**. Цей вираз, зокрема відповідає рядкам "кок", "док", "ток".

Вираз **"x.[0-9]"** відповідає рядку, що містить символ x, за яким йде будь-який інший символ і цифри від 0 до 9. Цьому критерію, наприклад, задовольняють рядки "ху1", "хз2" і т. д.

У регулярному виразі може бути кілька гілок, які розділяються символом |, чинним як оператор OR (АБО). Тобто, якщо у виразі використовуються гілки, то для відповідності регулярного виразу якому-небудь рядку, достатньо, щоб тільки одна з гілок відповідала цьому рядку: **"abc|abv"**.

Цьому регулярному виразу відповідає будь-який рядок, що містить підрядки "abc" або "abv". Галуження зручно застосовувати при перевірці розширень та імен файлів, зон доменних імен і т. д. Наприклад, наступний регулярний вираз перевіряє, чи містяться у рядку підрядки "ru", "com" або "net": **"ru | com | net"**.

Для виключення послідовності символів з пошуку перед нею ставиться символ "^": **"[^a-я]"**.

Цей регулярний вираз відповідає будь-якому символу, що не міститься в діапазоні а-я. Зверніть увагу, що символ ^ знаходиться всередині квадратних дужок, так як тільки в цьому випадку він має значення "не". При використанні символу ^ поза квадратних дужок, він має зовсім інше значення, про що піде мова трохи пізніше.

Регулярний вираз можна уточнити за допомогою кваліфікаторів - так називаються символи +, ? , * . Кваліфікатори говорять про те, скільки разів послідовність символів може зустрітись у рядку і вказуються безпосередньо після тієї частини виразу, до якої вони застосовуються:

- **"a+"** - хоча б один а (рядки "abv" і "абва" відповідають цьому виразу, а рядок "укр" - ні);

- **"a?"** - Нуль або один а (рядки "abv" і "укр" відповідають цьому виразу, а рядок "абва" - ні);

- **"a*"** - нуль або більше а (рядки "abv" і "абва" і "укр" відповідають цьому виразу).

Кордони - це числа у фігурних дужках, що вказують на кількість входжень у рядок фрагмента виразу, що безпосередньо передують кордону:

- **"xy{2}"** відповідає рядку, в якому за x слідує два у;

- **"xy{2,}"** відповідає рядку, в якому за x слідує не менше двох у (може бути і більше);

- **"xy{2,6}"** відповідає рядку, в якому за x слідує від двох до шести у.

Для зазначення кількості входжень не тільки одного символу, а їх послідовності, використовуються круглі дужки:

- **"x(yz){2,6}"** відповідає рядку, в якому за x слідує від двох до шести послідовностей yz;

- **"x(yz)*"** відповідає рядку, в якому за x слідує нуль і більше послідовностей yz;

Іноді буває зручно створювати регулярний вираз таким чином, щоб можна було, наприклад, сказати, що, принаймні, за одним з рядків "морська", слідує точно рядок "хвиля". Для цього регулярний вираз розбивають на підвирази за допомогою круглих дужок: **(морська)* хвиля**. Це вираз відповідає рядкам "хвиля", "морська хвиля", "морська морська хвиля" і т.д.

У регулярному виразі можна вказати, чи повинен конкретний підвираз зустрічатися на початку, в кінці рядка або та на початку й в кінці рядка. Символ `^` відповідає початку рядка: `"^xz"`.

Такий вираз відповідає будь-якому рядку, що починається з `xz`. Зверніть увагу, що в цьому випадку символ `^` ставиться за межами вирази в дужках, наприклад: `"^[a-z]"`.

Знак долара `$` відповідає кінцю рядка: `"xz$"`.

Цей регулярний вираз відповідає будь-якому рядку, що закінчується на `xz`.

Валідація форми з боку браузера (HTML5).

Основний атрибут тега `<input>`, що дозволяє задавати різні елементи форми – це `type`. Для кожного елемента існує свій список атрибутів, які визначають його вигляд і характеристики. У HTML5 додано ще більше десятка нових елементів.

Крім того, до появи стандарту HTML5, при використанні форми на вашому сайті, ви повинні були пропускати введення текст через JavaScript для перевірки. Атрибут `pattern` вказує регулярний вираз, згідно з яким потрібно вводити і перевіряти дані в полі форми. Якщо присутній атрибут `pattern`, то форма не буде відправлятися, поки поле не буде заповнено правильно.

`<input type="text" title="Принаймні вісім символів, що містять хоча б одну цифру, один символ нижнього та верхнього регістру" required pattern="(?!.*[a-z A-Z0-9]){8,}" />`

Використовуйте глобальний атрибут `title` для доповнення вбудованої помилки своїм описом шаблону для допомоги користувачеві. У той самий час форма виведення атрибута `title` відрізняється від браузера до браузера.

Нижче наведені глобальні атрибути подій, які можуть бути додані в теги HTML5 для визначення подальшої дії (таблиця 1.1).

Таблиця 1.1 – Події форми

Атрибут	Статус	Опис
<code>onblur</code>		Скрипт виконується, коли елемент втрачає фокус
<code>onchange</code>		Скрипт виконується, коли елемент змінився
<code>oncontextmenu</code>	новий	Скрипт виконується, коли контекстне меню спрацьовує
<code>onfocus</code>		Скрипт виконується, коли елемент отримує фокус
<code>onformchange</code>	новий	Скрипт виконується, коли форма змінена

Продовження таблиці 1.1

Атрибут	Статус	Опис
onforminput	новий	Скрипт виконується, коли форма отримує користувальницький введення
oninput	новий	Скрипт виконується, коли елемент отримує користувальницький введення
oninvalid	новий	Скрипт виконується, коли елемент недійсний
onreset	не підтримується в HTML5	Скрипт виконується, коли форма скидається
onselect		
onsubmit		Скрипт виконується, коли елемент вибрано

API і DOM є фундаментальними частинами специфікації HTML5. За допомогою нового DOM API можна звертатися до наступних властивостей валідації:

- **willValidate** true, якщо для форми буде виконуватися валідація:

```
<div id="one"></div>
<input type="text" id="two" />
<input type="text" id="three" disabled />
<script>
document.getElementById('one').willValidate; //false
document.getElementById('two').willValidate; //true
document.getElementById('three').willValidate; //false
</script>
```

- **customError** true, якщо встановлена кастомна помилка:

```
<input id="foo" />
<input id="bar" />
<script>
document.getElementById('foo').validity.customError; //false
document.getElementById('bar').setCustomValidity('Invalid');
document.getElementById('bar').validity.customError; //true
</script>
```

- **patternMismatch** true, якщо value не співпадає з патерном:

```
<input id="foo" pattern="[0-9]{4}" value="1234" />
<input id="bar" pattern="[0-9]{4}" value="ABCD" />
<script>
document.getElementById('foo').validity.patternMismatch; //false
document.getElementById('bar').validity.patternMismatch; //true
</script>
```

- **rangeOverflow** true, якщо value більше ніж max:

<script>
document.getElementById('foo').validity.rangeOverflow; //false
document.getElementById('bar').validity.rangeOverflow; //true
</script>
- **rangeUnderflow** true, якщо value менше ніж max:

<script>
document.getElementById('foo').validity.rangeUnderflow; //false
document.getElementById('bar').validity.rangeUnderflow; //true
</script>
- **stepMismatch** true, якщо value не збігається з кожним step

<script>
document.getElementById('foo').validity.stepMismatch; //false
document.getElementById('bar').validity.stepMismatch; //true
</script>
- **tooLong** true, якщо символів у value більше ніж зазначено в
maxlength

<script>
document.getElementById('foo').validity.tooLong; //false
document.getElementById('bar').validity.tooLong; //true в Opera, false в
інших браузерах.
</script>
- **typeMismatch** true, якщо value не валідно для атрибута type

<script>
document.getElementById('foo').validity.typeMismatch; //false
document.getElementById('bar').validity.typeMismatch; //true
</script>
- **valueMissing** true, якщо елемент має атрибут require, але не має
значення в value

<script>
document.getElementById('foo').validity.valueMissing; //false

```

document.getElementById('bar').validity.valueMissing; //true
</script>
– valid true, якщо елемент валідний за всіма параметрами
<input id="valid-1" type="text" required value="foo" />
<input id="valid-2" type="text" required value="" />
<script>
document.getElementById('valid-1').validity.valid; //true
document.getElementById('valid-2').validity.valid; //false
</script>

```

Перевірити валідацію всієї форми можна так:

```

<form id="form-1">
  <input id="input-1" type="text" required />
</form>
<form id="form-2">
  <input id="input-2" type="text" />
</form>
<script>
document.getElementById('form-1').checkValidity(); //false
document.getElementById('input-1').checkValidity(); //false
document.getElementById('form-2').checkValidity(); //true
document.getElementById('input-2').checkValidity(); //true
</script>

```

Повідомлення валідації можна подивитися наступним чином:

```

<input type="text" id="foo" required />
<script>
document.getElementById('foo').validationMessage;
//Chrome --> 'Please fill out this field.'
//Firefox --> 'Please fill out this field.'
//Safari --> 'value missing'
//IE10 --> 'This is a required field.'
//Opera --> ''
</script>

```

Встановити для користувача повідомлення про помилку можна так:

```

<input type="password" id="pass1" name="pass1" pattern="\w{6,}" re-
quired autofocus onchange="this.setCustomValidity(this.validity.patternMismatch
? 'Password must contain at least 6 characters' : '');"/>
або

```

```

if (input.willValidate )
{
    if(input.checkValidity())
    {
        document.getElementById('login').setCustomValidity('Login (' +
document.getElementById('login').value + ') must contain at least 7 characters ');
    }
    else { document.getElementById('login').setCustomValidity("");}
}

```

Обробка даних на клієнтському боці.

Об'єкт **String** в JavaScript надає набір методів, які підтримують регулярні вирази. Перший з них - це метод **search()**, що використовується для пошуку рядка для відповідності певному регулярному виразу. Подивіться на наступний приклад:

```

<script language="JavaScript">
// визначаємо рядок для пошуку
var str = "The Matrix";
// визначаємо шаблон пошуку
var pattern = /trinity/;
// шукаємо і повертаємо результат
if(str.search(pattern) == -1)
{
    alert("Трініті відсутня в матриці");
} else
{
    alert("Трініті в матриці на символі " + str.search(pattern));
}
</script>

```

Метод **search ()** повертає позицію підрядка, що відповідає регулярному виразу або -1 у разі відсутності такої відповідності. У нашому прикладі видно, що шаблон «trinity» у рядку «The Matrix» відсутній¹, тому ми і отримуємо повідомлення про помилку. Якщо змінити регулярний вираз на **var pattern = / tri /**, то результат пошуку буде позитивним.

Об'єкт **String** також надає метод **match()**, який може розцінюватися, як близький родич методу **search ()**. Однак метод **search ()** повертає позицію, де знаходиться відповідність. Метод **match ()** працює трохи по-іншому: він застосовує шаблон до рядка і повертає масив знайдених значень.

```

<script language="JavaScript">
// визначаємо рядок
var str = "Mississippi";
// визначаємо шаблон
var pattern = /is./;
// перевіряємо на входження
// поміщаємо результат у масив

```

```

var result = str.match(pattern);
// display matches
for(i = 0; i < result.length; i++)
{ alert("Відповідність #" + (i+1) + ": " + result[i]);}
</script>

```

Перегляньте цей приклад у браузері, і ви отримаєте повідомлення, що показує перший результат відповідності. Ось такий:

Відповідність #1: iss

У цьому прикладі було задано регулярний вираз «is.». Він знайде рядок «is», за яким слідує будь-який символ (оператор «.» наприкінці шаблону знаходить все що завгодно в рядку). Якщо ви подивитесь на рядок, в якому ми виробляли пошук, ви побачите, що в ньому є два входження цього шаблону. Для отримання всіх входжень шуканого підрядка необхідно додати

```

// визначаємо шаблон і глобальний модифікатор
var pattern = /is./g;

```

Доданий модифікатор «g» забезпечує знаходження всіх входжень шаблону в рядок і збереження в масив, що повертається. Далі будуть розглянуті й інші модифікатори.

Попередній набір прикладів демонстрував можливості пошуку об'єкта String. Ви також можете здійснювати операції пошуку і заміни за допомогою методу **replace()**, який приймає регулярний вираз і значення для його заміни. Ось так:

```

<script language="JavaScript">
// визначаємо рядок
var str = "Welcome to the Matrix, Mr. Anderson";
// замінюємо один рядок на інший
str = str.replace(/Anderson/, "Smith");
// показуємо новий рядок
alert(str);
</script>

```

Результатом виконання скрипта є заміна підрядка «Anderson» рядком «Smith».

Модифікатор «g» може використовуватися для заміни декількох входжень шаблону в рядок:

```

<script>
var str = "yo ho ho and a bottle of gum";
// повертає "yoo hoo hoo and a bottle of gum"
alert(str.replace(/o\s/g, "oo "));
</script>

```

Тут мета-символ «\s» позначає пробіли після «yo» та «ho» та замінює на «oo».

Також ви можете використовувати нечутливий до регістру пошук за шаблоном - просто додайте модифікатор «i» наприкінці шаблону.

Об'єкт String також надає метод **split()**, який може бути використаний для розділення одного рядка на окремі частини на основі особливого значення поділу. Ці частини потім поміщаються в масив для подальшої обробки. Демонструє це наступний приклад:

```
<script language="JavaScript">
// визначаємо рядок
var friends = "Joey, Rachel, Monica, Chandler, Ross, Phoebe";
// поділяємо на частини за допомогою ком
var arr = friends.split(", ");
</script>
```

У JavaScript версії 1.1 та нижче, ви можете використовувати тільки рядкові значення як роздільники. JavaScript 1.2 змінює все це, тепер ви можете розділяти рядки навіть на основі регулярних виразів.

Щоб краще це зрозуміти, розглянемо наступний рядок, яка демонструє поширену проблему: нерівну кількість пробілів між значеннями поділу:

Neo | Trinity | Morpheus | Smith | Tank

Тут символ «|» використовується для розділення різних імен. І кількість пропусків між різними «|» різне - це означає, що перш, ніж ви зможете використовувати різні елементи рядка, ви змушені видалити зайві пробіли навколо них. Поділ з використанням регулярного виразу як роздільника є елегантним вирішенням цієї проблеми, що ми і бачимо на наступному прикладі:

```
// визначаємо рядок
var str = "Neo| Trinity |Morpheus | Smith| Tank";
// визначаємо шаблон
var pattern = /\s*\|s*/;
// поділяємо рядок за допомогою регулярного виразу як роздільника
result = str.split(pattern);
```

Результатом роботи цього коду буде масив, що містить імена, без жодного видалення пробілів.

Отже, всі приклади в цій статті пов'язані з об'єктом String для демонстрації потужності реалізації регулярних виразів в JavaScript. Але JavaScript також надає базовий об'єкт об'єкт Regular Expression, , сенс існування якого - пошук за шаблоном у рядках і змінних.

Цей об'єкт має три корисних методів. Ось вони:

— test()
test() – перевіряє рядок на входження за шаблоном;

— ex
exec()— повертає масив знайдених входжень в рядку, дозволяючи розширену роботу з регулярними виразами;

— so
compile() — після того, як регулярний вираз пов'язано з об'єктом Regular Expression.

Розглянемо простий приклад:

```
<script language="JavaScript">  
// визначемо рядок  
var str = "The Matrix";  
// створюємо об'єкт RegExp  
var character = new RegExp("tri");  
// шукаємо за шаблоном у рядку  
if(character.test(str)) {  
    alert("User located in The Matrix.");  
} else {  
    alert("Sorry, user is not in The Matrix.");  
}  
</script>
```

Це схоже на один з найперших прикладів цієї статті. Тим не менш, як ви бачите, він має зовсім іншу реалізацію.

Основна відмінність знаходиться в тому, що створюється об'єкт Regular Expression для пошуку за допомогою регулярного виразу. Він створюється за допомогою ключового слова «new», наступного за конструктором об'єкта. За визначенням, конструктор приймає два параметри: шаблон для пошуку і модифікатори, якщо вони мають місце бути (у цьому прикладі їх немає).

Наступним кроком після створення об'єкта, є його використання. Тут ми використовували метод **test()** для пошуку входження у відповідності з шаблоном. Типово, цей метод приймає строкову змінну і порівнює її з шаблоном, переданим в конструктор об'єкта. У разі знаходження відповідності, він повертає true, у протилежному ж випадку false. Очевидно, що це більш логічна реалізація, ніж використання методу search() об'єкта String.

Поведінка методу **exec()** схожа на те, що робить метод match() об'єкта String. Подивимось:

```
<script language="JavaScript">  
// визначаємо рядок  
var place = "Mississippi";  
// вказуємо шаблон  
var obj = /is./;  
// шукаємо входження, поміщаємо результат в масив
```



```
result = obj.exec(place);  
</script>
```

Метод `exec()` повертає відповідність зазначеному регулярному виразу, якщо така є, як масив. Ви можете звернутися до першого елемента масиву, щоб отримати знайдений підрядок, а також її розташування за допомогою методу `index()`.

Головна відмінність між методами `match()` і `exec()` в переданих параметрах. Перший вимагає шаблон як аргумент, другий же вимагає рядок для перевірки.

І це ще не все. У методу `exec()` є можливість продовжити пошук у рядку для знаходження аналогічного входження без вказівки модифікатора «g». Протестуємо цю можливість за допомогою наступного прикладу:

```
<script language="JavaScript">  
  // визначимо рядок  
  var place = "Mississippi";  
  // визначаємо шаблон  
  var obj = /is./;  
  // шукаємо всі входження в рядок  
  // показуємо результат  
  while((result = obj.exec(place)) != null)  
  {  
    alert("Found " + result[0] + " at " + result.index);  
  }  
</script>
```

У даному прикладі використовується цикл «while» для виклику методу `exec()` до тих пір, поки не буде досягнут кінець рядка (на якому об'єкт поверне `null` і цикл закінчиться). Це можливо, тому що кожного разу, викликаючи `exec()`, об'єкт `Regular Expression` продовжить пошук з того місця, на якому закінчив.

Інша цікава особливість цього коду полягає у створенні об'єкта `Regular Expression`. Ви напевно помітили, що, на відміну від попереднього прикладу, тут не використовується конструктор для створення об'єкта. Замість цього, шаблон просто застосовується до змінної. Думайте про це просто як про більш короткий спосіб створення об'єкта `Regular Expression`.

У цьому прикладі використовується кілька регулярних виразів для перевірки даних, що введені у форму користувачем, щоб перевірити правильність їх формату. Цей тип перевірки на боці клієнта вкрай важливий у мережі для того, щоб бути впевненим у правильності і безпеки даних, що надходять.

Для забезпечення зручності заповнення форм, а також коректності введених даних необхідно правильно визначати допустимі для введення фо-

рмати даних. Також необхідно інформувати користувача про те, в якому форматі слід вводити інформацію.

У плані реалізації перевірки форм найбільш вдалим є використання регулярних виразів, які дозволяють підібрати шаблон в переважній більшості випадків.

Обробка даних на боці сервера.

PHP підтримує два види записи регулярних виражень: POSIX (Portable Operating System Interface, інтерфейс переносної операційної системи) та Perl (PCRE - Perl Compatible Regular Expression). Станом на PHP 5.3.0, розширення для застосування регулярних виразів POSIX вважається застарілим.

У загальному випадку, функції для роботи з регулярними виразами виконуються більш повільно, ніж строкові функції, що мають аналогічні можливості. Тому, якщо можна без шкоди для ефективності програми використовувати рядкові функції, їх слід використовувати.

PHP підтримує ряд функцій для роботи з Perl- сумісними регулярними виразами.

int preg_match (string pattern, string subject [, array matches])

Ця функція шукає в рядку subject відповідність регулярному виразу pattern. Якщо заданий необов'язковий параметр matches, то результати пошуку поміщаються в масив.

mixed preg_replace (mixed pattern, mixed replacement, mixed subject[int limit])

Ця функція шукає в рядку subject відповідності регулярному виразу pattern, і замінює їх на replacement. Необов'язкового параметр limit задає число відповідей, які треба замінити. Якщо цей параметр не вказаний, або дорівнює -1, то замінюються всі знайдені відповідності.

int preg_match_all (string \$pattern , string \$subject [, array &\$matches [, int \$flags = PREG_PATTERN_ORDER [, int \$offset = 0]]])

Функція шукає у рядку subject всі збіги з шаблоном pattern і поміщає результат в масив matches у порядку, що визначається комбінацією прапорів flags. Після знаходження першої відповідності наступні пошуки будуть здійснюватися не з початку рядка, а від кінця останнього знайденого входження.

Параметр flags регулює порядок виведення збігів у багатовимірному масиві, що повертається:

- PREG_PATTERN_ORDER (за замовчуванням) впорядковує результати так, що елемент *\$matches[0]* містить масив повних входжень шаблону, елемент *\$matches[1]* містить масив входжень першої подмаски, і так далі.

- PREG_SET_ORDER впорядковує результати так, що елемент *\$matches[0]* містить перший набір входжень, елемент *\$matches[1]* містить другий набір входжень, і так далі.

- PREG_OFFSET_CAPTURE для кожної знайденого підрядка буде вказана її позиція у вихідному рядку. Необхідно пам'ятати, що цей прапор змінює формат масиву matches, що повертається, в масив, кожен елемент

якого містить масив, що розміщує в індексі з номером 0 знайдений підрядок, а зсув цього підрядка в параметрі `subject` - в індексі 1.

Додатковий параметр `offset` може бути використаний для вказівки альтернативної початкової позиції для пошуку.

`array preg_split (string $pattern , string $subject [, int $limit = -1 [, int $flags = 0]])`

Функція розбиває рядок `subject` за регулярним виразом `pattern`.

Якщо вказаний параметр `limit`, функція повертає не більше, ніж `limit` підрядків, частина рядка, що залишилася, буде повернута в останньому підрядку.

Параметр `flags` може бути будь-якою комбінацією наступних прапорів (об'єднаних за допомогою побітового оператора `|`):

- `PREG_SPLIT_NO_EMPTY` – функція `preg_split ()` повертає тільки непорожні підрядки.

- `PREG_SPLIT_DELIM_CAPTURE` – вираз, укладений в круглі дужки в шаблоні, що розподіляє, також вилучається з заданого рядка і повертається функцією.

- `PREG_SPLIT_OFFSET_CAPTURE` – для кожного знайденого підрядка буде вказана її позиція у вихідному рядку. Необхідно пам'ятати, що цей прапор змінює формат масиву, що повертається: кожен елемент буде містити масив, що розміщує в індексі з номером 0 знайдений підрядок, а зсув цього підрядка в параметрі `subject` - в індексі 1.

1.4.2 Методи шифрування даних

Більшість сучасних web додатків працює з важливою та конфіденційною інформацією користувача, яка зберігається на сервері в базі даних (логін, пароль, IP- адреси, кредитні картки) або в масиві `session`, на клієнті як проміжні дані - в `cookies`, `WebStorage`.

Якщо ваш додаток працює з фінансовими, медичними або просто з дуже важливими даними - використовуйте `HTTPS` - розширення протоколу `HTTP`, що підтримує шифрування. Дані, що передаються за допомогою протоколу `HTTPS`, «упаковуються» в криптографічний протокол `SSL` або `TLS`, тим самим забезпечується захист цих даних. На відміну від `HTTP`, для `HTTPS` за замовчуванням використовує TCP-порт 443. Дані між браузером і веб-сервером передаються в зашифрованому вигляді і не можуть бути розшифровані у разі перехоплення шніффером.

Однак протокол `SSL/SSH` захищає дані, якими обмінюються клієнт і сервер, але не захищають самі збережені дані, оскільки `SSL` - протокол шифрування на рівні сеансу передачі даних. У цьому випадку, якщо зловмисник може отримати безпосередній доступ до даних (в обхід веб -сервера) і може витягти дані, що цікавлять, порушити їх цілісність або підмінити їх.

У такому випадку рекомендується застосовувати в web додатках шифрування та/або хешування даних. Шифруванням називається процес перетворення даних у формат, в якому вони можуть бути прочитані (у всякому разі, теоретично) тільки передбачуваним одержувачем повідомлення. Одержувач розшифровує дані за допомогою ключа чи секретного пароля. Хешування здійснюється хеш-функціями для отримання хеш-коду.

PHP підтримує велику кількість алгоритмів шифрування і надає розширення та бібліотеки з готовими методами, не вимагаючи професійної підготовки в області криптографії та детального вивчення алгоритмів шифрування, для вирішення такого роду завдань. Деякі з них представлені нижче.

string hash (string \$algo , string \$data [, bool \$raw_output = false])

Ця функція повертає рядок, що містить обчислений хеш-код у шістнадцятковому кодуванні в нижньому регістрі за допомогою обраного algo алгоритму хешування і текста для хешування data. Якщо *raw_output* заданий як **TRUE**, то повертається хеш-код у вигляді бінарних даних.

Генерація хеш-коду на основі ключа, використовуючи метод HMAC, прямим методом шифрування здійснюється на основі

string hash_hmac (string \$algo , string \$data , string \$key [, bool \$raw_output = false])

Дана функція на основі обраного алгоритму хешування (список алгоритмів, що підтримуються, можна дізнатися за допомогою `hash_algos ()`), повідомлення data для хешування і загального секретного ключа key. Повертає рядок, що містить обчислений хеш-код у шістнадцятковому кодуванні в нижньому регістрі. Якщо *raw_output* заданий як **TRUE**, то повертається хеш-код у вигляді бінарних даних.

Ініціалізація інкрементального контексту хешування здійснюється за допомогою функції

resource hash_init (string \$algo [, int \$options = 0 [, string \$key = NULL]])

Параметрами функції є ім'я algo обраного алгоритму хешування, необов'язкові налаштування options для генерації хеша (в даний час підтримується тільки один варіант: `HASH_HMAC`, причому в даному випадку параметр key повинен бути вказаний) і власне загальний секретний ключ key, який буде використовуватися з методом хешування HMAC. Значенням даної функції, що повертається, є ресурс хешування для використання у функціях `hash_update ()`, `hash_update_stream ()`, `hash_update_file ()` і `hash_final ()`.

bool hash_update (resource \$context , string \$data)

Ця функція на основі контексту хешування context, що повертається функцією `hash_init ()`, додає дані data в активний контекст хешування.

Завершує інкрементальне хешування та повертає результат у вигляді хеш-коду:

string hash_final (resource \$context [, bool \$raw_output = false])

Ця функція повертає рядок, що містить обчислений хеш-код у шістнадцятковому кодуванні в нижньому регістрі. Якщо *raw_output* заданий як **TRUE**, то повертається хеш-код у вигляді бінарних даних.

string mhash (int hash, string data [, string key])

Функція *mhash* () застосовує хеш-функцію *hash* до даних *data* і повертає результуючий хеш (що називають також *digest* / дайджест). Якщо *key* специфікований, повертається результуючий HMAC. HMAC це хешування з ключем для аутентифікації повідомлення, або просто дайджест повідомлення, який залежить від специфікованого ключа. Не всі алгоритми, що підтримуються в *mhash*, можуть використовуватися в режимі HMAC. При помилці повертає **FALSE**.

string mhash_keygen_s2k (int hash, string password, string salt, int bytes)

Генерує ключ довжиною *bytes* із заданого користувачем пароля відповідно до заданих ідентифікатором алгоритму *hash* (одна з констант *MHASH_hash_name*). Це алгоритм Salted S2K, як специфіковано в документі OpenPGP (RFC 2440).

Параметр *salt* зобов'язаний відрізнитися і бути досить довільним для кожного ключа, що генерується вами, щоб створювати різні ключі. *Salt* має фіксований розмір у 8 байтів і буде заповнюватися нулями, якщо ви надасте меншу кількість байтів.

Надані паролі користувачів реально не підходять для використання як ключі в алгоритмах криптографії, оскільки користувачі вибирають ключі, які вони можуть ввести з клавіатури. Ці паролі використовують тільки від 6 до 7 біт на символ (або менше). Рекомендується використовувати якусь трансформацію (на зразок цієї функції) наданого користувачем ключа.

Mcrypt можна використовувати для шифрування і дешифрування. *Mcrypt* може працювати з чотирма режимами шифрування (CBC, OFB, CFB і ECB). Якщо зв'язок є з бібліотекою *libmcrypt-2.4.x* або вище, ці функції можуть також працювати з блок-режимом шифрування *nOFB* і в режимі *STREAM*.

Для відкриття модуля конкретного алгоритму та режиму

resource mcrypt_module_open (string \$algorithm , string \$algorithm_directory , string \$mode , string \$mode_directory)

Ім'я алгоритму *algorithm* визначається однією з констант *MCRYPT_ciphername*, режим шифрування *mode* визначається однією з констант *MCRYPT_MODE_modename*, а параметри *algorithm_directory* і

mode_directory використовуються для знаходження модуля шифрування. Ви можете вказати ім'я каталогу, або ви можете встановити параметр в порожній рядок (""), тоді використовується значення, встановлене в конфігураційному файлі php.ini mcrypt.algorithms_dir Якщо він не встановлений, використовується каталог за замовчуванням, який був складений в Libmcrypt (зазвичай /usr / місцеві / Бібліотека / Libmcrypt). Повертає дескриптор шифрування, або FALSE при помилці. Модуль закривається з використанням функції mcrypt_module_close (), як параметр якого вказується дескриптор шифрування.

Ви зобов'язані (в режимах CFB і OFB) або можете (в режимі CBC) підтримувати вектор ініціалізації / initialization vector (IV) для відповідної функції шифрування. Цей IV зобов'язаний бути унікальним і зобов'язаний бути однаковим при дешифруванні / шифруванні. Для даних, які зберігаються в зашифрованому вигляді, ви можете отримати висновок функції індексу, під яким дані зберігаються (наприклад, MD5-ключ імені файлу).

Для випадкової генерації вектора ініціалізації (IV) застосовується

```
string mcrypt_create_iv ( int $size [, int $source = MCRYPT_DEV_RANDOM ] )
```

Вектор IV може бути не таємним, хоча це може бути небажано. Крім того ви навіть можете надіслати його разом із зашифрованим текстом, не втрачаючи безпеки.

Параметри ви можете вказати наступні: розмір (size) IV і джерело (source) генерації IV (MCRYPT_RAND - генератор випадкових чисел, MCRYPT_DEV_RANDOM - читання даних з /dev/random і MCRYPT_DEV_URANDOM - читання даних з /dev/urandom). До 5.3.0, MCRYPT_RAND єдиний, що підтримується Windows. Вектор IV, як правило, повинен мати розмір блоку алгоритму, тому його розміри визначаються викликом mcrypt_enc_get_iv_size ().

Ви повинні завжди викликати цю функція перед викликом mcrypt_generic() або mdencrypt_generic()

```
int mcrypt_generic_init ( resource $td , string $key , string $iv )
```

Як параметри функції застосовується дескриптор шифрування td, ключ key, максимальний розмір якого визначається викликом mcrypt_enc_get_key_size () і вектор IV.

Для деініціалізації модуля шифрування застосовується функція **mcrypt_generic_deinit()** з дескриптором шифрування як параметра.

Для шифрування даних застосовується функція

```
string mcrypt_generic ( resource $td , string $data )
```

Дані **data** заповнюються "\ 0" для того, щоб переконатися, довжина даних відповідає n*blocksize. Ця функція повертає зашифровані дані. Слід зазначити, що довжина рядка, що повертається, може фактично бути більше, ніж на вході, через заповнення даних.

Для дешифрування даних застосовується функція **mdencrypt_generic()**.

Шифрування input-значення шифром TripleDES для 2.4.x і вище в режимі ECB

```
<?php
    $key = "this is a secret key";
    $input = "Let us meet at 9 o'clock at the secret place.";
    $td = mcrypt_module_open('tripledes', '', 'ecb', '');
    $iv = mcrypt_create_iv (mcrypt_enc_get_iv_size($td),
MCRYPT_RAND);
    mcrypt_generic_init($td, $key, $iv);
    $encrypted_data = mcrypt_generic($td, $input);
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);?>
```

Результатом є зашифрований \$encrypted_data рядок.

Модуль використовує функції OpenSSL для генерації та перевірки підпису і відбитків (шифрування) і відкриття (дешифрування) даних. PHP-4.0.4pl1 вимагає OpenSSL >= 0.9.6, але PHP-4.0.5 і вище, також буде працювати і з OpenSSL >= 0.9.5.

Функції openssl_get_xxx використовують ресурси key або certificate для формування позитивного ідентифікатора ключа ресурсу.

resource openssl_pkey_get_private (mixed key [, string passphrase])

Ця функція повертає позитивний ідентифікатор ключа ресурсу при успіху, FALSE при помилці.

Як ключ key може використовуватися:

- рядок формату file : / / path / to / file.pem. Зазначений файл повинен містити сертифікат в кодуванні PEM / закритий ключ (або може містити обидва);
- закритий ключ формату PEM.

Додатковий параметр passphrase повинен бути використаний, якщо зазначений ключ зашифрований (захищений паролем).

Функція openssl_pkey_get_public () вилучає відкритий ключ з сертифікату і готує його для використання іншими функціями.

resource openssl_pkey_get_public (mixed certificate)

Ця функція повертає позитивний ідентифікатор ключа ресурсу при успіху, FALSE при помилці.

Як сертифікат certificate може використовуватися:

- сертифікат ресурсу X.509;
- рядок формату / / шлях / до / file.pem. Зазначений файл повинен містити сертифікат в кодуванні PEM / закритий ключ (або може містити обидва);
- закритий ключ формату PEM.

Для шифрування даних `data` із закритим ключем `key` і збереження результатів у `crypted` використовується функція

`bool openssl_private_encrypt (string $data , string &$crypted , mixed $key [, int $padding = OPENSSL_PKCS1_PADDING])`

Шифровані дані за допомогою функції `openssl_private_encrypt ()` можуть бути розшифровані функцією `openssl_public_decrypt ()`. Параметр `padding` може бути `OPENSSL_PKCS1_PADDING`, `OPENSSL_NO_PADDING`. Повертає `TRUE` у разі успішного завершення або `FALSE` у разі виникнення помилки.

Для розшифровки даних, які були попередньо зашифровані функцією `openssl_public_encrypt()` використовується

`bool openssl_private_decrypt (string $data , string &$decrypted , mixed $key [, int $padding = OPENSSL_PKCS1_PADDING])`

Ця функція розшифровує дані `data` із закритим ключем `key` і зберігає результат у `decrypted`. Як параметр `padding` може бути використані `OPENSSL_PKCS1_PADDING`, `OPENSSL_SSLV23_PADDING`, `OPENSSL_NO_PADDING`, `OPENSSL_PKCS1_OAEP_PADDING`. Повертає `TRUE` у разі успішного завершення або `FALSE` у разі виникнення помилки.

Для розшифрування даних, що були попередньо зашифровані функцією `openssl_private_encrypt()` використовується функція `openssl_public_decrypt()`.

1.5 Варіанти завдань

Завдання № 1. Побудувати клієнтську (HTML5, JS) і серверну (PHP) частини для перевірки правильності даних HTML-форми з вказаними полями та обмеженнями (таблиця 1.2, 1.3).

Таблиця 1.2 – Варіанти завдань для валідації

| № варіанту | Поле1 (HTML5) | Поле2 (JS) | Поле3 (PHP) | Правила (JS та/або PHP) |
|------------|-------------------|------------|-------------|---|
| 1 | Login | Pass1 | Pass2 | Pass1=Pass2 |
| 2 | Credit Visa | Text | Email | |
| 3 | Credit MasterCard | Text | Date | Визначити скільки днів минуло з дати Birthday до поточної дати |
| 4 | Credit MasterCard | Date | Email | Визначити скільки днів до закінчення терміну дії кредитної картки |

Продовження таблиці 1.2

| № варіанту | Поле1 (HTML5) | Поле2 (JS) | Поле3 (PHP) | Правила (JS та/або PHP) |
|------------|----------------|-----------------|-------------------|-------------------------|
| 5 | Email | IP adress | WebURL | |
| 6 | IP adress | Pass | Age | |
| 7 | Height, step=3 | Weight, max=200 | Credit MasterCard | Weight < Height-100 |
| 8 | Login | Pass1 | Pass2 | Login≠Pass1 |
| 9 | Credit Visa | Email | IP adress | |
| 10 | Credit Maestro | Email | WebURL | |

Таблиця 1.3 – Вимоги до значень полів

| Тип поля | Опис правил |
|-------------------|--|
| Text | Непорожній рядок. У рядку допустимими символами вважаються літери, цифри, " ", "?", "!". |
| Login | Текстовий рядок, що містить як мінімум 7 символів букв і цифр, що починається з заголовної букви латинського алфавіту |
| Pass | Текстовий рядок, що містить як мінімум 6 символів букв і цифр |
| Height | Число для відображення росту в футах або см (1 ' = 30,48 см або 0,3048 м) |
| Weight | Число для відображення ваги в кг або фунтах (1 фунт = 0,454 кг) |
| Age | Ціле число, яке вказує вік (число в діапазоні від 1 до 100) |
| Email | Адреса електронної пошти - тільки один символ @, одна або більше точок, перевірка на існуючі домени верхнього рівня. |
| IP adress | Будь-яка IP-адреса в межах діапазону 192.168.1.0-192.168.1.255. |
| WebURL | Адреса сайту с перевіркою на існуючі протоколи (наприклад, http, https, ftp, file) |
| Date | Дата у форматі PPPP.ММ.ДД или pp.ММ.ДД |
| Credit Visa | Номер картки міжнародної платіжної системи VISA, що завжди починається з цифри «4», складається всього з 16 цифр (чотири групи, що складаються з чотирьох цифр) або з 13 цифр (перша група з чотирьох цифр і три групи з трьох). |
| Credit MasterCard | Номер картки MasterCard починається з цифри «5», друга цифра знаходиться в діапазоні від 1 до 5, складається з 16 цифр |
| Credit Maestro | Номер картки Maestro починається з цифр "3", "5" "6" і може складатися з 13,16 або 19 цифр. |

Задание №2. Реалізувати для одного поля форми механізм безпеки даних за вказаним методом шифрування (таблиця 1.4).

Таблиця 1.4 – Варіанти завдань для реалізації методів шифрування даних

| № варіанта | Поле | Метод |
|------------|-------------------|--|
| 1 | Pass1 | Метод hash з використанням таємного слова як соль |
| 2 | Credit Visa | Шифрування методом hash_update і запис значення в COOKIES |
| 3 | Credit MasterCard | Метод mcrypt з використанням вектора IV для шифрування і дешифрування даних |
| 4 | Credit MasterCard | Метод hash_update з використанням ключа, що генерується тегом HTML5 <keygen> |
| 5 | IP adress | Шифрування методом mcrypt і запис значення в COOKIES |
| 6 | IP adress | Шифрування методом hash_hmac ідентифікатора SESSION і запис значення в SESSION |
| 7 | Credit MasterCard | Метод mhash з використанням ключа, що генерується методом mhash_keygen |
| 8 | Pass1 | Метод hash_hmac з використанням ключа, що генерується тегом HTML5 <keygen> |
| 9 | Credit Visa | Шифрування методом mcrypt і запис значення в SESSION |
| 10 | Credit Maestro | OpenSSL з використанням закритого ключа та сертифікату |

1.6 Зміст звіту

Звіт з лабораторної роботи повинен містити:

- мета роботи;
- вихідний варіант завдання;
- HTML5 - файл з дизайном форми і реалізацією валідації форми браузером;
- JS – файл с блоком JavaScript для клієнтської перевірки даних, що відсилаються;
- PHP – файл для валідації та шифрування даних, що приймаються на боці сервера;
- результати роботи всіх вищенаведених модулів;
- висновки з виконаної роботи.

1.7 Контрольні запитання і завдання

- 1).. Якими спеціальними символами визначається кількість повторень групи символів при завданні регулярного виразу?
- 2).. Яким чином реалізувати з використанням HTML5 обмеження на введення значення дати, наприклад, введення дати, починаючи з поточної?
- 3).. Яким чином реалізувати з використанням HTML5 для одного елемента кілька обробників?
- 4).. Яким чином реалізувати для HTML5 форми виведення кастомного повідомлення у вигляді тексту на веб-сайт?
- 5).. Які функції JavaScript і PHP використовуються для перевірки відповідності вхідних даних типу integer?
- 6).. Які функції JavaScript і PHP використовуються для перевірки відповідності вхідного текстового рядка заданим шаблоном?
- 7).. У чому полягає відмінність процесу шифрування від хешування?
- 8).. Назвіть способи вирішення завдання, що є зворотним хешуванню?
- 9).. Що являє собою криптографічна сіль?
- 10) У чому полягають недоліки застосування незмінної солі?
- 11) У чому відмінність прямого й інкрементального способу хешування?
- 12) Які ви знаєте способи генерації ключів для шифрування?
- 13) У чому полягає відмінність розширення HASH і MHASH?

2 ВИВЧЕННЯ РОЗШИРЕННЯ PDO ДЛЯ ЗАБЕЗПЕЧЕННЯ АБСТРАКЦІЇ ДОСТУПУ ДО БАЗ ДАНИХ

2.1 Мета роботи

Вивчення методів керування підключенням, виконання запитів та вибірки рядків, транзакцій PDO для реалізації основних можливостей різних СУБД в web додатку.

2.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до виконання лабораторної роботи слід повторити синтаксис операторів мови SQL стосовно операцій вибірки (SELECT) і модифікації (INSERT, DELETE, UPDATE) даних.

Крім того, потрібно ознайомитися з програмними методами виконання запитів до бази даних, вибірки рядків, а також етапами виконання підготовленого запиту до бази даних із застосуванням інтерфейсу PDO [5,6].

2.3 Опис лабораторної установки

При виконанні лабораторної роботи використовується ПЕОМ під керуванням операційної системи Windows XP і старше, веб-сервер хатрр 2.3.3, середовище розробки Notepad++ 6.4.5.

2.4 Порядок виконання роботи і методичні вказівки з її виконання

Розширення об'єктів даних PHP (PDO) визначає простий і узгоджений інтерфейс для доступу до баз даних в PHP. Кожен драйвер бази даних, в якому реалізований цей інтерфейс, що може представити специфічний для баз даних функціонал у вигляді стандартних функцій розширення (рисунок 2.1). PDO не даремно розшифровується як PHP Data Object – так як взаємодія з базами даних здійснюється за допомогою об'єктів.



Рисунок 2.1 – Схема реалізації інтерфейсу доступу до баз даних в PHP

До основних завдань PDO відносять наступні:

- забезпечити стандартизоване API для реалізації основних можливостей різних СУБД;

- бути потенційно розширюваним, щоб розробники баз даних могли реалізовувати нові можливості своїх СУБД в PDO;
- забезпечити легке перенесення додатків між різними СУБД;
- PDO покликано забезпечити абстракцію засобів доступу до СУБД, але не абстракцію функціонала самих СУБД. Іншими словами: PDO *не* абстрагує саму *базу даних*, це розширення не переписує SQL запити та не емулює відсутній в СУБД функціонал. В PDO немає емуляції тих можливостей, які не підтримуються однією СУБД, але присутні в іншій;
- спростити створення нових драйверів для роботи з базами даних в PHP, забезпечуючи спрощений інтерфейс з “нутрощами” PHP, робота з якими займає найбільше часу.

Схема підготовки та виконання підготовленого запиту до бази даних (БД) з застосуванням інтерфейсу БД PDO, а також виведення результатів показана на рисунку 2.2.

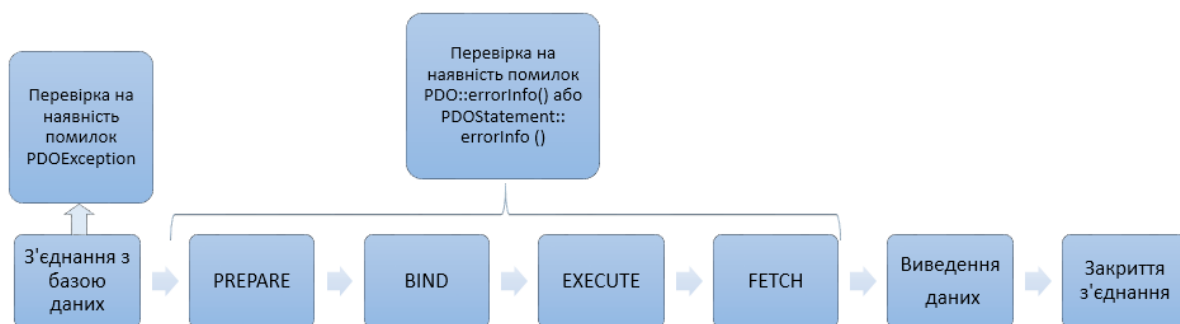


Рисунок 2.2 – Схема вибірки рядків з бази даних із застосуванням підготовленого запиту та виведення результатів

Далі будуть розглянуті основні принципи роботи з класами PDO та PDOStatement.

Підтримує наступні драйвери БД (переглянути список драйверів можливо за допомогою метода `PDO::getAvailableDrivers()`):

| | |
|--------------|--|
| PDO_CUBRID | Cubrid |
| PDO_DBLIB | FreeTDS / Microsoft SQL Server / Sybase |
| PDO_FIREBIRD | Firebird/Interbase 6 |
| PDO_IBM | IBM DB2 |
| PDO_INFORMIX | IBM Informix Dynamic Server |
| PDO_MYSQL | MySQL 3.x/4.x/5.x |
| PDO_OCI | Oracle Call Interface |
| PDO_ODBC | ODBC v3 (IBM DB2, unixODBC и win32 ODBC) |
| PDO_PGSQL | PostgreSQL |
| PDO_SQLITE | SQLite 3 и SQLite 2 |
| PDO_SQLSRV | Microsoft SQL Server / SQL Azure |
| PDO_4D | 4D |

З'єднання та закриття з'єднання з базою даних.

З'єднання з базою даних встановлюється тоді, коли створюється екземпляр класу PDO (наприклад, об'єкт `$pdh`), причому не має значення, який драйвер використовується. Його конструктор приймає параметри для того, щоб визначити джерело бази даних (відомий як DSN), і необов'язкові параметри для імені користувача (`username`), пароля (`password`) і масив встановлюваних опцій підключення (`driver_options`).

Формат запису конструктора наступний:

```
PDO::__construct ( string dsn [, string username [, string password  
[, array driver_options]]] )
```

– ім'я джерела даних PDO_MYSQL (DNS) потрібне як перший параметр конструктора при створенні нового об'єкта класу PDO, воно складено з наступних елементів:

- DSN prefix - приставка DSN, наприклад, "mysql:" или "mysqli";
- host - ім'я хоста, на якому знаходиться сервер бази даних;
- port - номер порту сервера бази даних;
- dbname - ім'я бази даних;
- unix_socket - сокет Unix MySQL (не повинен використовуватися з хостом або портом);
- charset – кодування (до PHP 5.3.6 цей елемент ігнорується).

Варіанти рядка підключення драйвера PDO_MYSQL:

```
mysql:host=localhost;dbname=testdb
```

```
mysql:host=localhost;port=3307;dbname=testdb
```

```
mysql:unix_socket=/tmp/mysql.sock;dbname=testdb
```

Варіант рядка підключення драйвера PDO_SQLITE:
`sqlite:/path/to/database.db`

Для створення бази даних в пам'яті, використовуйте: `sqlite::memory:`

Варіант рядка підключення драйвера для `sqlite (version 2)`:
`sqlite2:/path/to/database.db`

Для створення бази даних в пам'яті, використовуйте: `sqlite2::memory:`

Приклад створення об'єкта PDO та підключення до СУБД MySQL (`$dbh` розшифровується як "database handle").

```
<?php
```

```
$dsn = 'mysql:host=localhost;dbname=testdb';
```

```
$username = 'имя пользователя'; $password = 'пароль';
```

```
$options = array( PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8');
```

```
$dbh = new PDO ($dsn, $username, $password, $options);?>
```

У результаті успішного підключення до бази даних повертається екземпляр класу PDO. З'єднання залишається активним протягом життя цього об'єкта PDO. Для того, щоб завершити з'єднання, вам необхідно знищити об'єкт, гарантуючи, що всі інші посилання на нього будуть знищені – це можливо здійснити шляхом присвоєння NULL змінної, що містить об'єкт. Якщо ви не зробите цього явно, PHP автоматично закриває з'єднання після завершення скрипта.

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
// використовуємо з'єднання тут
// тепер закриваємо його
$dbh = null;
?>
```

Опції підключення можливо встановлювати трьома способами:

– При з'єднанні з базою даних найбільш часто використовувані опції з'єднання – це такі як PDO::ATTR_PERSISTENT для створення постійних підключень до серверів баз даних, які не закриваються наприкінці сценарію, але кешируються та використовуються повторно, коли інший сценарій запитує з'єднання з використанням тих же облікових даних, та PDO::MYSQL_ATTR_INIT_COMMAND, що дозволяє частково реалізувати вказівку кодування при створенні об'єкта PDO:

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass, array(
    PDO::ATTR_PERSISTENT => true,
    PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\'
));
?>
```

– При використанні метода PDOObj->setAttribute() для присвоювання атрибута PDO об'єкта, метода PDOStatement->setAttribute() для встановлення опцій конкретного драйверу:

```
<?php
$dbh = new PDO($connection_string);
$dbh->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE,
PDO::FETCH_OBJ);
$dbh->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
?>
```

– При використанні деяких функцій вибірки даних, наприклад, у методі PDO::prepare() як другий параметр враховуються опції драйвера (одна або більше пар ключ=>значення для встановлення значень атрибутів об'єкта PDOStatement):


```
<?php
$stmt = $dbh->prepare($sql, array(PDO::ATTR_CURSOR =>
PDO::CURSOR_FWDONLY));
?>
```

Якщо ви бажаєте написати код, який не залежить від типу бази даних, що підключається, рекомендується не використовувати специфічні для конкретного драйвера константи та конструкції запитів. Більше інформації за темою відмінних опцій різних СУБД та методах підключення до них можна знайти на php.net.

Блок try/catch рекомендується використовувати для оброблення в нього всіх PDO-операцій та користування механізмом винятків PDOException для виведення помилок:

```
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
    foreach($dbh->query('SELECT * from FOO') as $row) {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();}
?>
```

Виконання запитів до бази даних.

Метод **query()** виконує SQL запит statement без підготовки та повертає результуючий набір у вигляді об'єкта PDOStatement або FALSE при помилці.

```
PDOStatement PDO::query ( string $statement )
```

Приємною особливістю **query()** є те, що після виконання SELECT запиту можна одразу працювати з результуючим набором за допомогою курсора.

Метод query() потребує, щоб ви належним чином екранували всі дані, щоб уникнути SQL ін'єкцій та других питань. Тому його потрібно використовувати в запитах без зовнішніх даних.

```
.<?php
$conn->query("SET CHARACTER SET utf8");
function getFruit($conn) {
    $sql = 'SELECT name, color, calories FROM fruit ORDER BY name';
    foreach ($conn->query($sql) as $row) {
```

```
print $row['name'] . "\t";
print $row['color'] . "\t";
print $row['calories'] . "\n";
}}?>
```

Швидке отримання даних із запити.

PDO пропонує деякі допоміжні шляхи для отримання інформації із запити без використання `fetch*()` функцій і циклу на основі отриманого результату.

Метод **`PDOStatement::rowCount`** повертає кількість рядків, котрі були використані в ході виконання останнього запиту `DELETE`, `INSERT` або `UPDATE`, запущеного відповідним об'єктом `PDOStatement`.

```
$stmt = $db->query('SELECT * FROM table');
$row_count = $stmt->rowCount();
```

Інакше використовуйте метод **`PDOStatement::columnCount()`**, щоб узнати кількість стовбців в результуючому наборі, котрий представляє собою об'єкт `PDOStatement`.

Якщо об'єкт `PDOStatement` був повернутий з метода `query()`, число стовбців можна узнати одразу. Якщо об'єкт `PDOStatement` був повернутий з метода `PDO::prepare()`, точну кількість стовбців можна буде дізнатися тільки після запуску метода `PDOStatement::execute()`.

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
/* Підрахунок кількості стовбців в (неіснуючому) результуючому на-
борі */
$colcount = $sth->columnCount();
print("Перед викликом execute(), в результуючому наборі $colcount
стовбців (повинно бути 0)\n");
$sth->execute();
/*Підрахунок кількості стовбців у результуючому наборі*/
$colcount = $sth->columnCount();
print("Після виклику execute(), в результуючому наборі $colcount стов-
бців (повинно бути 2)\n");
?>
```

Метод **`PDO::lastInsertId`** повертає ID останнього вставленого рядка або останнє значення, яке видав об'єкт послідовності.

```
$db->query("INSERT INTO users SET
name='Vasya',address='Here',email='vasya@test.com'");
$insertId=$db->lastInsertId();
```

Якщо ви не використовуєте prepared statements, тоді необхідно для безпечного використання SQL запитів використовувати метод **quote()**, який повертає рядок, в якому проставлені лапки в строкових даних (якщо потрібно) та екрановані спеціальні символи усередині рядка підходящим для драйвера способом.

```
<?php
$conn = new PDO('sqlite:/home/lynn/music.sql3');
/*небезпечний рядок*/
$string = 'Naughty \' string';
print " Неекранований рядок: $string\n";
print " Екранований рядок:" . $conn->quote($string) . "\n";
//запит з умовою та екрануванням
$conn->query('SELECT * FROM table WHERE id = ' . $conn->quote($id));
?>
```

Виведення результату:

Неекранований рядок: Naughty ' string

Екранований рядок: 'Naughty " string'

Метод **exec()** запускає SQL запит statement на виконання і використовується для операцій, які не повертають ніяких даних, крім кількості порушених ними записів. Дані всередині запиту повинні бути правильно екрановані. Використовується, наприклад, при видаленні даних з бази даних.

```
<?php
$dbh = new PDO('odbc:sample', 'db2inst1', 'ibmdb2');
/ * Видаляємо всі записи з таблиці FRUIT * /
$count = $dbh->exec("DELETE FROM fruit WHERE colour = 'red'");
/ * Отримаємо кількість записів, що видалені * /
print("Видалено $count рядків.\n");
?>
```

Виконання підготовлених запитів.

Підготовлені вирази (англ. prepared statments) дозволяють визначити вираз один раз, а потім багаторазово його виконувати з різними параметрами, що підвищує швидкість виконання і економить трафік між додатком і базою даних. Також вони дозволяють відокремити змінні від запиту, що дозволяє зробити код безпечніше. З цією метою застосовується метод **prepare()**, який готує запит statement до виконання і повертає асоційований з цим запитом об'єкт PDOStatement.

Зверніть увагу, що після створення об'єктом PDO підготовленого виразу в класі PDOStatement, використовується тільки він, відповідно, в ньому є

свої функції `errorCode`, `errorInfo`, а також результат виконання запитів, також відразу ж зберігається в ньому.

```
PDOStatement PDO::prepare ( string $statement [, array $driver_options = array() ] )
```

Запит може містити іменовані (`:` name) або не іменовані (`?`) псевдозмінні, які будуть замінені реальними значеннями під час запуску запиту на виконання. Якщо ви визначили змінні знаком питання, то в функцію `execute` передайте масив значень, в тій, послідовності, в якій стоять змінні. Використовувати одночасно іменовані та неіменовані псевдоперемінні в одному запиті заборонено, необхідно вибрати щось одне.

```
<?php
/* Виконання SQL запиту з передачею йому масиву іменованих параметрів */
$sql = 'SELECT name, colour, calories FROM fruit
WHERE calories < :calories AND colour = :colour';
/* Створення об'єкту PDOStatement з послідовним курсором */
$stmt = $dbh->prepare($sql, array(PDO::ATTR_CURSOR =>
PDO::CURSOR_FWDONLY));
$stmt->execute(array(':calories' => 150, ':colour' => 'red'));
$red = $stmt->fetchAll();
/* Виконання запиту з передачею йому масиву не іменованих параметрів */
$stmt = $dbh->prepare('SELECT name, colour, calories FROM fruit
WHERE calories < ? AND colour = ?');
$stmt->execute(array(150, 'red'));
$red = $stmt->fetchAll();
?>
```

У попередньому прикладі пропущена функція **`bind*()`**, що задає значення іменованої або не іменованого псевдозмінної в підготовленому SQL запиті.

Існує три види функції типу `bind * ()`:

– **`bindValue()`** задає значення іменованої або не іменованої псевдозмінної в підготовленому SQL запиті

```
bool PDOStatement::bindValue ( mixed $parameter , mixed $value [, int $data_type = PDO::PARAM_STR ] )
```

```
<?php
/*Виконання підготовленого запиту з іменованими псевдозмінними*/
$stmt = $dbh->prepare('SELECT name, colour, calories
FROM fruit WHERE calories < :calories);
$stmt->bindValue(':calories', 150, PDO::PARAM_INT);
$stmt->execute();
```

```

/*Виконання підготовленого запиту з не іменованими псевдозмінними
(?) * /
$sth = $dbh->prepare('SELECT name, colour, calories
FROM fruit WHERE calories < ? AND colour = ?');
$sth->bindValue(1, 150, PDO::PARAM_INT);
$sth->bindValue(2, 'red', PDO::PARAM_STR);
$sth->execute();
?>

```

– **bindParam()** прив'язує іменований або не іменований параметр підготовлюваного SQL запиту з дійсною змінною, при зміні цієї змінної, не потрібно більше викликати ніяких додаткових функцій, можна відразу `execute()`. На відміну від `bindValue()`, змінна прив'язується за посиланням, і її значення буде обчислюватися під час виклику `execute()`

```

bool PDOStatement::bindParam ( mixed $parameter , mixed &$variable [,
int $data_type = PDO::PARAM_STR [, int $length [, mixed $driver_options ]]] )
<?php
/*Виконання запиту з прив'язкою PHP змінних*/
$calories = 150;
$colour = 'red';
/*Виконання підготовленого запиту з іменованими псевдозмінними */
$sth = $dbh-
>prepare('SELECT name, colour, calories FROM fruit WHERE calories <
:calories);
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->execute();
/*Виконання підготовленого запиту з не іменованими псевдо змінними
(?) * /
sth = $dbh->prepare('SELECT name, colour, calories
FROM fruit WHERE calories < ? AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->execute();
?>

```

– **bindColumn()** прив'язує змінну до заданого стовпця в результуючому наборі запиту. Кожен виклик `PDOStatement :: fetch()` або `PDOStatement :: fetchAll()` буде оновлювати всі змінні, задавати їм значення стовпців, з якими вони пов'язані.

```

bool PDOStatement::bindColumn ( mixed $column , mixed &$param [, int
$type [, int $maxlen [, mixed $driverdata ]]] )
<?php
$sql = 'SELECT name, colour, calories FROM fruit';
$stmt = $dbh->prepare($sql);

```

```

/* Зв'язування за номером стовпця */
$stmt->bindColumn(1, $name);
$stmt->bindColumn(2, $colour);
/* Зв'язування за іменем стовпця */
$stmt->bindColumn('calories', $cals);
$stmt->execute();
?>

```

Першим параметром *parameter* кожної з функцій є ідентифікатор параметру запиту. Для підготовлюваних запитів з іменованими змінними це буде ім'я у вигляді : name. Якщо використовуються не іменовані змінні (знаки питання «?») Це буде позиція псевдозмінної у запиті (починаючи з 1).

Третім параметром кожної з функцій *data_type* є явно заданий тип даних параметра (одна з констант PDO :: PARAM_*):

```

<?php $sth3->bindParam(':id',$id,PDO::PARAM_INT);
$stmt3->bindParam(':id',$id,PDO::PARAM_STR); ?>

```

Третім методом у ланцюжку при виконанні підготовлених виразів є метод **PDOStatement::execute()**, який запускає підготовлений запит. Метод execute також прекрасно працює коли ви повторюєте запит декілька разів.

Якщо у вашому SQL-виразі багато параметрів, то призначати кожному змінну вельми незручно. У таких випадках можна зберігати дані в масиві і передавати його:

```

<?php
// набір даних, які ми будемо вставляти
$data = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (?,
?, ?)");
$stmt->execute($data); ?>

```

Елемент \$ data [0] вставиться на місце першого placeholder'a, елемент \$data [1] - на місце другого, і т.д.

Вибірка рядків із запиту.

Для вилучення следующей строки з результуючого набору використовуйте метод **fetch()**:

```

mixed PDOStatement::fetch ([ int $fetch_style [, int $cursor_orientation =
PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0 ]]] )

```

Для вилучення всіх рядків з результуючого набір можна застосувати цикл за результатами вибірки:

```
<?php
// Вибираємо кожен рядок на кожній ітерації поки не виберемо всі ряд-
ки
while ($row = $res->fetch(PDO::FETCH_NUM)) {
echo $id = $row[0];
} ?>
```

У разі успішного виконання функції повертається значення, що залежить від режиму вибірки. Наприклад, для запиту зазначеного у прикладі нижче можна задати різні режими.

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();?>
```

Параметр *fetch_style* визначає, в якому вигляді PDO поверне цей рядок (одна з констант PDO::FETCH_*):

- PDO::FETCH_ASSOC повертає масив з назвами стовпців у вигляді ключів

```
<?php print_r($sth->fetch(PDO::FETCH_ASSOC));?>
```

Виведення результату:

```
Array ( [ID_Authors] => 1 [name] => Косминский Е.А. )
```

Доступ до даних:

```
<?php while($row = $sth->fetch(PDO::FETCH_ASSOC))
{ echo "<p>" . $row[' ID_Authors ' ] . "&nbsp;" . $row[' name ' ] . "</p>"; }?>
```

- PDO::FETCH_NUM повертає масив, з ключами у вигляді порядкових номерів стовпців (починаючи з 0):

```
<?php print_r($result->fetch(PDO::FETCH_NUM));?>
```

Виведення результату:

```
Array ( [0] => 1 [1] => Косминский Е.А. )
```

Доступ до даних:

```
<?php while($row ($result->fetch(PDO::FETCH_NUM))
{ echo "<p>" . $row[0] . "&nbsp;" . $row[1] . "</p>"; }?>
```

- PDO::FETCH_BOTH (за замовчуванням) повертає масив, індексований іменами стовпців результуючого набору, а також їх номерами (починаючи з 0):

```
<?php print_r($sth->fetch(PDO::FETCH_BOTH)); ?>
```

Виведення результату:

```
Array ( [ID_Authors] => 1 [0] => 1 [name] => Косминский Е.А. [1] => Косминский Е.А. )
```

Доступ до даних:

```
<?php while($row ($result->fetch(PDO::FETCH_BOTH))
{echo "<p>" . $row[0] . "&nbsp;" . $row['name'] . "</p>";}?>
```

– PDO::FETCH_OBJ створює анонімний об'єкт з властивостями, відповідними іменам стовпців результуючого набору:

```
<?php print_r($result->fetch(PDO::FETCH_OBJ)); ?>
```

Виведення результату:

```
stdClass Object ( [ID_Authors] => 1 [name] => Косминский Е.А. )
```

Доступ до даних:

```
while ($row = $result->fetch(PDO::FETCH_OBJ))
{
    echo $row->ID_Authors;
    echo $row->name;}>
```

– PDO::FETCH_LAZY комбінує PDO :: FETCH_BOTH і PDO :: FETCH_OBJ, створюючи новий об'єкт з властивостями, відповідними іменам стовпців результуючого набору:

```
<?php print_r($result->fetch(PDO::FETCH_LAZY)); ?>
```

Виведення результату:

```
PDORow Object ( [queryString] => select ID_Authors, name from Authors
order by ID_Authors [ID_Authors] => 1 [name] => Косминский Е.А. )
```

Доступ до даних:

```
$number=0;
while ($row = $result->fetch(PDO::FETCH_LAZY))
{
    echo $row->$number; echo "<br>";
    echo $row->name; echo "<br>";}
```

– PDO::FETCH_BOUND повертає TRUE і привласнює значення стовпців результуючого набору змінним PHP, які були прив'язані до цих стовпців методом PDOStatement::bindColumn().

– PDO::FETCH_CLASS створює і повертає об'єкт запитаного класу, привласнюючи значення стовпців результуючого набору іменованих властивостей класу. Якщо *fetch_style* включає в себе атрибут PDO::FETCH_CLASSTYPE (наприклад, PDO::FETCH_CLASS |

PDO::FETCH_CLASSTYPE), то ім'я класу, від якого потрібно створити об'єкт, буде взято з першого стовпця.

– PDO::FETCH_INTO оновлює існуючий об'єкт запитаного класу, привласнюючи значення стовпців результуючого набору іменованих властивостями об'єкта.

Значенням параметра *cursor_orientation* має бути одна з констант PDO :: FETCH_ORI_*, за замовчуванням PDO :: FETCH_ORI_NEXT, що дозволяє вибрати наступний рядок з результуючого набору.

Для об'єктів PDOStatement, що представляють прокручуваний курсор, параметр *cursor_orientation* яких приймає значення PDO :: FETCH_ORI_ABS, а величина *offset* означає абсолютний номер рядка, яку необхідно витягти з результуючого набору.

Для об'єктів PDOStatement, що представляють прокручуваний курсор, параметр *cursor_orientation* яких приймає значення PDO :: FETCH_ORI_REL, величина *offset* вказує, який рядок щодо поточного положення курсору буде витягнутий функцією PDOStatement :: fetch ().

Для повернення масиву, що містить всі рядки результуючого набору використовуйте метод **fetchAll()**:

```
array PDOStatement::fetchAll ([ int $fetch_style [, mixed $fetch_argument  
[, array $ctor_args = array() ]]] )
```

Масив являє кожен рядок або у вигляді масиву значень одного стовпця, або у вигляді об'єкта, імена властивостей якого збігаються з іменами стовпців.

```
<?php  
$sth = $dbh->prepare("select ID_Authors, name from Authors order by  
ID_Authors");  
$sth->execute();  
/* Вилучення всіх рядків результуючого набору, що залишилися */  
print("Вилучення всіх рядків результуючого набору, що залишили-  
ся:\n");  
$result = $sth->fetchAll();  
print_r($result);  
?>
```

Виведення результату:

```
Array ( [0] => Array ( [0] => 1 [1] => Косминский Е.А. )  
[1] => Array ( [0] => 2 [1] => Бочаров В.В. )  
[2] => Array ( [0] => 3 [1] => Уильям Уолкер Аткінсон )  
[3] => Array ( [0] => 4 [1] => Лоис Макмастер Буджолд ) )
```

За замовчуванням параметр *fetch_style* приймає значення PDO::ATTR_DEFAULT_FETCH_MODE, яке в свою чергу має значення PDO::FETCH_BOTH. Якщо потрібно витягти тільки унікальні рядки одного стовпця, потрібно передати побітове АБО (|) констант PDO::FETCH_COLUMN и PDO::FETCH_UNIQUE.

Щоб отримати асоціативний масив рядків, що згрупований за значеннями певного стовпця, потрібно передати побітове АБО (|) констант PDO::FETCH_COLUMN и PDO::FETCH_GROUP.

Сенс аргументу *fetch_argument* залежить від значення параметра *fetch_style*. Наприклад, якщо це параметр PDO::FETCH_COLUMN, то буде повернуто зазначений стовпець (індексація стовпців починається з 0).

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();
/* Вилучення всіх значень першого стовпчика */
$result = $sth->fetchAll(PDO::FETCH_COLUMN, 0);
foreach ($result as $row)
{ echo $name=$row; echo "<br>"; }
?>
```

Виведення результату:

```
Array(3)( [0] => string(5) => apple
[1] => string(4) => pear
[2] => string(10) => watermelon)
```

Для випадків, коли параметру *fetch_style* присвоєно значення PDO::FETCH_CLASS можуть бути задані аргументи конструктора класу *ctor_args*.

Якщо необхідно повернути дані одного стовпця наступного рядка результуючого набору можна скористатися функцією **fetchColumn()**, параметром якої є номер стовпчика, дані якого необхідно витягти:

```
string PDOStatement::fetchColumn ([ int $column_number = 0 ] )
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();
print("Отримання значення першого стовпчика наступного рядка:\n");
$result = $sth->fetchColumn();
print("Отримання значення другого стовпчика наступного рядка:\n");
$result = $sth->fetchColumn(1);
?>
```

Виведення результату:

Отримання значення першого стовпчика наступного рядка:

name = lemon

Отримання значення другого стовпчика наступного рядка:

colour = red

Для того, щоб витягти наступний рядок і отримати її у вигляді об'єкта скористайтеся функцією **fetchObject()**:

```
mixed PDOStatement::fetchObject ([ string $class_name = "stdClass" [, array $ctor_args ] ] )
```

Цей метод є альтернативою виклику `PDOStatement::fetch()` з параметром `PDO::FETCH_CLASS` або `PDO::FETCH_OBJ`. Як параметри цього метода застосовується ім'я класу створюваного об'єкта (*class_name*), елементи цього масиву будуть передані в конструктор класу (*ctor_args*).

Для завдання режиму вибірки за замовчуванням для всіх викликів об'єкта запиту використовується метод **setFetchMode()**, варіанти якого представлені нижче:

```
bool PDOStatement::setFetchMode ( int $mode )
```

```
bool PDOStatement::setFetchMode ( int $PDO::FETCH_COLUMN , int $colno )
```

```
bool PDOStatement::setFetchMode ( int $PDO::FETCH_CLASS , string $classname , array $ctorargs )
```

```
bool PDOStatement::setFetchMode ( int $PDO::FETCH_INTO , object $object )
```

Повертає 1 у разі успішного встановлення режиму або FALSE у разі виникнення помилки.

```
<?php
$sql = 'SELECT name, colour, calories FROM fruit';
try {
    $stmt = $dbh->query($sql);
    $result = $stmt->setFetchMode(PDO::FETCH_NUM);
    // цикл while() перебере весь результат запиту.
    while ($row = $stmt->fetch()) {
        print $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";
    }
    catch (PDOException $e) {
        print $e->getMessage();
    }
}>
```

Метод **closeCursor()** класу PDOStatement звільняє з'єднання з сервером, даючи можливість запускати інші SQL запити. Метод залишає запит в стані готовності до повторного запуску.

Виконання транзакцій даних.

Транзакція – це сукупність запитів до бази даних, які повинні бути обов'язково всі виконані. Якщо який-небудь запит не виконаний чи виконаний з помилкою, то транзакція відміняється і зміни в базі даних не відбуваються.

Це потрібно, щоб гарантувати збереження цілісності даних при кількох запитах, наприклад, при переказі грошових коштів за рахунку на рахунок.

Щоб виконати транзакцію в PDO необхідно перейти в режим ручного підтвердження запитів.

До речі, транзакції використовуються постійно, але зазвичай PDO працює в режимі автопідтвердження, тому всі транзакції складаються з одного запиту.

Ініціалізація транзакції здійснюється за допомогою методу PDO::beginTransaction(), який вмикає режим автоматичної фіксації транзакції (режим автопідтвердження). Після цього виконуємо стільки запитів до бази даних скільки необхідно зробити в цій транзакції.

В той час, як режим автоматичної фіксації вимкнений, зміни, внесені до бази даних через об'єкт екземпляру PDO, не застосовуються, доки ви не завершите (фіксуєте) транзакцію, викликавши функцію PDO::commit(), повертаючи з'єднання з базою даних в режим автоматичної фіксації до тих пір, поки наступний виклик PDO :: beginTransaction () не почне нову транзакцію.

```
<?php
/* Початок транзакції, відключення автоматичної фіксації */
$dbh->beginTransaction();
/* Вставка множини записів за принципом "все або нічого" */
$sql = 'INSERT INTO fruit (name, colour, calories) VALUES (?, ?, ?)';
$stmt = $dbh->prepare($sql);

foreach ($fruits as $fruit) {
    $stmt->execute(array( $fruit->name, $fruit->colour, $fruit->calories, ));
}
/* Фіксація змін*/
$dbh->commit();
/* З'єднання з базою даних знову в режимі автоматичної фіксації */
?>
```

Виклик PDO :: rollBack () відмінить всі зміни в базі даних, зроблені в рамках поточної транзакції, яка була створена методом PDO ::

beginTransaction (), і поверне з'єднання до режиму автоматичної фіксації. Якщо активної транзакції немає, буде показано виключення PDOException.

У наступному прикладі створюється транзакція і виконуються два запити, які модифікують дані в базі, а потім база повертається до вихідного стану. В MySQL, тим не менш, вираз DROP TABLE автоматично фіксує зміни, що знаходяться всередині транзакції, тому нічого відмінено не буде.

```
<?php
/* Починаємо транзакцію, вимикаємо автофіксацію */
$dbh->beginTransaction();

/* Змінюємо схему бази даних і дані в таблицях */
$stmt = $dbh->exec("DROP TABLE fruit");
$stmt = $dbh->exec("UPDATE dessert
    SET name = 'hamburger'");

/* Усвідомлюємо свою помилку і відмінюємо транзакцію */
$dbh->rollBack();
/* База даних повертається в режим автофіксації */
?>
```

Для перевірки чи є активні транзакції в даний час всередині драйвера використовується метод PDO :: inTransaction ().

2.5 Варіанти завдань

Остаточний варіант завдання має бути реалізований з використанням підготовлених виразів.

Варіант 1. Створити структуру бази даних для зберігання інформації про інформаційні ресурси бібліотеки (рисунок 2.3). Розрізняють три види ресурсів : книги, журнали, газети. Книги характеризуються назвою, унікальним номером (ISBN), видавництвом, роком видання, кількістю сторінок . У книги може бути довільна кількість авторів. Журнали характеризуються назвою, роком випуску, номером. Газети характеризуються назвою і датою виходу (день, місяць, рік). Книги і журнали можуть містити додаткові інформаційні ресурси (наприклад, компакт - диски) , які враховуються і реєструються окремо .

Сформулювати запити, які будуть виводити на екран інформацію про:

- книги, журнали і газети з вказаною назвою;
- книги, журнали і газети за вказаний часовий проміжок;
- книги зазначеного автора.

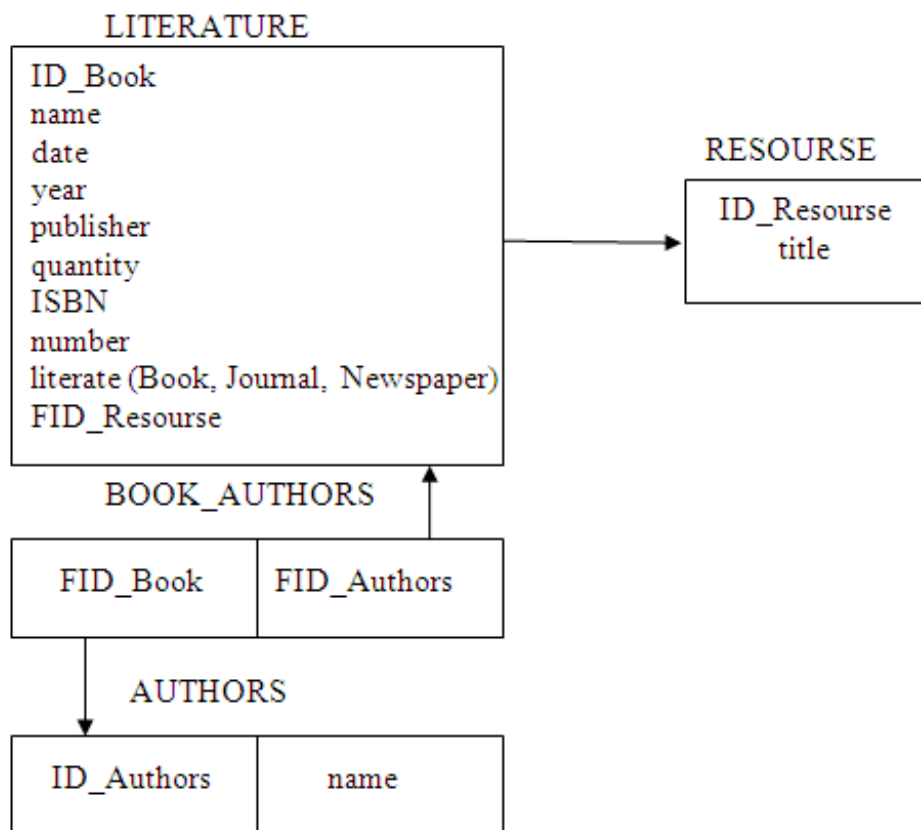


Рисунок 2.3 – Структура бази даних «Бібліотека»

Варіант 2. Створити структуру бази даних для зберігання інформації про розклад проведення занять (рисунок 2.4). Для кожного заняття задається день тижня, номер пари, аудиторія, група, дисципліна, вид заняття. Розрізняють три види занять - лекції (один викладач веде заняття з однією та більше групами), практичні заняття (один викладач працює з однією групою), лабораторні заняття (два викладача працюють з однією групою).

Сформувані запити і вивести розклад занять для:

- довільної групи зі списку;
- довільного викладача зі списку;
- довільної аудиторії зі списку.

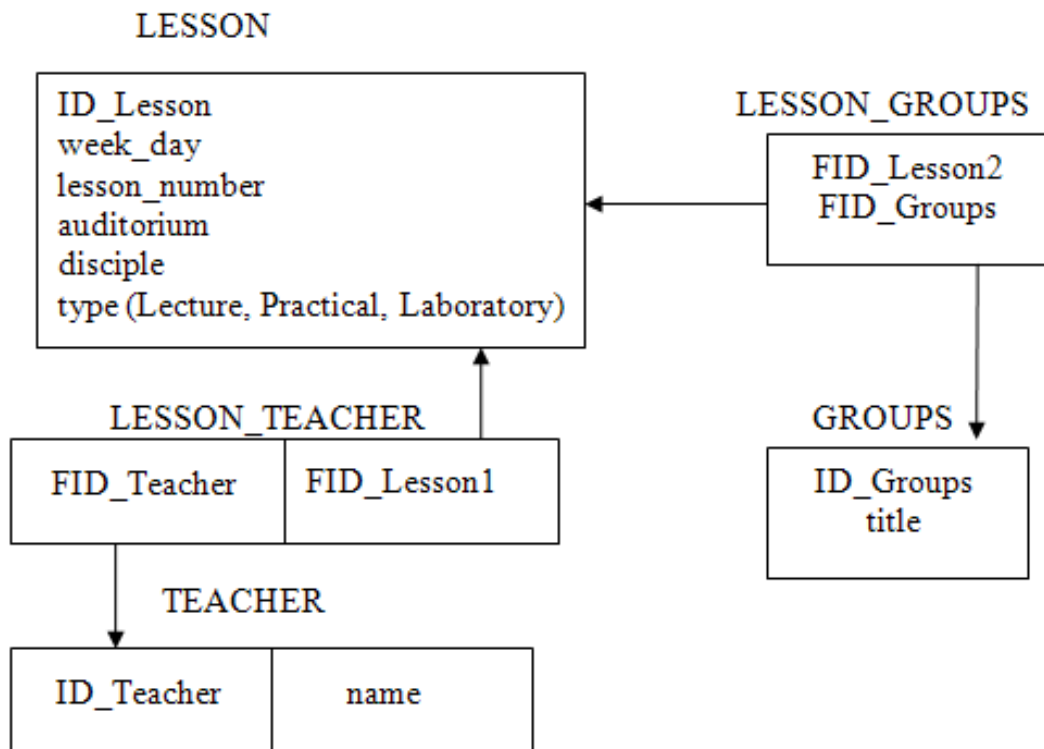


Рисунок 2.4 – Структура бази даних «Розклад занять»

Варіант 3. Створити структуру бази даних для зберігання інформації про час роботи над проектом кожного співробітника (рисунок 2.5). Розрізняють три види співробітників:

- начальник відділу може переглядати інформацію про час роботи над проектами всіх підлеглих;
- менеджер проекту може переглядати інформацію про час роботи над проектом кожного розробника;
- розробник може поміщати в базу інформацію про час роботи над проектом.

Організація може одночасно вести кілька проектів. Кожен співробітник може бути залучений до роботи над кількома проектами і виконувати різні ролі (наприклад, начальник відділу може бути менеджером одного проекту і розробником в іншому).

Сформулювати запити і вивести результати:

- час роботи будь-якого співробітника;
- час роботи над проектом всіх залучених співробітників;
- час роботи співробітників обраного відділу.

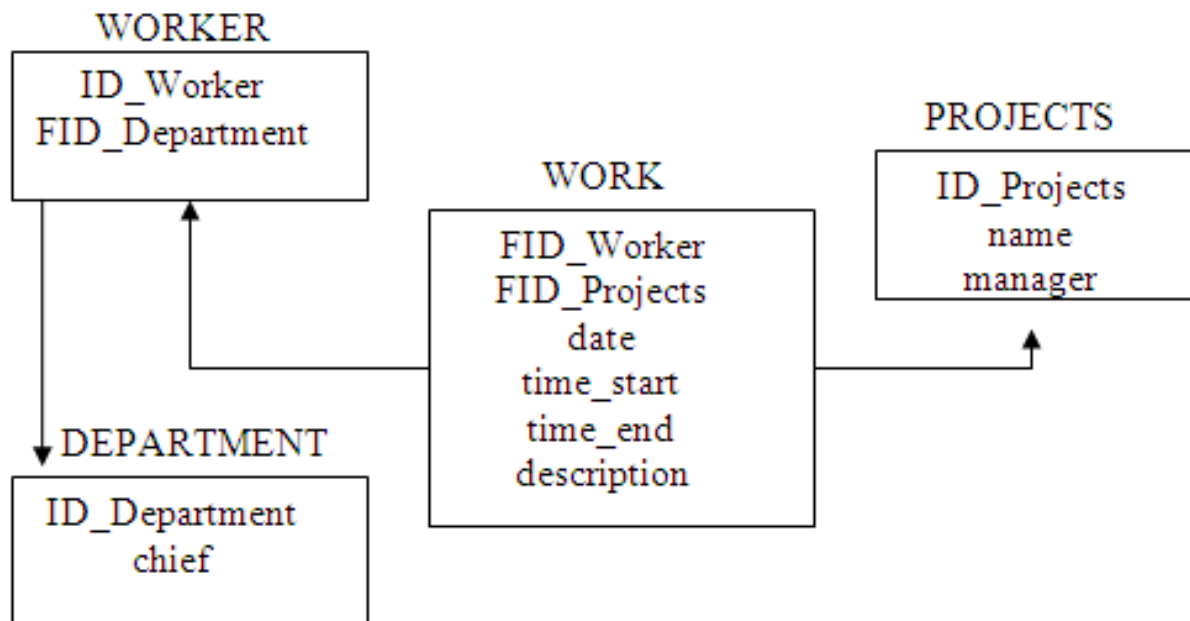


Рисунок 2.5 – Структура бази даних «Робота співробітників»

Варіант 4. Створити структуру бази даних для зберігання інформації про зміни чергувань у поліклініці (рисунок 2.6). Для медсестри задається дата чергування і зміна (перша, друга або третя), відділення, номери палат.

Сформувані запити і вивести результати:

- розклад чергувань вибраної медсестри;
- розклад чергувань за обраним відділенням;
- розклад чергувань за обраною зміною.

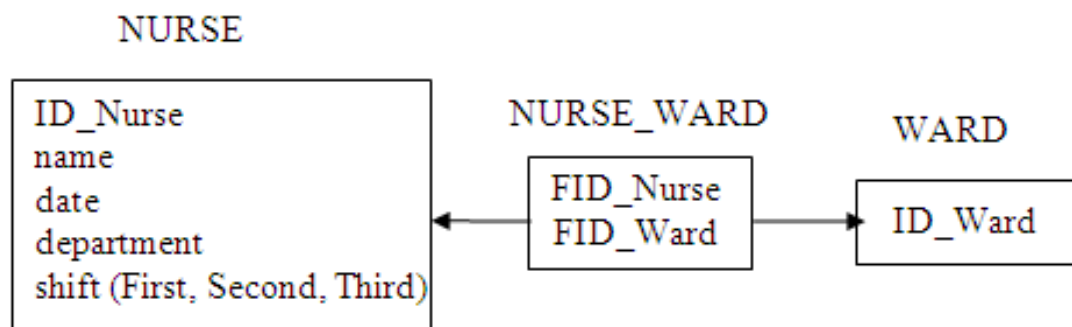


Рисунок 2.6 – Структура бази даних «Зміна чергувань»

Варіант 5. Створити структуру бази даних для зберігання додаткової інформації про фільмотеку (рисунок 2.7). Для фільму задається назва, жанр (може бути більш ніж один), рік виходу, країна, якість, тип носія (відеокасета, CD, DVD, BR). Для цифрових носіїв визначається роздільна здатність в пікселях і типи застосованих кодеків. Також для фільму може задаватися додаткова інформація, така як: продюсер, режисер і актори (довільна кількість).

Сформувані запити і вивести результати:

- список фільмів за обраною категорією;

- список фільмів з обраним актором;
- список фільмів за вказаний часовий інтервал.

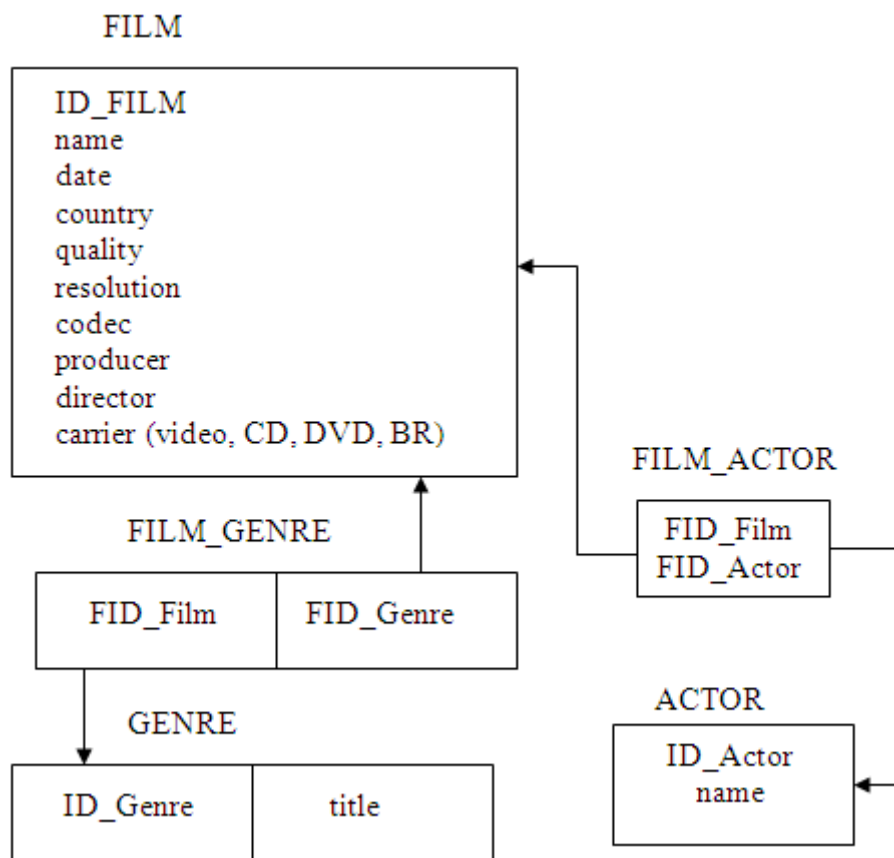


Рисунок 2.7 – Структура бази даних «Фільмотека»

Варіант 6. Створити структуру бази даних для зберігання інформації про товари в інтернет-магазині (Рисунок 2.8). Для товару задається назва, фірма-виробник, категорія товару (процесори, материнські плати і т.д.), ціна товару, кількість одиниць на складі.

Сформувати запити і вивести результати:

- товари обраного виробника;
- товари обраної категорії;
- товари у вибраному ціновому діапазоні.

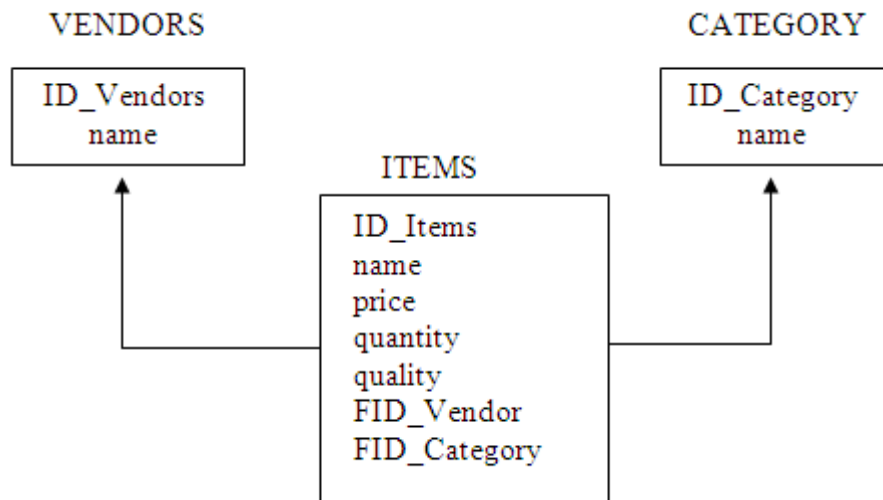


Рисунок 2.8 – Структура бази даних «Товари в магазині»

Варіант 7. Створити структуру бази даних для зберігання інформації про пункт прокату машин (рисунок 2.9). У базі даних задається назва машини, виробник, рік випуску, пробіг, стан, вартість оренди (у годину і за добу), час оренди автомобіля.

Сформувані запити і вивести результати:

- автомобілі зазначеного цінового діапазону;
- автомобілі обраного виробника;
- вільні автомобілі на обрану дату.

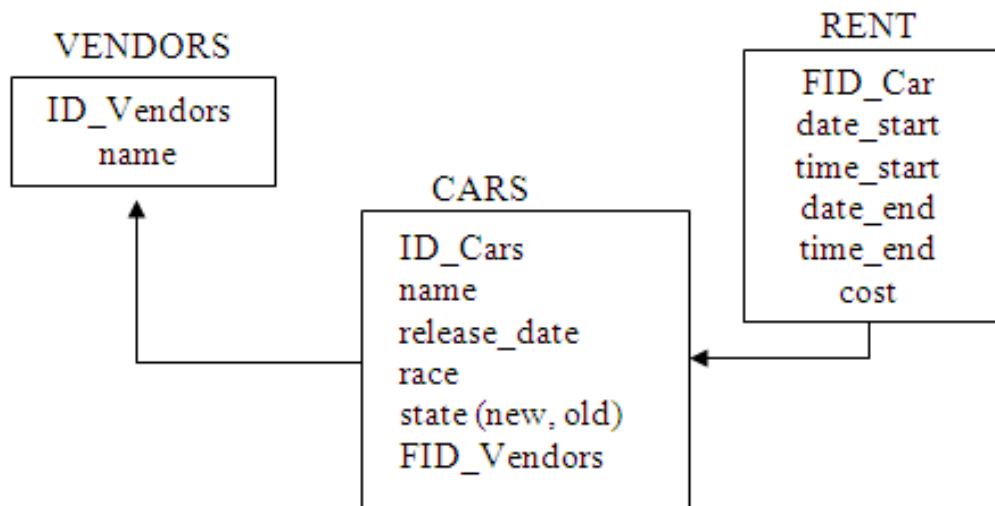


Рисунок 2.9 – Структура бази даних «Прокат машин»

Варіант 8. Створити структуру бази даних для зберігання інформації про результати футбольного чемпіонату (рисунок 2.10). Для кожної футбольної команди задається така інформація: назва, ліга, головний тренер. Для кожної гри задається дата проведення, команди учасниці, місце проведення, фінальний рахунок, футболісти що приймали участь.

Сформувати запити і вивести результати:

- таблиця чемпіонату обраної ліги;
- список ігор за вказаний часовий інтервал;
- список ігор вибраного футболіста.

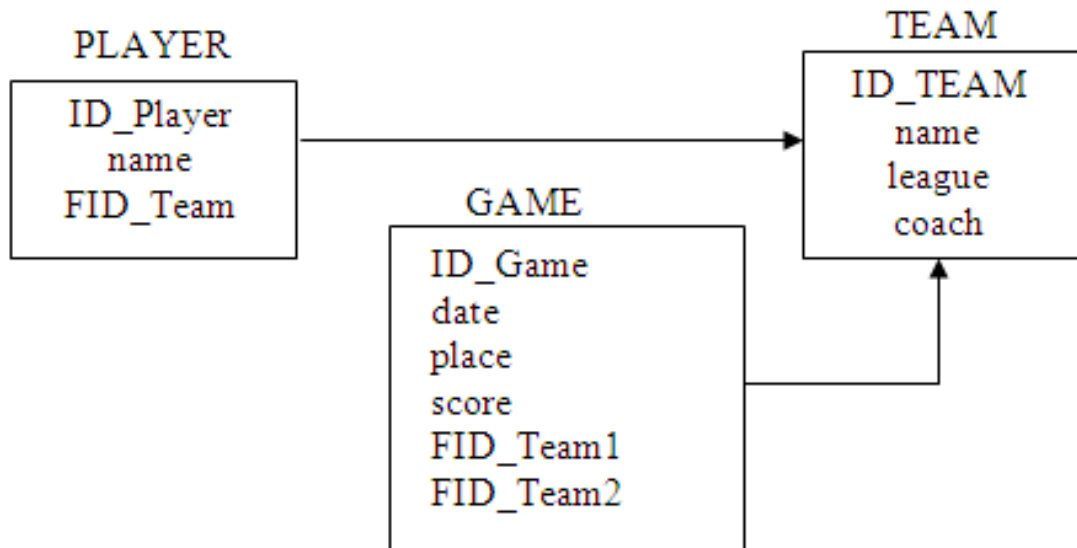


Рисунок 2.10 – Структура бази даних «Футбольний чемпіонат»

Варіант 9. Створити структуру бази даних для зберігання інформації про використання мережевого трафіку (рисунок 2.11). Для кожного клієнта задається логін, пароль, статичний IP-адресу машини клієнта, баланс рахунку. Для сеансів роботи задається час початку і закінчення сеансу роботи, кількість вхідного трафіку, кількість вихідного трафіку.

Сформувати запити і вивести результати:

- статистику роботи в мережі вибраного клієнта;
- статистику роботи в мережі за вказаний проміжок часу;
- вивести список клієнтів з негативним балансом рахунку.

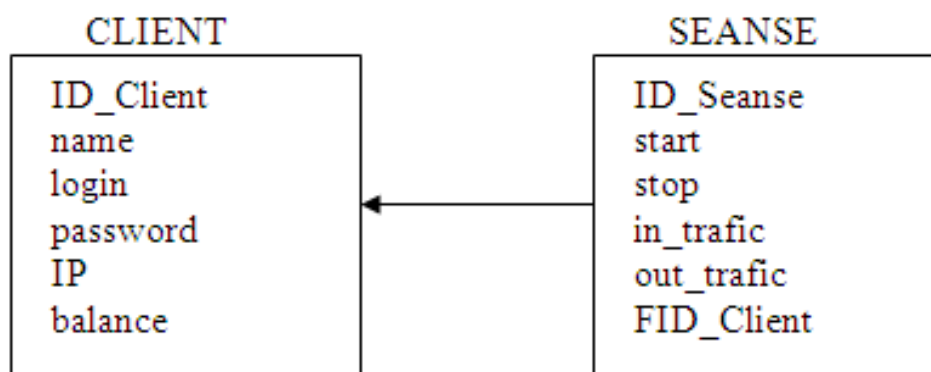


Рисунок 2.11 – Структура бази даних «Мережевий трафік»

Варіант 10. Створити структуру бази даних для зберігання інформації про комп'ютери організації (рисунок 2.12). Для кожного комп'ютера задається мережеве ім'я, тип центрального процесора, тип материнської плати, обсяг ОЗУ і НЖМД, тип монітора, встановлене програмне забезпечення, фірма-продавець, дата покупки, термін гарантії.

Сформулювати запити і вивести результати :

- список комп'ютерів із заданим типом центрального процесора;
- список комп'ютерів з встановленим ПЗ (назва ПЗ вибирається з переліку);
- список комп'ютерів з вичерпаним гарантійним терміном.

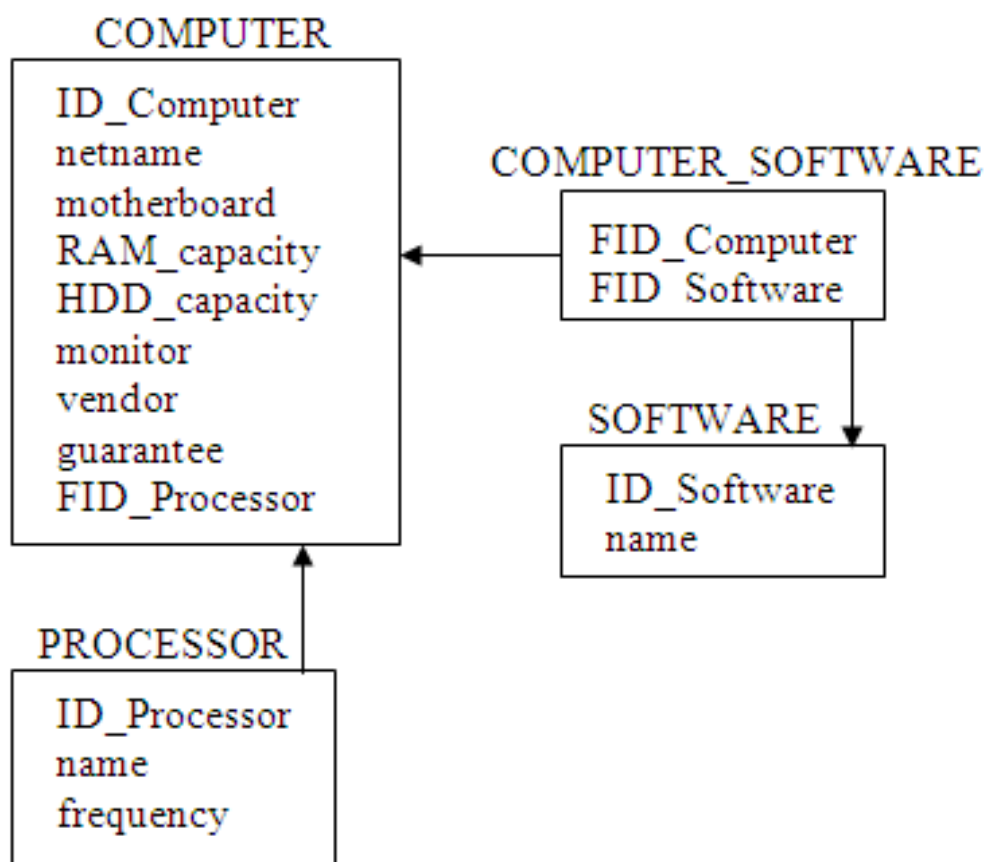


Рисунок 2.12 – Структура бази даних «Комп'ютери організації»

2.6 Зміст звіту

- мета лабораторної роботи;
- завдання на лабораторну роботу;
- ER-діаграма структури бази даних;
- дамп бази даних, сформований RHPMyAdmin;
- вихідні коди PHP - скриптів;
- результати виконання запитів у вигляді знімків екрана браузера;
- висновки з виконаної роботи.

2.7 Контрольні запитання і завдання

- 1) У чому перевага застосування інтерфейсу доступу до баз даних PDO?
- 2) Яким чином здійснюється керування підключеннями до бази даних?
- 3) Як виконується обробка помилок за допомогою розширення PDO?
- 4) Пояснити принципи використання процедур, що зберігаються, і звернення до них засобами PDO.
- 5) Для яких цілей призначені підготовлені запити?
- 6) Визначення транзакції, приклад використання?

3 ПРОГРАМНІ МЕХАНІЗМИ ШАБЛОНІЗАЦІЇ WEB-ДОДАТКІВ З ВИКОРИСТАННЯМ TWIG

3.1 Мета роботи

Вивчення синтаксису та принципів роботи з компіляційним механізмом шаблонів Twig.

3.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до виконання лабораторної роботи слід повторити основні концепції моделі об'єктно-орієнтованого програмування, а також шаблони проектування додатків [5]. Крім того, потрібно ознайомитися з базовим синтаксисом оголошення змінних, умов і циклів механізму шаблонів Twig, а також програмні інтерфейси відповідного класу [7-9].

3.3 Опис лабораторної установки

При виконанні лабораторної роботи використовується ПЕОМ під керуванням операційної системи Windows XP і старше, веб-сервер хампр 2.4.4, шаблонізатор Twig, середовище розробки Notepad++ 6.4.5.

3.4 Порядок виконання роботи і методичні вказівки з її виконання

Практика використання мови PHP для проектування складних і масштабованих інтернет-додатків вказує про необхідність відділення даних від представлення цих даних. Оскільки мова PHP є вбудовуваною і, фактично, змішує код виводу даних і код відображення даних. Це істотно ускладнює завдання як розробника інтернет-додатку, оскільки йому потрібно шукати вкраплення PHP-коду в досить великий HTML-сторінці, так і дизайнера сторінки. Фактично, навіть невелика зміна дизайну сторінок для великого корпоративного сайту, що не використовує відділення представлення від виведення даних, є суттєвою проблемою. Для вирішення даної задачі розроблено цілий ряд механізмів шаблонів, які дозволяють зберігати шаблон HTML-сторінки окремо від логіки інтернет-додатку, розділяючи файли на групи у відповідності з тими функціями, які вони виконують у додатку (рисунок 3.1). При цьому можна відносно легко змінювати і сам код логіки програми і дизайн сторінки.

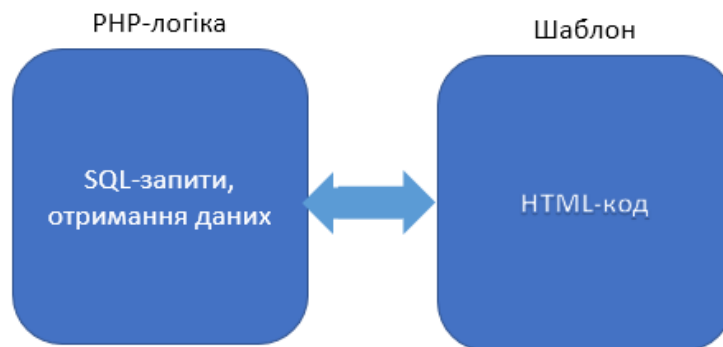


Рисунок 3.1 – Схема розробки додатку із застосуваннями клієнтських шаблонів

Twig - обробник шаблонів з відкритим вихідним кодом, що написаний на мові програмування PHP. Говорячи більш чітко, він надає один з інструментів, що дозволяють добитися відділення прикладної логіки і даних від уявлення. Одне з призначень Twig – це відділення логіки додатка від подання. Twig написаний Fabien Potencier, творцем фреймворка Symfony, та поширюється за новою ліцензією BSD.

Twig – сучасний шаблонізатор для PHP:

- Швидкий: компілює шаблони в оптимізований PHP код. Використання пам'яті порівнянно із звичайним PHP кодом зведено до мінімуму.

- Безпечний: має режим "пісочниці" для оцінки "ненадійного" коду в шаблонах. Це дозволяє використовувати Twig в додатках, де користувачі можуть міняти вміст шаблонів.

- Гнучкий: використовує гнучкі лексичний і граматичний аналізатори, що настроюються. Це дозволяє розробнику визначати свої теги і фільтри, створювати власний DSL(Domain Specific Language).

Переваги використання шаблонізатора Twig:

- Стислість. Мова PHP багатослівна і навіть занадто багатослівна, коли потрібно екранувати виведення даних:

```
echo $var;
```

```
echo htmlspecialchars($var, ENT_QUOTES, 'UTF-8');
```

- Twig має більш лаконічний синтаксис, який дозволяє легко читати шаблони:

```
{{ var }}
```

```
{{ var|escape }}
```

```
{{ var|e }}      {# скорочення для escape #}
```

- Орієнтований на шаблони синтаксис. Twig має скорочення для багатьох спільних патернів, наприклад, відображення тексту за замовчуванням, у випадку перебору порожнього масиву:

```
{% for user in users %}
```

```
    * {{ user.name }}
```

```
{% else %}
```

Користувачів немає.

```
{% endfor %}
```

- Повнофункціональний. Twig підтримує все, що вам треба для використання шаблонів: множинне спадкування, макроси, блоки, автоматичне екранування, користувальницькі фільтри, кешування і багато іншого:

```
{% extends "layout.html" %}
```

```
{% block content %}
```

- Легкий у вивченні. Синтаксис просто вивчити, тому що він оптимальний для верстальників, що дозволяє їм швидко виконувати свою роботу.

Звичайно, для PHP написана велика кількість шаблонізаторів. Але більшість з них написані під PHP4 і не використовують кращі практики розробки:

1) Розширюваність. Twig це гнучкий движок для будь-яких ваших потреб, навіть найскладніших. Завдяки відкритій архітектурі, можна визначати нові мовні конструкції (теги і фільтри) для створення свого власного DSL.

2) Юніт-тестування. Twig повністю покритий тестами. Бібліотека стабільна і готова до використання у великих проектах.

3) Документація. Twig повністю документований, вся документація доступна на сайті, і, звичайно, має повний опис API.

4) Безпека. Відносно безпеки, Twig має кілька абсолютно унікальних особливостей:

- Автоматичне екранування виводу. Для безпечного виведення даних, є можливість включити екранування як глобально, так і для окремих блоків:

```
{% autoescape on %}
```

```
{% var %}
```

```
{% var|safe %}    {# var не екранується #}
```

```
{% var|escape %}  {# var не екранується двічі #}
```

```
{% endautoescape %}
```

- Пісочниця. Twig дозволяє використовувати для будь-якого шаблону "пісочницю", де користувачі мають обмежений набір тегів, фільтрів і методів об'єктів, визначених розробником. Режим " пісочниці " може бути включений як глобально, так і локально, для певних шаблонів:

```
{{ include "user.html" sandboxed }}
```

5) Чисті повідомлення про помилки. Кожен раз, коли у вас виникають синтаксичні помилки в шаблоні, шаблонізатор виводить повідомлення про ім'я файлу з помилкою і номером рядка, який її викликав. Це дуже спрощує налагодження.

6) Швидкий. Одна з цілей створення даного шаблонізатор - зробити його настільки швидким, наскільки це можливо. Для досягнення максимальної швидкості роботи, Twig компілює шаблони в оптимізований PHP код. Використання пам'яті в порівнянні із звичайним PHP кодом зведено до мінімуму.

Twig для програмістів.

Twig використовує центральний об'єкт – environment (клас Twig_Environment) :

```
$twig = new Twig_Environment($loader, array('debug' => true));
```

Примірники цього класу використовуються для зберігання конфігурації і розширень, використовуються для завантаження шаблонів з файлової системи або інших місць.

Більшість додатків створює один об'єкт Twig_Environment при ініціалізації програми і використовує його для завантаження шаблонів. У деяких випадках, однак, корисно мати кілька середовищ спільно (поруч один з одним), якщо використовуються різні конфігурації.

Зверніть увагу, що другим параметром навколишнього середовища є масив опцій:

- debug: при установці в true, генеруються шаблони, що мають __toString() метод, який можна використовувати для відображення згенерованих вузлів (за замовчуванням false);

- charset: кодування, що використовується в шаблонах (за замовчуванням в UTF- 8);

- base_template_class: шаблон базового класу, який використовують для згенерованих шаблонів (за замовчуванням Twig_Template);

- cache: абсолютний шлях, де зберігаються скопійовані шаблони або false щоб відключити кешування (який за замовчуванням);

- auto_reload: при розробці з Twig, корисно повторно компілювати шаблон при зміні вихідного коду. Якщо не встановлено значення опції auto_reload, то вона буде визначена автоматично на основі змінної debug;

- strict_variables: якщо встановлено false, Twig буде за замовчуванням ігнорувати недійсні змінні (змінні чи атрибути / методи, які не існують) і замінювати їх значенням null. Коли встановлюється в true, Twig генерує виняток (за замовчуванням false);

- autoescape: якщо встановити true, автозбереження буде дозволено за замовчуванням для всіх шаблонів (за замовчуванням true). Починаючи з Twig 1.8, є можливість вибрати яку методику збереження використовувати (html, js, false для блокування). Починаючи з Twig 1.9, є можливість вибрати яку методику css, url, html_attr або зворотний виклик PHP, який бере шаблон "ім'я файлу" і повинен повернути методику збереження для використання, причому зворотний виклик не може бути назвою функції, щоб уникнути конфлікту (колізії) з вбудованими методами збереження;

- optimizations: прапор, який вказує, які оптимізації застосовувати (за замовчуванням -1 - всі оптимізації дозволені; встановіть його на 0 для відключення).

Twig може викидати винятки:

- Twig_Error: основний виняток для всіх помилок;

- Twig_Error_Syntax: викидається для того, щоб повідомити користувачеві, що є проблема з синтаксисом шаблону;
- Twig_Error_Runtime: викидається, коли виникає помилка під час виконання (наприклад, коли фільтр не існує);
- Twig_Error_Loader: викидається при виникненні помилки під час завантаження шаблону;
- Twig_Sandbox_SecurityError: викидається коли недозволений тег, фільтр, або метод викликається з ізолюваного шаблону.

Для завантаження шаблону з середовища вам просто потрібно викликати loadTemplate() метод, який повертає потім примірник Twig_Template:

```
$template = $twig->loadTemplate('index.html');
```

Щоб передати змінні в шаблон, викличте метод render():

```
echo $template->render(array('the' => 'variables', 'go' => 'here'));
```

Можна також завантажити і відтворити шаблон одночасно:

```
echo $twig->render('index.html', array('the' => 'variables', 'go' => 'here'));
```

Приклад повноцінного додатка, що написаний з застосуванням Twig і використовує змінні, призначені з PHP, складається з HTML-сторінки і логіки роботи в PHP - скрипті .

HTML-сторінка з тегами Twig (index.twig) показана нижче.

```
<!doctype html>
<html lang="ru-RU">
<head>
  <meta charset="utf-8" />
</head>
<title>User Info</title>
</head>
<body>
<h2>Account successfully created!</h2>
<p>Hello {{ name }}</p>
<p>Thank you for registering with us. Your account details are as follows:
</p>
<p style="margin-left: 10px">
Username: {{ username }} <br/>
Password: {{ password }}
</p>
<p>You've already been logged in, so go on in and have some fun!</p>
</body>
</html>
```

Логіка роботи в PHP-скрипті (index.php) може бути реалізована у вигляді, запропонованому нижче.

```
<?php
```

```
// активуємо автозавантажувач Twig
```

```

require_once 'vendor/autoload.php';
// вказує, де зберігаються шаблони
$loader = new Twig_Loader_Filesystem('templates');
//ініціалізуємо Twig
$twig = new Twig_Environment($loader, array(
    'cache' => false, // replace with 'cache' => 'templates_c' to enable com-
piled templates caching
    'charset' => 'utf-8'
));
// завантажуюмо шаблон і передаємо в шаблон змінні і значення, виво-
димо сформований вміст
echo $twig->render('index.twig', array(
    'name'=>'Clark
Kent','username'=>'ckent', 'password'=>'krypt0n1t'));

```

Twig для розробників шаблонів.

Шаблон - це просто текстовий файл, який може генерувати будь-який текстовий формат (HTML , XML , CSV , Latex , і т.д.) і не зобов'язаний мати особливого розширення (.html або .xml розширення підійдуть).

Шаблон містить змінні або вирази, які будуть замінюватися значеннями в процесі обчислення шаблону, і теги, які контролюють логіку шаблону.

Twig визначає три типи спеціальних синтаксичних конструкцій:

- { { ... } }: “надрукувати що-небудь”: відображає змінну або результат деякого виразу ;
- { % ... % } : “виконати що-небудь”: тег , який контролює логіку шаблону ; він використовується для виконання виразів , таких як цикли for .
- {# це коментар #} : є також третій тип синтаксичної конструкції для створення коментарів. Цей синтаксис може бути використаний як багато-строчного коментаря як PHP - аналог / * коментар * / .

Робота зі змінними.

Додаток передає змінні, з якими можна працювати в шаблоні. Змінні можуть мати атрибути або елементи, до яких є доступ, причому як виглядає мінлива визначається додатком, яка її надала. Можна використовувати оператор крапки (.), щоб отримати доступ до атрибута змінної (методи або властивості PHP - об'єкта або елементи PHP - масиву), або так званий індекс ([]) :

```

{{ foo.bar }}

{{ foo['bar'] }}

```

Коли атрибут містить спеціальні символи (такі як "-" , що буде інтерпретовано як оператор віднімання) , використовуйте функцію attribute замість доступу до атрибута змінної.

```

{# еквівалент не працюючого змінної foo.data - foo #}
{{ attribute(foo, 'data-foo') }}

```

Асоціативні масиви можуть бути використані в PHP - скрипті у вигляді, наведеному нижче:

```
$books=array(
    'title' => 'Harry Potter',
    'author' => 'Rowling J.K.',
    'publisher' => 'Scholsstic');
echo $template->render (array (
    'books' => $books ));
```

При цьому їх використання в шаблоні Twig може бути наступним:

```
<h2>Book details</h2>
<table>
<tr><td><strong> Title </strong></td>
<td> {{ books.title }} </td></tr>
<tr><td><strong> Author </ strong></td>
<td> {{ books.author }} </ </td></tr>
</table>
```

Надати значення змінним всередині блоків коду можна таким чином:

```
{% set foo = 'foo' %}
{% set foo = [1, 2] %}
{% set foo = {'foo': 'bar'} %}
```

Наступні змінні завжди доступні в шаблонах і є глобальними:

- `_self`: посилається на поточний шаблон;
- `_context`: посилається на поточний контекст;
- `_charset`: посилається на поточну кодування.

Фільтри.

Змінні можуть бути змінені за допомогою фільтрів. Фільтри відокремлюються від змінних за допомогою ріре-символа (`|`) і можуть мати додаткові аргументи в дужках. Можна об'єднувати кілька фільтрів, при цьому вихід одного фільтра направляється в наступний.

У Twig - шаблонах застосовуються наступні фільтри:

- `abs` возвращает абсолютне значення;
- `batch` повертає список із заданим числом елементів;
- `capitalize` робить перший символ рядка заголовним , а решта- малими;
- `convert_encoding` перетворить рядок з одного кодування в іншу;
- `date` формує дату в заданому форматі;
- `date_modify` змінює дату за заданим модифікатором;
- `default` повертає передане значення за замовчуванням , якщо значення не визначено або порожньо, в іншому випадку значення змінної;
- `escape` екранує рядок для висновку, підтримує різні настройки залежно від контексту шаблону;
- `first` повертає перший "елемент" послідовності , відображення або рядок;

- format форматує задану рядок , замінивши мітки (мітки пишуться в нотації sprintf);
- join повертає рядок , яка є конкатенацією елементів послідовності;
- json_encode повертає подання json рядка;
- keys повертає ключі масиву;
- last повертає останній "елемент" послідовності , відображення або рядок;
- length повертає кількість елементів послідовності або відображення , або довжину рядка;
- lower перетворює значення в нижньому регістрі;
- merge зливає два масиви разом;
- nl2br вставляє
 перед кожним переведенням рядків;
- number_format дозволяє формувати числа , можете змінювати кількість знаків після коми , десяткову точку і роздільник тисяч , використовуючи додаткові аргументи;
- raw відзначає значення як " безпечний" , що означає , що в блоці екранування ця змінна буде виведена що не екраніруемого , якщо raw є останнім фільтр застосовується до нього;
- replace форматує вхідну рядок , замінюючи мітки (мітки визначаються як ключ хеша);
- reverse змінює послідовність , відображення або рядок;
- round округлює числа з вказаною точністю;
- slice витягує шматок послідовності , відображення або рядок;
- sort сортує масив;
- split розбиває рядок за заданим разделителю і повертає масив рядків;
- striptags видаляє sgml / xml - теги і замінює множинні прогалини на 1 пробіл;
- title возвращает весь рядок в нижньому регістрі , перший символ в рядку буде заголовним;
- trim видаляє пробіли (або будь-які інші символи) з початку і кінця рядка;
- upper возвращает рядок у верхньому регістрі;
- url_encode кодує задану рядок як сегмент URL або масив як рядок запити.

Наступний приклад видаляє всі HTML-теги і title з name :

```
{{ name|striptags|title }}
```

Фільтри , які беруть аргументи мають круглі дужки навколо аргументів.

Приклад нижче приєднає список , розділений комою.

```
{{ list|join(',') }}
```

Щоб застосувати фільтр для секції в коді , оберніть його з тегом filter :

```
{% filter upper %}
```

Цей текст буде виводиться верхньому регістрі

```
{ % endfilter % }
```

Функції.

Можна викликати функції щоб генерувати контент. Функції можуть бути викликані за іменем з дужками і можуть мати аргументи.

Наприклад, функція `range` повертає список, що містить арифметичну прогресію цілих чисел :

```
{% for i in range(0, 3) %}  
  {{ i }},  
{% endfor %}
```

Підтримка для іменованих аргументів була додана в Twig 1.12:

```
{% for i in range(low=1, high=10, step=2) %}  
  {{ i }},
```

```
{% endfor %}
```

Використання іменованих аргументів допомагає зрозуміти значення змінних , переданих як аргументи.

```
{{ data|convert_encoding('UTF-8', 'iso-2022-jp') }}
```

```
{# або #}
```

```
{{ data|convert_encoding(from='iso-2022-jp', to='UTF-8') }}
```

Іменовані аргументи також дозволяють припустити деякі аргументи , для яких значення за замовчуванням залишається незмінним:

```
{# перший аргумент - це формат дати, який для глобальних дат не використовується #}
```

```
{{ "now"|date(null, "Europe/Paris") }}
```

```
{# або пропустити значення формату за допомогою іменованого аргументу для часового поясу #}
```

```
{{ "now"|date(timezone="Europe/Paris") }}
```

Можна використовувати позиційні та іменовані аргументи на одному виклику , і в цьому випадку позиційні аргументи повинні йти попереду іменованих аргументів.

```
{{ "now"|date("d/m/Y H:i", timezone="Europe/Paris") }}
```

Макроси.

Макроси можна порівняти з функціями в звичайних мовах програмування. Вони корисні для повторного використання HTML-фрагментів, що часто використовуються. Макрос визначається через `macro` теги. Невеликий приклад (згодом `forms.html`) макросу, який представлений у вигляді елемента форми:

```
{% macro input(name, value, type, size) %}  {% endmacro %}
```

Макрос може бути визначений в будь-якому шаблоні , і повинен бути "імпортовано " через тег `import` до використання :

```
{% import "forms.html" as forms %}  
{{ forms.input('username') }}
```

Для імпорту макросу , визначеного в поточному шаблоні можна використовувати:

```
{% import _self as forms %}
<p>{{ forms.input('post_genre') }}</p>
```

Крім того, можна імпортувати окремі імена макросів з шаблону в поточне простір імен за допомогою from тега або дати їм ім'я:

```
{% from 'forms.html' import input as input_field %}
```

За замовчуванням значення також може бути визначено для макроаргументов прямо у виклику макросу :

```
{% macro input(name, value = "", type = "text", size = 20) %}{% endmacro %}
```

Керуючі структури.

Керуюча структура обумовлена всередині блоків { % ... % } та належить до тих програмних засобів, які керують програмою - умови (if / elseif / else), for - цикли і блоки.

Наприклад, логіка роботи в PHP-скрипті може бути визначена таким чином :

```
$num=rand(0,30);
$div=($num % 2);
echo $template->render (array (
    'num' => $num,
    'div' => $div ));
```

А Twig - шаблон з використанням **умов if/elseif /else** в такому випадку буде виглядати так :

```
<html>
<head></head>
<body>
<h2>Odd or Even</h2>
{% if div==0 %}
{{ num }} is even.
{% else %}
{{ num }} is odd
</body>
</html>
```

Цикл for здійснює перебір послідовності, яка може бути масивом або об'єктом. Наприклад, виведемо список користувачів users :

```
<h1>Members</h1>
<ul>
    {% for user in users %}
        <li>{{ user.username|e }}</li>
    {% endfor %}
</ul>
```

Якщо цикл проходить по масиву чисел або букв, можна використовувати оператор " .. " :

```
{% for i in 0..10 %}  
  * {{ i }}  
{% endfor %}
```

У Twig є можливість використання змінних циклу:

- loop.index - поточна ітерація циклу (починаючи від 1);
- loop.index0 - поточна ітерація циклу (починаючи від 0);
- loop.revindex - кількість ітерація з кінця циклу (починаючи від 1);
- loop.first - кількість ітерація з кінця циклу (починаючи від 0);
- loop.last - виставляється в true, якщо поточна ітерація є першою;
- loop.length - кількість елементів у послідовності;
- loop.parent - батьківський контекст.

Приклад використання змінних циклу :

```
{% for user in users %}  
  {{ loop.index }} - {{ user.username }}  
{% endfor %}
```

На відміну від PHP, в даному випадку не можна перервати або продовжити цикл. Однак можна створювати правила проходження послідовностей під час ітерацій, що дозволяють пропускати або відбирати елементи за встановленими критеріями. У прикладі нижче неактивні користувачі пропускаються :

```
<ul>  
  {% for user in users if user.active %}  
    <li>{{ user.username|e }}</li>  
  {% endfor %}  
</ul>
```

Якщо ітерація не відбулася через те, що послідовність була порожня, є можливість здійснити заміну/підстановку блоку, використовуючи оператор else:

```
<ul>  
  {% for user in users %}  
    <li>{{ user.username|e }}</li>  
  {% else %}  
    <li><em>no user found</em></li>  
  {% endfor %}  
</ul>
```

Включення інших шаблонів.

Тег include використовується для включення шаблону і використаного контенту до поточного:

```
{% include 'sidebar.html' %}
```

За замовчуванням включені шаблони передаються в поточний контекст.

Контекст, який передається шаблону включає змінні, визначені в основному шаблоні:

```
{% for box in boxes % }
  {% include "render_box.html" % }
{% endfor % }
```

Включений шаблон `render_box.html` може отримати доступ до `box`.

Спадкування шаблонів.

Найбільш потужний засіб Twig це спадкування шаблонів. Воно дозволяє побудувати базовий "скелет" шаблону, який містить всі загальні елементи вашого сайту і визначає блоки, які дочірні шаблони можуть заміщати.

Наприклад, шаблон `base.twig`, показаний нижче, визначає базовий скелетон HTML - документа для простої сторінки з двома колонками:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{% block title %}Test Application{% endblock %}</title>
  </head>
  <body>
    <div id="sidebar">
      {% block sidebar % }
      <ul> <li><a href="/">Home</a></li>
        <li><a href="/blog">Blog</a></li> </ul>
      {% endblock % }
    </div>
    <div id="content">
      {% block body % }{% endblock % }
    </div>
  </body>
</html>
```

У цьому прикладі визначено три області `{% block %}` (`title`, `sidebar` і `body`). Кожен блок може бути перевизначений дочірнім шаблоном, інакше буде збережена початкова реалізація цих блоків. Цей шаблон може також бути відображений самостійно. У цьому випадку блоки `title`, `sidebar` і `body` будуть утримувати свої значення за замовчуванням.

Дочірній шаблон `index.twig` може виглядати наступним чином:

```
{% extends 'base.twig' % }
{% block title %}My cool blog posts{% endblock % }
{% block body % }
  {% for entry in blog_entries % }
    <h2>{{ entry.title }}</h2>
    <p>{{ entry.body }}</p>
```

```
{% endfor %}  
{% endblock %}
```

Ключем до спадкоємства шаблонів є тег `{ % extends % }`. Він повідомляє движку шаблонізатора, що необхідно спочатку виконати базовий шаблон, який налаштує загальну розмітку і визначить деяку кількість блоків. Після цього буде відображатися дочірній шаблон, який вказує, що блоки батька `title` і `body` будуть заміщатися аналогічними блоками нащадка.

Кешування шаблонів в Twig .

Всі завантажувачі шаблонів можуть кешувати скомпільовані шаблони в файлової системі для майбутнього використання :

```
$twig = new Twig_Environment($loader, array(  
    'cache' => 'templates_c',  
    'auto_reload' => true ));
```

Це набагато прискорює Twig, бо шаблони компілюються всього один раз. Зростання продуктивності навіть вище в порівнянні з PHP - акселераторами, такими як APC .

Другий аргумент середовища (опція кеша) - компіляція папки кеш де Twig зберігає кешовані шаблони для запобігання фази парсинга в наступних запитах. Цей кеш відрізняється від кеша, який є можливість додати в певних шаблонах, для чого можна використовувати будь-яку PHP кеш- бібліотеку.

3.5 Приклад виконання лабораторної роботи

Розглянемо структуру бази даних для зберігання додаткової інформації про фільмотеку. ER-діаграма бази даних зображена на рисунку 3.1.

Необхідно реалізувати спрощений веб-інтерфейс до бази даних, показаної на рисунку 3.1, з використанням шаблонів Smarty. Веб-інтерфейс містить заповнене поле введення, кнопку, після натискання якої відбувається вибірка даних за запитом з бази даних, і список, що випадає, для виведення результатів.

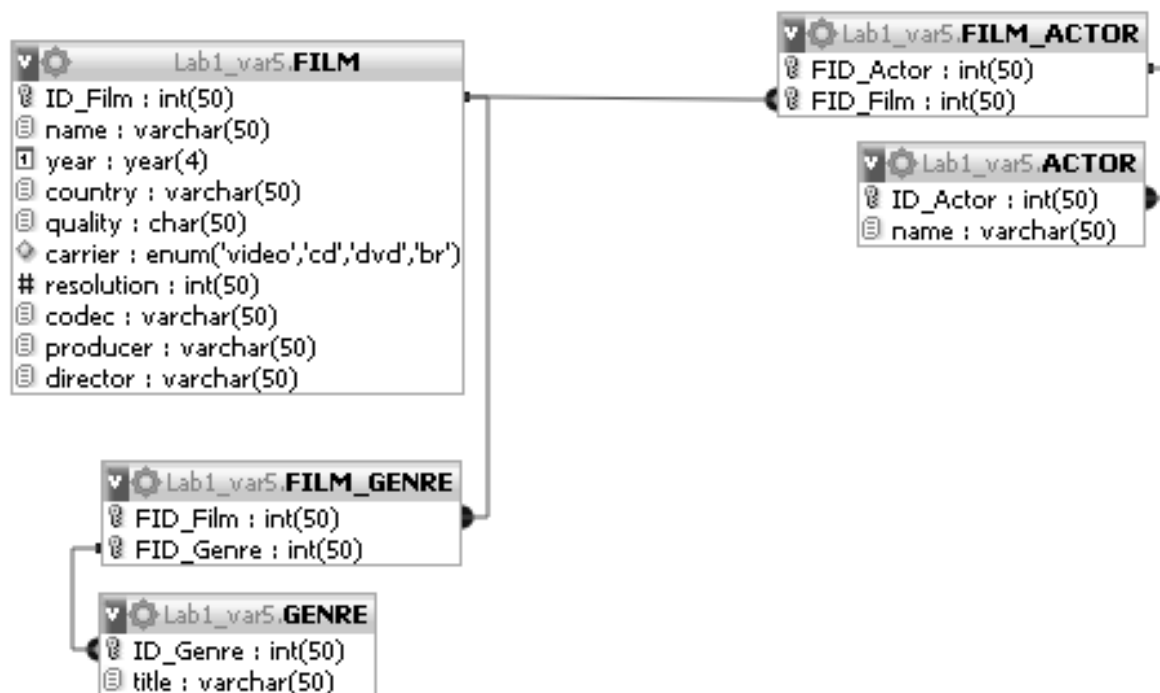


Рисунок 3.1 – ER-діаграма бази даних «Фільмотека»

На рисунках 3.2, 3.3, 3.4 представлено зміст таблиць FILM, FILM_GENRE, GENRE, вони беруть участь у підсумковому запиті для створення результуючої таблиці.

ID_Film	name	year	country	quality	carrier	resolution	codec	producer	director
1	Американец	2010	США	высокое	cd	230	кодек№1	Алан Бернон	Элизабет Авеллан
2	Призрак	2009	Канада	среднее	dvd	430	кодек№2	Робер Бенмусса	Роман Полански

Рисунок 3.2 – Зміст таблиці FILM

FID_Film	FID_Genre
1	1
2	2

Рисунок 3.3 – Зміст таблиці FILM_GENRE

ID_Genre	title
1	драма
2	комедия

Рисунок 3.4 – Зміст таблиці GENRE

Вихідний код PHP-скрипту, представлений нижче, дозволяє вивчити основні механізми передачі даних в шаблон.

```
<?php
require_once 'vendor/autoload.php';
```

```

$loader = new Twig_Loader_Filesystem('templates');
$twig = new Twig_Environment($loader, array(
    'cache' => false, // replace with 'cache' => 'templates_c' to enable com-
piled templates caching
    'charset' => 'utf-8'
));

// параметри з'єднання з базою даних
$dbh = new PDO('mysql:host=localhost;dbname=Lab1_var5', 'root', '');
$dbh->exec("SET CHARACTER SET 'utf8'");
error_reporting(E_ERROR ^ E_DEPRECATED);

if (isset($_POST['f1submit']))
{
    // формування результату
    $rth = $dbh->prepare('SELECT distinct a.name, country, quality, carrier
FROM FILM a, FILM_GENRE b, GENRE WHERE ID_Film=b.FID_Film and
ID_Genre=FID_Genre and title=:title');
    $rth->bindParam(':title', $_POST['post_genre']);
    $rth->execute();
    $result= $rth->fetchAll();
}
// передача значень у шаблон
echo $twig->render('extends.twig', array(
    'genres' => $dbh->query('select title from GENRE',
$dbh::FETCH_COLUMN, 0),
    'results' => $result ));
$dbh=null;

```

Вихідний код шаблону Twig є прикладом реалізації концепції поділу файлів на групи у відповідності з тими функціями, які вони виконують у додатку.

```

//index.twig
<!doctype html>
<html lang="ru-RU">
<head>
    {% include 'include.twig' %}
</head>
<body>
<H1> Film Database</H1>
{# форма пошуку за фільмами обраного жанру #}
<form action="index.php" method="post" name="form1">
    <h2>search by genre</h2>
    {% macro input(name, value = 'драма', type = 'text', size = 20) %}

```

```

        <input type="{{ type }}" name="{{ name }}" value="{{ value | e }}"
size="{{ size }}" />
    {% endmacro %}
{% import _self as forms %}
<p>{{ forms.input('post_genre') }}</p>
    <input type="submit" name="f1submit" value="search">
</form>

```

```

    {# список, що випадає, для виведення результатів пошуку#}
    {% block forms %} Вміст батьківського шаблону {% endblock forms %}
</body>
</html>

```

```

//include.twig
<meta charset="utf-8" />

```

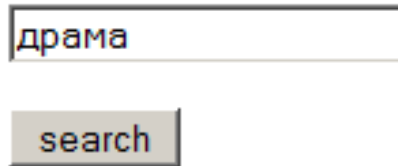
```

//extends.twig
{% extends "index.twig" %}
{% block forms %}
    {# список, що випадає, для виведення результатів пошуку#}
    <form action="index.php" method="post" name="form1">
        <h2>search result</h2>
        <select name="post_genres">
            {% for result in results %}
                <option value="{{ result[0] }}">{{ result[0] }}</option>
            {% endfor %}
        </select>
    </form>
    {{ parent() }}
    {% endblock forms %}

```

Film Database

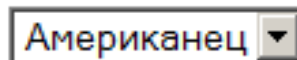
search by genre



драма

search

search result



Американец ▼

Содержимое родительского шаблона

Рисунок 3.5 – Результат виконання програми

3.6 Варіанти завдань

Розробити веб-інтерфейс до бази даних, аналогічний реалізованому в лабораторній роботі №2, з використанням механізму шаблонів Twig. Остаточний варіант завдання має бути реалізований із застосуванням різних методів вибірки даних для веб-інтерфейсу до бази даних, механізмів успадкування і підключення шаблонів.

3.7 Зміст звіту

Звіт з лабораторної роботи повинен містити:

- тему і мету роботи;
- завдання на лабораторну роботу;
- ER-діаграму структури бази даних (з лабораторної роботи № 1);
- вихідні коди PHP-скриптів і Twig-шаблонів;
- результати виконання скриптів у вигляді знімків екрана браузера;
- висновки з виконаної роботи.

3.8 Контрольні запитання і завдання

- 1) У чому недолік змішування коду й представлення?

- 2) Які функції виконують елементи архітектури Model-View-Controller?
- 3) Які основні відмінності шаблонів і каркасів?
- 4) У яких випадках виправданим є використання CMS (Content Management System) і CMF (Content Management Framework)?
- 5) Які відмінні особливості має механізм шаблонів Twig порівняно зі Smarty або HTML_Template_Flexy?
- 6) Які можливості для виведення складного вмісту надає дизайнерові сторінок шаблонізатор Smarty ?
- 7) Яким чином здійснюється автоматична фільтрація та екранування шаблонів Twig?
- 8) Як здійснюється налаштування кешування шаблонів Twig?
- 9) У чому перевага і відмінність використання підключення та успадкування шаблонів?
- 10) Для чого застосовуються макроси? Перерахуйте програмні механізми для імпорту макросів.

4 АСИНХРОННИЙ ОБМІН ДАНИМИ З СЕРВЕРОМ НА ОСНОВІ ТЕХНОЛОГІЇ AJAX

4.1 Мета роботи

Вивчити програмні механізми технології AJAX для динамічної відправки або завантаження даних з сервера.

4.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до виконання лабораторної роботи слід ознайомитися з особливостями синхронного та асинхронного методів передачі даних, вивчити формати обміну даними JSON і XML, а також повторити методи та властивості класу XMLHttpRequest [9].

4.3 Опис лабораторної установки

При виконанні лабораторної роботи використовується ПЕОМ під керуванням операційної системи Windows XP і старше, веб-сервер хатрр 2.4.4, середовище розробки Notepad++ 6.4.5.

4.4 Порядок виконання роботи і методичні вказівки з її виконання

При розробці web-додатків існує два способи обміну даними клієнтських додатків з сервером: синхронний та асинхронний. AJAX – аббревіатура, яка означає Asynchronous Javascript and XML, не є новою технологією, так як і Javascript, і XML існують вже досить тривалий час.

AJAX – підхід до побудови інтерактивних інтерфейсів користувача веб-додатків, що полягає в «фоновому обміні» даними браузера з веб-сервером. Асинхронний обмін даними з сервером базується на синтезі Javascript і XML. У результаті, при оновленні даних, веб-сторінка не перезавантажується повністю, і веб-додатки стають швидшими і зручнішими.

Однак при використанні AJAX розробнику необхідно створювати засоби, за допомогою яких користувач буде знати, що відбувається на сторінці. Це зазвичай реалізується за допомогою використання індикаторів завантаження, текстових повідомлень про те, що йде обмін даними з сервером.

AJAX базується на двох основних принципах:

- 1) використання технології динамічного звернення до сервера «на льоту», без перезавантаження всієї сторінки повністю, наприклад:
 - з використанням XMLHttpRequest (основний об'єкт);
 - через динамічне створення дочірніх фреймів;
 - через динамічне створіння тега <script>;
- 2) використання DHTML для динамічної зміни змісту сторінки.

Як формат передачі даних зазвичай використовуються JSON (JavaScript Object Notation) або XML.

До переваг AJAX можна віднести:

- Економія трафіку – використання AJAX дозволяє значно скоротити трафік при роботі з веб-додатком завдяки тому, що часто замість завантаження всієї сторінки достатньо завантажити лише частину, що змінена, як правило, досить невелику.

- Зменшення навантаження на сервер – AJAX дозволяє дещо знизити навантаження на сервер. Наприклад, на сторінці роботи з поштою, коли ви відзначаєте прочитані листи, серверу достатньо внести зміни в базу даних і відправити клієнтському скрипту повідомлення про успішне виконання операції без необхідності повторно створювати сторінку і передавати її клієнту.

- Прискорення реакції інтерфейсу – оскільки потрібно завантажити лише змінену частину, користувач бачить результат своїх дій швидше.

До недоліків AJAX можна віднести:

- Відсутність інтеграції зі стандартними інструментами браузера. Динамічно створювані сторінки не реєструються браузером в історії відвідин сторінок, тому не працює кнопка «Back», що надає користувачам можливість повернутися до переглянутих раніше сторінок, але існують скрипти, які можуть вирішити цю проблему. Іншим недоліком динамічної зміни контенту сторінки при постійному URL є неможливість збереження закладки на бажаний матеріал. Частково вирішити ці проблеми можна за допомогою динамічної зміни ідентифікатора фрагмента (частини URL після #), що дозволяють більшості браузерів.

- Динамічно завантажуваний вміст недоступний пошуковим машинам (якщо не перевіряти запит, звичайний він або XMLHttpRequest) . Пошукові машини не можуть виконувати JavaScript, тому розробники мають подбати про альтернативні способи доступу до вмісту сайту.

- Старі методи обліку статистики сайтів стають неактуальними. Багато сервісів статистики ведуть облік переглядів нових сторінок сайту. Для сайтів, сторінки яких використовують AJAX, така статистика втрачає актуальність.

- Ускладнення проекту. Перерозподіляється логіка обробки даних – відбувається виділення і часткове перенесення на бік клієнта процесів первинного форматування даних. Це ускладнює контроль цілісності форматів і типів. Кінцевий ефект технології може бути нівельований необґрунтованим зростанням витрат на кодування і управління проектом, а також ризиком зниження доступності сервісу для кінцевих користувачів.

- Потрібен увімкнений JavaScript в браузері.

Технологія AJAX використовує комбінацію:

- (X)HTML, CSS для представлення та стилізації інформації;

- DOM-модель, операції над якою виконуються javascript на боці

клієнта, щоб забезпечити динамічне відображення та взаємодію з інформацією;

– XMLHttpRequest для асинхронного обміну даними з веб-сервером. У деяких AJAX-фреймворк і в деяких ситуаціях, замість XMLHttpRequest використовується IFrame, SCRIPT-тег або інший аналогічний транспорт.

– Формат обміну даними – будь-який, не обов'язково XML (текст, список, JSON).

Об'єкт XMLHttpRequest.

Браузери надають засіб для виконання асинхронних запитів – об'єкт XMLHttpRequest. Об'єкт XMLHttpRequest – низькорівнева основа більшості AJAX-додатків. Використання його методів та властивостей допомагає писати програми на низькому рівні з мінімумом javascript-коду, а також зрозуміти, яким чином здійснюються операції всередині фреймворків.

У таблиці 4.1 представлені методи об'єкта XMLHttpRequest.

Таблиця 4.1 - Методи об'єкта XMLHttpRequest

Метод	Опис
abort()	Виклик цього методу обриває поточний запит. Для браузера Internet Explorer виклик abort() може не обривати з'єднання, а залишати його у підвішеному стані на деякий таймаут (20-30 секунд).
getAllResponseHeaders()	Повертає рядок з усіма HTTP-заголовками відповіді сервера.
getResponseHeader(headerName)	Повертає значення заголовка відповіді сервера з ім'ям headerName.
open(method, URL, async, userName, password)	Визначає опціональні параметри запиту: <ul style="list-style-type: none">– method – HTTP-метод. Як правило, використовується GET або POST;– URL – адреса запиту;– async – визначає режим запиту (при встановленні значення в true, задається асинхронний режим);– userName, password – дані для HTTP-авторизації.
send(content)	Відсилає запит на сервер. Аргумент – тіло запиту. Наприклад, для GET-запиту тіла немає, тому використовується send (null), а для POST-запитів тіло містить параметри запиту.

Продовження таблиці 4.1

Метод	Опис
setRequestHeader(name, value)	Встановлює заголовок name запиту зі значенням value. Якщо заголовок з таким name вже є – він замінюється. Приклад: xmlhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
overrideMimeType(mimeType)	Дозволяє вказати mime-type документа, якщо сервер його не передав або передав неправильно. Метод відсутній у Internet Explorer! Деякі версії деяких браузерів Mozilla НЕ будуть коректно працювати, якщо відповідь сервера не містить mime-заголовка XML. Щоб вирішити цю проблему, ви можете використовувати виклики додаткових методів для перевизначення заголовка отриманого від сервера, у випадку, якщо він є відмінним від text/xml. Приклад: httpRequest = new XMLHttpRequest(); httpRequest.overrideMimeType('text / xml');

У таблиці 4.2 представлені властивості класу XMLHttpRequest.

Таблиця 4.2 - Властивості класу XMLHttpRequest

Властивість	Опис
onreadystatechange	Встановлює обробник події, яка відбувається при кожній зміні стану об'єкта.
readyState	Повертає поточний стан об'єкта (0 – неініціалізованих, 1 – відкритий, 2 – відправка даних, 3 – отримання даних, 4 – дані завантажені)
responseText	Текст відповіді на запит. Повний текст відповіді можливо отримати тільки при readyState, що встановлений у стан 4.
responseXML	Текст відповіді на запит у вигляді XML (при readyState = 4), який потім можливо розпарсити за допомогою DOM. Щоб браузер коректно розібрав відповідь сервера в responseXML, в заголовку повинен бути Content-Type: text/xml.

Продовження таблиці 4.2

Властивість	Опис
status	Повертає HTTP-статус у вигляді числа (404 – «Not Found», 200 – «OK» і т. д.). Запити протоколів FTP, FILE не повертають статусу, тому нормальним для них є status = 0.
statusText	Повертає статус у вигляді рядка (наприклад, «Not Found», «OK» і т.д.).

4.5 Приклад використання технології AJAX

Будь-який додаток, що побудован за технологією AJAX, складається з двох взаємодіючих між собою частин: клієнтської та серверної. Серверна частина – це сценарій, який запускається на сервері у відповідь на той чи інший GET/POST запит користувача. Клієнтська частина системи – деякий код на мові програмування Javascript, що виконується безпосередньо в браузері користувача. Він приймає дані, згенеровані серверною частиною, обробляє їх і відображає у відведений для цього області сторінки. Загальна схема етапів виконання AJAX-запиту показана на рисунку 4.1.



Рисунок 4.1 – Етапи виконання AJAX-запроса

Основним об'єктом, за допомогою якого йде звернення клієнтських запитів до сервера, є XMLHttpRequest. Ініціалізація даного об'єкта відрізняється в різних браузерах. Розглянемо приклад кросбраузерності створення об'єкту XMLHttpRequest.

```

<script type="text/javascript">
// глобальная переменная для хранения обработчика запросов
var ajax;
InitAjax();
function InitAjax()
{
// используем структуру try..catch для попытки создать обработчик запросов
XMLHTTP
try
  
```

```

    {
    // пробуємо створити компонент XMLHttpRequest для IE старих версій
        ajax = new ActiveXObject("Microsoft.XMLHTTP");
    }
    catch (e)
    {
    // якщо не вийшло створити компонент XMLHttpRequest для IE пробуємо следу-
    // ючий і т.д.
        try
        {
        // пробуємо створити компонент XMLHttpRequest для IE версій 6 і ви-
ше
            ajax = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e)
        {
            try
            {
            // пробуємо створити компонент XMLHttpRequest для Mozilla, і інших
                ajax = new XMLHttpRequest();
            }
            catch (e)
            {
                ajax = 0;
            }
        }
    }
}
</script>

```

Результат функції `createRequestObject` повинен посилатися на коректний об'єкт `XMLHttpRequest`, незалежно від використовуваного користувачем браузера.

У першому прикладі пропонується завдання динамічного завантаження даних для формування списку, що випадає. У цьому прикладі використовується властивість `responseText` класу `XMLHttpRequest` як доступ до текстового вмісту.

У клієнтської частини програми з бази даних формуються елементи першого списку, другий список залишається порожнім і заповнюється на основі асинхронної передачі даних з боку сервера.

```

<!DOCTYPE html >
<html xmlns="http://www.w3.org/1999/xhtml">
<head>

```

```

    <meta http-equiv="Content-Type" content="text/html; charset=windows-
1251" />
    <meta http-equiv="Pragma" content="no-cache" />
    <title>Аjax-приклад</title>
</head>
<body>
    <b>Завдання №1</b>
    <form name="form1" method="get">
        <select id="select1" name="name1">
            <option value="0">SQL: Повний посібник</option>
            <option value="1">3 історії культури середніх віків та Від-
родження </option>
            <option value="2">Щоденний журнал</option>
        </select>
        <input type="button" name="form1submit" value="Пошук" on-
click="javascript: gets2();" />
        <br><br>
        <select id="select2" >
            <option>no data</option>
        </select>
    </form>
</body>
</html>

```

У скриптовій частини визначається метод (функція gets2()), що реагує на натискання кнопки. Ця функція виконує пошук елемента в списку select1 в об'єктній моделі документа і викликати функцію open, яка створює з'єднання. Для даного випадку в GET-параметрі запиту передається ідентифікатор обраної книги.

```

<script type="text/javascript">
// функція, яка буде запитувати вміст нової сторінки
function gets2()
{
    if(!ajax)
    {
        alert("Аjax не ініціалізований ");
        return;
    }
    var s1val = document.getElementById("select1").value;
    ajax.onreadystatechange = UpdateSelect2; //визначення обробника
/* відкриття з'єднання із зазначенням типу запиту (GET або POST), URL сер-
верної частини, прапора асинхронного режиму і імені та пароля користувача
(якщо необхідно) */

```

```

        ajax.open("GET", "get.php?select1="+s1val, true); //формування за-
питу
        ajax.send(null); // безпосередньо відправка запиту на сервер
    }
</script>

```

Крім того визначається функція UpdateSelect2(), яка буде обробляти (вставляти повернутий текст в потрібну область сторінки).

```

<script type="text/javascript">
function UpdateSelect2()
{
    if(ajax.readyState == 4) // якщо статус – виконано
    {
        if(ajax.status == 200) // якщо немає помилок
        {
            var divBody = document.getElementById('select2');
            divBody.innerHTML = ajax.responseText;
        }
        else alert(ajax.status + " - " + ajax.statusText);
        ajax.abort();
    }
}
</script>

```

У серверній частині реалізований механізм формування другого списку, що випадає, на основі обраного значення першого списку.

```

<?php
$хid = $_GET['select1'];
switch ($хid)
{
    case "0":
        echo '<option>BHV</option><option>10000
</option><option>10-15-XX-44</option>';
        break;
    case "1":
        echo '<option>Science</option><option>1000
</option><option>44-78-106-X </option>';
        break;
    case "2":
        echo '<option>BHV</option><option>150</option>';
        break;
} ?>

```

У другому прикладі пропонується завдання динамічного завантаження даних для блочного елемента. Використовується властивість `responseXML` класу `XMLHttpRequest` для отримання відповіді на запит у вигляді XML.

У клієнтської частини програми з бази даних додані три порожніх блокових тега `<div>`

```
<div id="book"></div>
<div id="publisher"></div>
<div id="quantity"></div>
```

Перевизначимо метод `UpdatePage()`, вказавши іншу властивість `responseXML` для обробки відповіді від сервера у вигляді формату XML і розпарсим кожен елемент отриманої XML відповіді.

```
function UpdatePage()
{
    if(ajax.readyState == 4) // якщо статус – виконано
    {
        if(ajax.status == 200) // якщо немає помилок
        {
            xmlDoc=ajax.responseXML;
            document.getElementById("book").innerHTML =
            xmlDoc.getElementsByTagName("book")[0].childNodes[0].nodeValue;
            document.getElementById("publisher").innerHTML =
            xmlDoc.getElementsByTagName("publisher")[0].childNodes[0].nodeValue;
            document.getElementById("quantity").innerHTML =
            xmlDoc.getElementsByTagName("quantity")[0].childNodes[0].nodeValue;
        }
        else
        {
            alert(ajax.status + " - " + ajax.statusText);
            ajax.abort();
        }
    }
}
```

У серверної частині необхідно вказати коректний тип документа і реалізувати механізм формування XML файлу, який буде переданий клієнту для обробки.

```
<?php
header('Content-Type: text/xml');
header("Cache-Control: no-cache, must-revalidate");
$id = $_GET['select1'];
echo '<?xml version="1.0" encoding="utf8" ?>';
```



```

echo "<row>";
echo "<book>BOOK from XML: ".$id."</book>";
echo "<publisher>PUBLISHER from XML: BHV </publisher>";
echo "<quantity>QUANTITY from XML: 10000</quantity>";
echo "</row>";    ?>

```

У третьому прикладі використовується властивість `responseText` класу `XMLHttpRequest` для отримання відповіді на запит у вигляді JSON. Перевизначити метод `UpdatePage()`, здійснивши парсинг отриманої відповіді від сервера у вигляді JSON.

```

function UpdatePage()
{
    if(ajax.readyState == 4) // якщо статус - виконано
    {
        if(ajax.status == 200) // якщо немає помилок
        {
            var res = JSON.parse(ajax.responseText);
            document.getElementById("book").innerHTML = res.book;
            document.getElementById("publisher").innerHTML =
res.publisher;
            document.getElementById("quantity").innerHTML =
res.quantity;
        }
        else
        {
            alert(ajax.status + " - " + ajax.statusText);
            ajax.abort();
        }
    }
}

```

У серверної частини програми необхідно вказати коректний тип документа і реалізувати механізм формування JSON файлу, який буде переданий клієнту для обробки.

```

<?php
    header('Content-Type: application/json');
    header("Cache-Control: no-cache, must-revalidate");
    $id = $_GET['select1'];
    $data = array('book' => $id, 'publisher' => 'Smit', 'quantity' => 1000);
    echo json_encode($data); ?>

```

У всіх браузерях, крім ІЕ, це можна зробити за допомогою властивості innerHTML. Для досягнення кросбраузерності в ІЕ елементи повинні додаватися один за одним.

4.6 Варіанти завдань

Для варіантів завдань, зазначених у лабораторній роботі № 2-3, додати динамічне завантаження необхідних даних з сервера за технологією AJAX.

Варіанти 1, 4, 7, 10 здійснюють передачу даних на сервер з використанням XML як протоколу обміну даними між сервером і клієнтом.

Варіанти 2, 5, 8 здійснюють передачу даних на сервер з використанням JSON як протокол обміну даними між сервером і клієнтом.

Варіанти 3, 6, 9 здійснюють передачу даних на сервер з використанням текстового формату.

4.7 Зміст звіту

Звіт з лабораторної роботи повинен містити:

- тему и мету роботи;
- завдання на лабораторну роботу;
- вихідні коди PHP- і AJAX-скриптів;
- результати виконання запитів у вигляді знімків екрану браузера;
- висновки з роботи.

4.8 Контрольні запитання і завдання

1) Опишіть ключові концепції Web2.0, приведіть зміни, внесені Web2.0 в порівнянні з Web1.0 (web.com).

2) Перерахуйте переваги і недоліки використання технології AJAX.

3) Вкажіть можливості, які надаються технологією AJAX.

4) Які зміни обробки даних на клієнтському боці введені для підтримки AJAX?

5) Якими об'єктами і методами підтримується методологія AJAX в JavaScript?

6) Які реалізації бібліотек підтримки AJAX існують?

5 ПОВНОДУПЛЕКСНИЙ ОБМІН ДАНИМИ МІЖ БРАУЗЕРОМ ТА ВЕБ-СЕРВЕРОМ НА ОСНОВІ ТЕХНОЛОГІЇ WEBSOCKET

5.1 Мета роботи

Дослідження клієнтських і серверних програмних засобів для обміну повідомленнями між браузером і веб-сервером в режимі реального часу.

5.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до виконання лабораторної роботи слід ознайомитися з механізмом обміну повідомленнями на основі протоколу WebSocket, а також з подіями і методами колективного обміну повідомленнями на боках клієнта і сервера бібліотеки socket.io [10,11].

5.3 Опис лабораторної установки

При виконанні лабораторної роботи використовується ПЕОМ під керуванням операційної системи Windows XP і старше, серверна платформа NODE.js, пакет socket.io, середовище розробки Notepad++ 6.4.5.

5.4 Порядок виконання роботи і методичні вказівки з її виконання

WebSocket - протокол повнодулексного зв'язку поверх TCP - з'єднання, призначений для обміну повідомленнями між браузером і веб-сервером в режимі реального часу. Організація такого роду тісної взаємодії між браузером і веб-сервером дає можливість створювати інтернет-додатки нового рівня, наприклад, ігри реального часу та інші веб-додатки з інтенсивним обміном даними, що є вимогливим до швидкості обміну і каналу.

Стандартний протокол HTTP обмежений моделлю запит-відповідь: клієнт посилає запит HTTP і очікує від нього HTTP-відповідь. Таким чином, сервер не може повідомити що-небудь клієнту до тих пір, поки клієнт його «не попросить». Такого роду програми обмежені мережевою затримкою і необхідністю створення всього документа заново в момент переходу за посиланням.

Завдяки наданню стандартного способу для відправки вмісту сервером браузеру без додаткового запиту клієнта у звичній схемі «запит URL - відповідь», між браузером і сервером може відбуватися двосторонній (двонаправлений) обмін повідомленнями одночасно, поки з'єднання відкрито.

Протокол WebSocket надає значно менші накладні витрати за рахунок постійно відкритого каналу в порівнянні з повторюваними зверненнями до

сервера для відстеження змін. Раніше такі функції були доступні тільки за допомогою технології плагінів типу Flash. Крім того, обмін інформацією йде через TCP-порт 80, що є великою перевагою для тих середовищ, які блокують нестандартні підключення до інтернету за допомогою брандмауера.

Розглянемо схему обміну повідомленнями між клієнтом і веб-сервером за протоколом WebSocket.

Коли браузер направляє запит за URL-адресою веб-сокета, сервер відправляє назад заголовки, які завершують обмін підтвердженнями на основі протоколу WebSocket. Квитування WebSocket у відповідь має починатися строго з рядка HTTP/1.1 101 WebSocket Protocol Handshake. Фактично, порядок і зміст заголовків "рукописання" визначаються більш строго, ніж для заголовків HTTP. Після завершення квитування клієнт і сервер можуть у будь-який момент почати обмін повідомленнями. Кожне з'єднання представляється на сервері екземпляром об'єкта WebSocket Connection.

Метод send об'єкта WebSocket Connection перетворить рядок повідомлення до вигляду, що відповідає протоколу WebSocket. Межі кадру позначаються байтами 0x00 і 0xFF, між якими міститься рядок повідомлення в кодуванні UTF-8: 0x00, < рядок в кодуванні UTF-8 >, 0xFF. Для відправки рядка можна використовувати дані, у тому числі формату XML і JSON. Отримуючи повідомлення від сервера, браузер виконує функцію callback. Коефіцієнт корисної дії такого протоколу наближається до 95%, оскільки немає необхідності пересилати кілька кілобайт заголовків, що особливо помітно якщо робити частий обмін невеликими блоками даних.

Готовий веб-додаток, реалізований з використанням протоколу WebSocket, являє собою клієнт-серверну реалізацію.

Тому, перш ніж почати використовувати протокол WebSocket, потрібно створити сервер, що підтримує веб-сокети. Найбільш популярним є серверний JavaScript-фреймворк node.js, на основі якого було створено кілька серверів WebSocket. Причому node.js - подієво-орієнтована модель і добре розвинені функції callback в javascript.

Бібліотека Socket.io забезпечує взаємодію в режимі реального часу між сервером node.js і клієнтами, забезпечуючи вбудоване мультиплексування, горизонтальну масштабованість, автоматичне кодування / декодування JSON і більше.

Конфігурація socket.io.

Socket.io можна налаштувати за допомогою методів configure, set, enable і disable, а також при встановленні з'єднання.

Наступні опції можуть бути налаштовані **на боці сервера**:

– heartbeats (за замовчуванням увімкнено) - чи використовується режим heartbeats для перевірки стану з'єднання socket.io;

- transports - транспортує за замовчуванням websocket, htmlfile, xhr-polling, jsonp-polling;
- log level (за замовчуванням дорівнює 3) - дані, що виводяться реєстратору: 0 - помилка, 1 - попередження, 2 - інформація, 3 - налагодження;
- close timeout (за замовчуванням 60 секунд) - тайм-аут для клієнта, впродовж якого у разі закриття з'єднання є можливість повторного його відкриття. Це значення посилають клієнту після успішного рукостискання;
- heartbeat timeout (за замовчуванням 60 секунд) - тайм-аут для клієнта, впродовж якого він повинен відправити нове значення heartbeat на сервер. Це значення посилають клієнту після успішного рукостискання.

Наступні опції можуть бути налаштовані **на боці клієнта**:

- connect timeout (за замовчуванням 10000 мс) - затримка перед спробою підключення до сервера з використанням іншого транспорту;
- reconnect (за замовчуванням увімкнена) - повторне підключення у випадку, якщо socket.io виявить розрив зв'язку або тайм-аут;
- reconnection delay (за замовчуванням 500 мс) - затримка перед початком відновлення з'єднання;
- maxReconnectionAttempts - максимальна кількість спроб перепідключення.

Вибір транспорту передачі інформації.

За допомогою пакету socket.io можливо з легкістю створювати крос-браузерні real-time додатки. Легкий і зручний рівень абстракції в socket.io досягається за рахунок використання різних транспортів передачі інформації на сервер і з сервера в браузер. Причому технологія вибирається абсолютно прозоро і для клієнта, і для сервера.

Транспорти вибираються у наступній послідовності:

- WebSocket;
- Adobe Flash Socket;
- AJAX multipart streaming;
- AJAX long polling;
- Iframe(тільки в IE);
- JSONP Polling.

Якщо браузер підтримує WebSockets, буде використовуватися саме він. Для інших браузерів буде забезпечений fallback до флешових сокетів, а якщо і цих немає - до звичайного XHR з long polling.

Socket.io підтримує такі десктопні браузери: Internet Explorer 5.5+, Safari 3+, Google Chrome 4+, Firefox 3+, Opera 10.61+. А також наступні мобільні браузери: iPhone Safari, iPad Safari, iPad Safari, Android WebKit, WebOs WebKit.

Нижче наведені приклади вибору **на боці сервера** транспорту протоколу WebSocket або декількох протоколів з використанням методу `socket.set (key, value)`.

```
// підключаємо модуль для створення сервера
// і ставимо на прослуховування 80-порт
var io = require('socket.io').listen(80);

//Спосіб 1
/// відключаємо показ повного логу
io.set('log level', 1);
// обмежуємо транспорт тільки протоколом WebSocket
io.set('transports', ['websocket']);

//Спосіб 2
io.configure(function () {
  io.set('transports', ['websocket', 'flashsocket', 'xhr-polling']);
});
//Спосіб 3
io.configure('development', function () {
  io.set('transports', ['websocket', 'xhr-polling']);
});
```

Отримання з'єднання WebSocket.

Нижче наведені приклади використання клієнтської і серверної бібліотеки `socket.io` для отримання з'єднання відповідно.

Серверний бік.

Підключення WebSocket на веб-сервері здійснюється за допомогою наступної функції:

```
var socket = require('socket.io').listen(80, {
  // опції можна описати тут
});
```

Приклад використання серверної бібліотеки `socket.io` (`server.js`) показаний нижче.

```
var port = 8080;
// відкриваємо з'єднання
// з встановленою опцією дозволених методів транспорту
var io = require('socket.io').listen(port, { 'transports': ['websocket'] });
io.sockets.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' });
});
```

```
    socket.on('my other event', function messageReceived(data) {      con-
sole.log(data); });
});
```

Клієнтський бік.

Підключення WebSocket на клієнті здійснюється за допомогою наступної функції:

```
var socket = io.connect('http://server.com', {
// опції можна описати тут
});
```

Приклад використання клієнтської бібліотеки socket.io (client.html) показаний нижче.

```
<!DOCTYPE html>
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<!-- Підключаємо бібліотеку, саме таким способом: запитуємо її з боку
сервера, де вона динамічно генерується. -->
<script src="http://localhost:8080/socket.io/socket.io.js"></script>
<script>
//адрес сервера
var serverURL = 'http://localhost:8080';
var socket = io.connect(serverURL, { 'connect timeout': 5000 });
    socket.on('news', function (data) {
        console.log(data);
        socket.emit('my other event', { my: 'data' });
    });
</script>
</head>
</html>
```

Події встановлення підключення, перепідключення і закриття з'єднання.

Розглянемо події на боці клієнта і сервера, які надсилаються у разі встановлення підключення, перепідключення і закриття з'єднання.

На боці сервера, припускаючи, `var io = require('socket.io')`:

- `io.sockets.on('connection', function(socket) {})` - початкове з'єднання від клієнта. Аргумент `socket` повинен бути використаний в подальшій комунікації з клієнтом;
- `socket.on('disconnect', function() {})` - подія роз'єднання спрацьовує у всіх випадках, коли з'єднання клієнт-сервер закрито. Спрацьовує у випадках бажаного, небажаного, мобільного чи не мобільного, клієнтського та серверного відключення. Не існує події відновлення зв'язку. Ви повинні використовувати подію "connection" для відновлення керованості.

На боці клієнта, припускаючи, `socket = io.connect (host, options)`:

1) у разі першого підключення:

- `socket.on('connect', function () {})` – подія "connect" надсилається, у разі якщо сокет успішно підключений;
- `socket.on('connecting', function () {})` - подія "connecting" надсилається, коли сокет намагається підключитися до сервера;

2) при миттєвій втраті з'єднання:

- `socket.on('disconnect', function () {})` – подія "disconnect" надсилається, у разі якщо сокети відключені;
- `socket.on('reconnecting', function () {})` - подія "reconnecting" надсилається при спробі сокета відновити зв'язок з сервером;
- `socket.on('connect_failed', function () {})` - подія "connect_failed" надсилається, у разі якщо `socket.io` не може встановити з'єднання з сервером і не має інших варіантів транспорту;
- `socket.on('reconnect_failed', function () {})` – подія "reconnect_failed" надсилається, у разі якщо `socket.io` вдається відновити робочий зв'язок після того як підключення було припинено;
- `socket.on('reconnect', function () {})` - подія "reconnect" надсилається, у разі якщо `socket.io` успішно повторно підключився до сервера;

3) при повній втраті з'єднання:

- `socket.on('disconnect', function () {})` - подія "disconnect" надсилається, у разі якщо сокети відключені;
- `socket.on('reconnecting', function () {})` - подія "reconnecting" надсилається, у разі якщо сокет намагається відновити зв'язок з сервером.

Закрити з'єднання є можливість у будь-якої зі сторін, як сервера та/або браузера, оскільки в підсумку існує тільки одне з'єднання.

При виникненні помилки можна скористатися подією на боці клієнта: `socket.on ('error', function () {})` - подія "error" надсилається, у разі якщо виникає помилка, яка не може бути оброблена іншими типами подій.

Надсилення даних до клієнта.

Метод `socket.send` виконує передачу повідомлення `text` на основі базової події `"message"`:

```
socket.send(text).
```

У `socket.io v0.6` метод `socket.send` буде автоматично конвертувати об'єкт в JSON, наприклад, відправлення даних клієнтові буде відповідати `socket.send ({a: 'b'})`.

Ця властивість одночасно є і перевагою при передачі даних, і в той же час створює проблеми, оскільки JSON не тільки шифрує об'єкти, але також і рядки, числа, і т.д. Це в свою чергу призводить до втрат продуктивності, пов'язані з кодуванням/декодуванням JSON. API стане прозорішим, якщо явно буде вказано, що буде передаватися JSON.

У `socket.io v0.7` використовується прапор `json` для відправки повідомлення в форматі JSON:

```
socket.json.send(data[, callback]).
```

Передача клієнтам повідомлення у форматі JSON дозволяє генерувати на клієнті легко змінювані дані меншого об'єму, подання яких не залежить від сервера.

Відправка повідомлення для всіх.

На сервері існує можливість вибору «поточного» клієнта за допомогою `socket`, так і вибору всіх підключених клієнтів за допомогою `io.sockets`.

Таким чином, якщо ви хочете відправити повідомлення будь-кому ви можете посилатися на `io.sockets`:

```
io.sockets.send('повідомлення');  
io.sockets.emit('подія');
```

Відправка непостійних повідомлень.

У разі якщо певний клієнт не буде готовий отримати повідомлення (через мережеву повільність або інших проблем, або тому що він з'єднаний за допомогою довгого опитування і знаходиться в середині циклу відповіді запити), тоді певні повідомлення можуть бути відкинуті.

У цьому випадку, є можливість відправки непостійних повідомлень на боці сервера:

```
var io = require('socket.io').listen(80);  
io.sockets.on('connection', function (socket) {  
  setInterval(function () {  
    socket.volatile.emit('bieber tweet', tweet); }, 100);  
});
```

Відправка подій.

Тепер ви можете надсилати та отримувати події користувача між браузером і сервером за допомогою:

```
socket.emit('custom event'[, arguments][, callback]);
```

Аргументи для події автоматично шифруються у формат даних JSON.

Приклад відправки на клієнтський сокет поточного запиту:

```
socket.emit('message', 'this is a test');
```

Приклад відправки на клієнтський сокет даних формату JSON:

```
socket.emit('whereami', { 'location': loc })
```

Отримання подій.

При виникненні події event методу socket.on (event, callback) виконується виклик функції зворотнього виклику callback.

На клієнті, припускаючи, socket = io.connect (host, options):

- socket.on('message', function (message, callback) { }) - подія "message" виконується у разі, якщо отримано повідомлення, надіслане з socket.send. Повідомлення «message» означає, що відправлено повідомлення, і функція зворотного виклику є додатковою функцією підтвердження;

- socket.on('anything', function(data, callback) { }) - подія "anything" може бути будь-якою призначеною для користувача подією, крім резервних. Аргумент data і функція зворотного виклику можуть бути використані для відправлення відповіді.

На боці сервера, припускаючи, var io = require('socket.io'):

- socket.on('message', function(message, callback) { }) - подія "message" виконується у разі, якщо отримано повідомлення, надіслане з socket.send. Повідомлення "message" означає, що відправлено повідомлення, і функція зворотного виклику є додатковою функцією підтвердження;

- socket.on('anything', function(data) { }) - подія "anything" може бути будь-якою подією, крім резервних.

Приклад отримання події:

```
socket.on('whereami', function(loc){ console.log('I\'m in ' + loc + '!'); })
```

Зберігання даних, пов'язаних з клієнтом.

Іноді необхідно зберігати дані, пов'язані з клієнтом, що необхідно на час сеансу.

У прикладі нижче показано як реалізувати зберігання даних **на боці сервера**.

```
var io = require('socket.io').listen(80);

io.sockets.on('connection', function (socket) {
  socket.on('set nickname', function (name) {
    socket.set('nickname', name, function () { socket.emit('ready'); });
  });

  socket.on('msg', function () {
    socket.get('nickname', function (err, name) {
      console.log('Chat message by ', name);
    });
  });
});
```

І на боці клієнта відповідно.

```
<script>
var socket = io.connect('http://localhost');

socket.on('connect', function () {
  socket.emit('set nickname', prompt('What is your nickname?'));
  socket.on('ready', function () {
    console.log('Connected !');
    socket.emit('msg', prompt('What is your message?'));
  });
});
</script>
```

Широкомовна передача повідомлень.

У socket.io v0.7.0 є можливість відправити повідомлення з окремого сокету до іншої частини сокетів, використовуючи прапор broadcast:

```
socket.broadcast.send('повідомлення');
socket.broadcast.emit('подія[, arguments]);
```

Приклад відправки повідомлення всім клієнтам, крім поточного:

```
socket.broadcast.emit('message', "this is a test");
```

Відправлення та передача повідомлень окремому сокету.

Крім того, є в наявності засоби вибору конкретно взятого клієнта із заданим ідентифікатором ID:

```
io.sockets.socket(< id>).send(моє повідомлення)  
io.sockets.socket(< id>).emit(ім'я події[, arguments])
```

Приклад відправки повідомлення окремим socketid на боці сервера:
`io.sockets.socket(socketid).emit('message', 'for your eyes only');`

На боці клієнта socketid отримують за допомогою:
`var io = io.connect('localhost');
io.on('connect', function () {
 console.log(this.socket.id);
});`

5.5 Варіанти завдань

Завдання №1.

Створити, виконати і переконатися в працездатності клієнтської і серверної частин програми «Чат», реалізованої з використанням пакету socket.io. Додати функціональність згідно із завданням 2.

Завдання №2.

Реалізувати систему реального часу, що складається з клієнтської і серверної частин.

Остаточний варіант завдання повинен бути протестований на швидкість передачі даних з використанням протоколу WebSocket за допомогою інструменту веб-розробника браузера Google Chrome.

Варіант 1.

Диспетчер таксі.

Клієнтська частина надає диспетчеру, що працює віддалено, інтерфейс для заповнення даних про клієнта (ПІБ, контактний телефон, адреса, дата і час подачі таксі, адресу призначення, вартість), після натискання кнопки відбувається передача замовлення на сервер. Клієнтська частина також отримує від сервера таблицю поточних замовлень, причому частота оновлення даних задається диспетчером. Сервер генерує таблицю поточних замовлень від усіх диспетчерів із заданим інтервалом.

Варіант 2.

Моніторинг транспортних засобів.

Клієнтська частина надає диспетчеру, що працює віддалено, можливість вибору водія, що працює на цій зміні, кнопку для накладання штрафу і відображає на екрані таблицю переміщень транспортного засобу в перебігу дня, завдяки розташованому в ньому GPS-трекеру (ідентифікатор машини, ПІБ водія, вимірювання широти і довготи проміжних точок маршруту, швид-

кість руху, кількість накопичених штрафів). У разі порушення заданого маршруту диспетчер накладає на водія штраф. Сервер генерує таблицю переміщень транспортних засобів для всіх диспетчерів з урахуванням накопичених штрафів (у разі перевищення кількості штрафів рівному п'яти, виводиться повідомлення про звільнення водія).

Варіант 3.

Онлайн-гра «Управління марсоходом».

Клієнтська частина надає можливість вибору марсохода, кількості мін, розкиданих на полі, розмірності поля (поле квадратне) і поточних координат марсохода, а також отримує від сервера дані про місцезнаходження міни. Сервер генерує дані про поточне місцезнаходження міни, а також у разі попадання на міну передає дані про руйнування марсоходу.

Варіант 4.

Система виробничо-екологічного моніторингу.

Клієнтська частина надає можливість вибору санітарно-захисної зони і частоти оновлення даних, а також отримує від сервера дані вимірів стаціонарних і мобільних постів екологічного моніторингу (метеорологічного контролю, хімічного контролю, пиломаси, аналізаторів води та ґрунту). Сервер генерує дані дистанційного зондування вибраної санітарно-захисної зони з заданим інтервалом.

Варіант 5.

Біржа.

Клієнтська частина надає брокеру, що працює віддалено, інтерфейс для подачі заявки клієнта (компанія, назва товару, ціна на одиницю товару, кількість товару, покупка або продаж), після натискання кнопки відбувається передача замовлення на сервер. Клієнтська частина також отримує від сервера таблицю поточних замовлень, причому частота оновлення даних задається брокером. Сервер генерує таблицю поточних замовлень від усіх диспетчерів із заданим інтервалом.

Варіант 6.

Система підтримки життєдіяльності.

Клієнтська частина надає можливість вибору об'єкта для дослідження і частоти оновлення даних, а також отримує від сервера дані від датчиків середовища (хімічний склад, температура, вологість, випромінювання) і відображає їх користувачеві. Сервер генерує дані датчиків із заданим інтервалом, а також збирає дані про хід дослідження об'єктів (повідомлення про загрозу життю чи ні) всіх поточних експериментів для виведення загальної таблиці.

Варіант 7.

Моніторинг погоди.

Клієнтська частина надає можливість вибору населеного пункту і частоти оновлення даних, а також отримує від сервера дані про погодні умови (поточна температура, наявність опадів, рівень тиску, швидкість вітру) та відображає їх користувачеві. Сервер генерує дані про погодні умови вибраного міста з заданим інтервалом .

Варіант 8.

Навчальна система «Організм людини»

Клієнтська частина надає можливість вибору органу людини та її стану, а також отримує від сервера дані про загальні фізіологічні показники організму (загальна кількість лейкоцитів, тромбоцитів, еритроцитів, швидкість осідання еритроцитів, кількість білків, що виділяються гормони) і відображає їх учню. На підставі даних учень може заповнити форму зі своїми даними і з висновками про протікання захворювання і лікування. Сервер генерує дані про фізіологічні показники організму із заданим інтервалом, а також збирає дані заповнених форм усіх учнів для виведення загальної таблиці.

5.6 Зміст звіту

Звіт з лабораторної роботи повинен містити:

- тему і мету роботи;
- завдання на лабораторну роботу;
- вихідні коди серверного та клієнтського JS-скриптів;
- результати виконання запитів у вигляді знімків екрана браузера;
- висновки з роботи.

5.7 Контрольні запитання і завдання

1) Порівняйте роботу з'єднання за допомогою HTTP Long polling з протоколом WebSocket .

2) Опишіть процес створення з'єднання WebSocket на боках клієнта і сервера, а також його закриття.

3) Перерахуйте клієнтські і серверні події, що виконуються при створенні з'єднання або повторному підключенні.

4) Яким чином здійснюється обмін даними між клієнтом і сервером із застосуванням методів пакета socket.io? Наведіть приклади.

5) Яким чином існує можливість відправки відповіді всім клієнтам крім відправника?

6) Перерахуйте методи для відправки та отримання даних формату JSON.

7) Поясніть механізм створення кімнат.

ПЕРЕЛІК ПОСИЛАНЬ

1. Дронов В.А. HTML5, CSS3 и Web 2.0 Разработка современных Web-приложений / В.А. Дронов. – СПб:БХВ-Петербург, 2011. – 416с.
2. Хоган Б. HTML5 и CSS3 Веб-разработка по стандартам нового поколения/ Б. Хоган. – СПб: Питер, 2012. – 272 с.
3. Зольников Д.С. PHP5 / Д.С. Зольников. – М.:ИТ Пресс, 2007. – 256 с.
4. Кузнецов М.В. PHP5 на примерах / М.В. Кузнецов, И.В. Симдянов, С.В. Голышев. – СПб: БХВ-Петербург, 2005. – 576с.
5. Ловэйн П. Объектно-ориентированное программирование на PHP5 / Питер Ловэйн; пер. с англ. А.А. Слинкина. – М.: ИТ Пресс, 2007. – 224с.
6. Гутманс Э. PHP5 Профессиональное программирование/ Э. Гутманс, С. Баккен, Д. Ретанс. – СПб: Символ-Плюс, 2006. -704с.
7. Зервас Квентин Web 2.0 создание приложений на PHP / Квентин Зервас. – Москва: ООО «И.Д. Вильямс», 2010. – 544с.
8. The Twig Book. – SensioLabs, 2014. – 158 с.
9. XTwig [Электронный ресурс]. – Режим доступа: <http://x-twig.ru/> – Загл. с экрана.
10. Котеров Д.В. PHP5/ Д.В. Котеров, А.Ф. Костарев. – 2-е изд., перераб. и доп. – СПб:БХВ-Петербург, 2008. – 1104 с.
11. Introducing of Socket.io v.9 [Электронный ресурс]. – Режим доступа: <http://socket.io/> – Загл. с экрана.
12. NODE.js v0.10.21 [Электронный ресурс]. – Режим доступа: <http://nodejs.org/> – Загл. с экрана.

Електронне навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт

з дисципліни
"INTERNET ТЕХНОЛОГІЇ"
Частина 2

для студентів денної та заочної форм навчання
напряму 6.050102 - «Комп'ютерна інженерія»

Упорядники: ЛЕБЬОДКІНА Алла Юріївна,
САРАНЧА Сергій Миколайович,

Відповідальний випусковий О.Г. Руденко

Авторська редакція