

Fullstack-розробка web-застосунку за допомогою MySQL, PHP, HTML, CSS та JavaScript

ЗМІСТ

1 СТВОРЕННЯ БАЗИ ДАНИХ ЗАСОБАМИ СУБД MYSQL. РОБОТА З БАЗОЮ ДАНИХ MYSQL У МОВІ PHP	4
1.1 Підготовка до виконання роботи	4
1.2 Створення бази даних	4
1.2.1 Запуск і налаштування XAMPP	4
1.2.2 Робота з MySQL в phpMyAdmin	7
1.2.3 Створення облікових записів користувачів	11
1.2.4 Створення тригерів для таблиць	13
1.2.5 Створення збережених процедур і функцій	16
1.3 Робота з базою даних в мові PHP	20
1.3.1 Функції для роботи з базою даних	20
1.3.2 Патерн проектування Repository	23
1.3.3 Патерн проектування Service Layer	28
1.4 Вимоги до звіту	32
1.5 Питання для самоперевірки	32
2 СТВОРЕННЯ WEB-ЗАСТОСУНКІВ ЗА ДОПОМОГОЮ ФРЕЙМВОРКА BOOTSTRAP. ВИКОРИСТАННЯ ТЕХНОЛОГІЇ AJAX ДЛЯ АСИНХРОННОГО ОБМІНУ ДАНИМИ З WEB-СЕРВЕРОМ	34
2.1 Підготовка до виконання роботи	34
2.2 Знайомство з фреймворком Bootstrap	34
2.3 Створення web-сторінок	39
2.3.1 Створення сторінки входу в додаток	39
2.3.2 Робота з одними сеансами	40

2.3.3 Створення сторінки роботи з даними	44
2.4 Асинхронний обмін даними з web-сервером	49
2.4.1 Знайомство з технологією AJAX.....	49
2.4.2 Використання AJAX на сторінці входу в додаток.....	51
2.5 Вимоги до звіту	54
2.6 Питання для самоперевірки	55

1 СТВОРЕННЯ БАЗИ ДАНИХ ЗАСОБАМИ СУБД MYSQL. РОБОТА З БАЗОЮ ДАНИХ MYSQL У МОВІ PHP

1.1 Підготовка до виконання роботи

1. Створити нову гілку в системі управління версіями Git і назвати її storage. Перебуваючи на створеній гілці, в робочому каталозі (наприклад, D:\GIT_PRACTICE) створити підкаталог, в якому буде виконуватися вся подальша робота (наприклад, D:\GIT_PRACTICE\db).

2. Завантажити та встановити XAMPP – cross-platform збірку web-сервера, що містить HTTP-сервер Apache, реляційну систему управління базами даних MySQL, інтерпретатор мови PHP і велику кількість додаткових бібліотек, що дозволяють запустити повноцінний web-сервер.

1.2 Створення бази даних

1.2.1 Запуск і налаштування XAMPP

Після того, як сервер XAMPP буде встановлено, необхідно запустити панель управління сервером і переконатися у відсутності будь-яких помилок (рисунк 1.1):

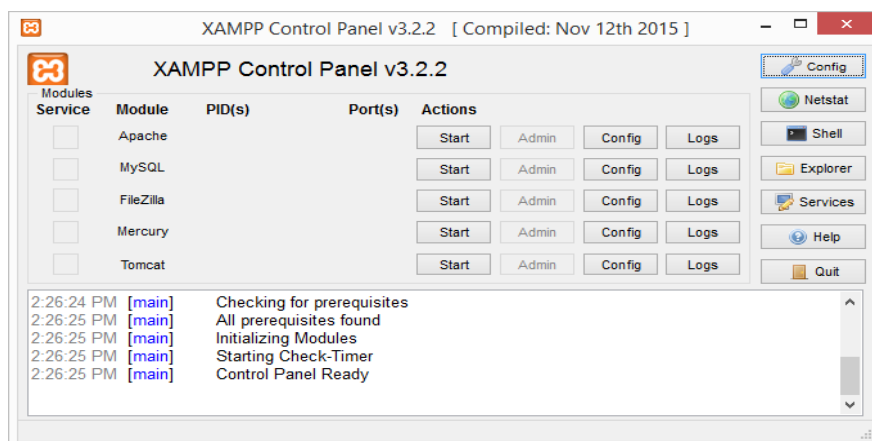


Рисунок 1.1

Більшість помилок, що виникають при запуску XAMPP або його модулів (Apache, MySQL і т.д.), пов'язані з тим, що використовувані порти вже зайняті якимись програмами або службами.

Для вирішення даної проблеми можна вимкнути програми і/або служби, що займають необхідні порти, або переналаштувати їх. Іншим же шляхом вирішення проблеми є налагодження портів служб XAMPP. Для цього потрібно відкрити вікно Config -> Service and Port Settings і вказати вільні порти для використання різними службами XAMPP (рисунок 1.2).

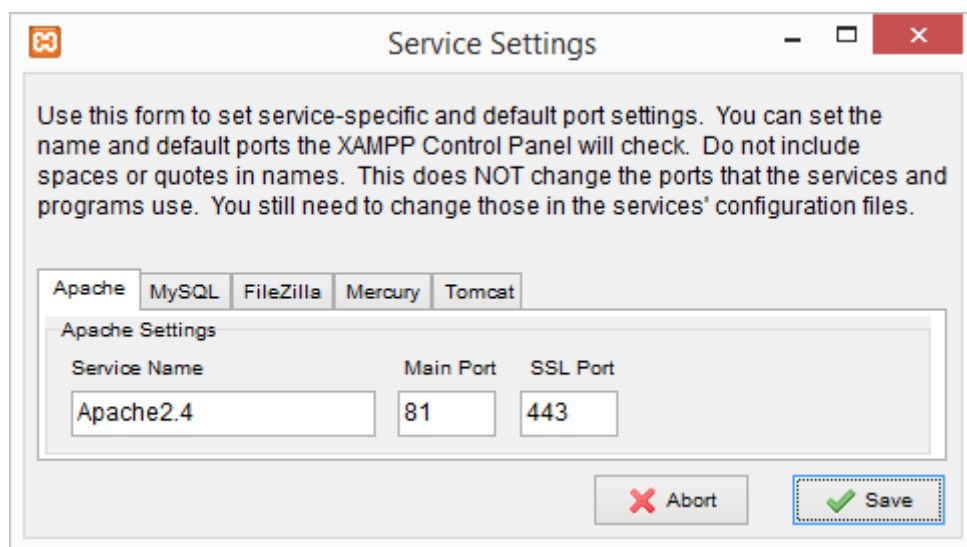


Рисунок 1.2

Крім того, може знадобитися налаштувати прив'язку Apache до певного порту вручну. Для цього необхідно вибрати дію Config для модуля Apache, вибрати пункт Apache (httpd.conf) і замінити номер порту в рядку (рисунок 1.3):

```
Listen XX # де XX – порт, встановлений за замовчуванням
```

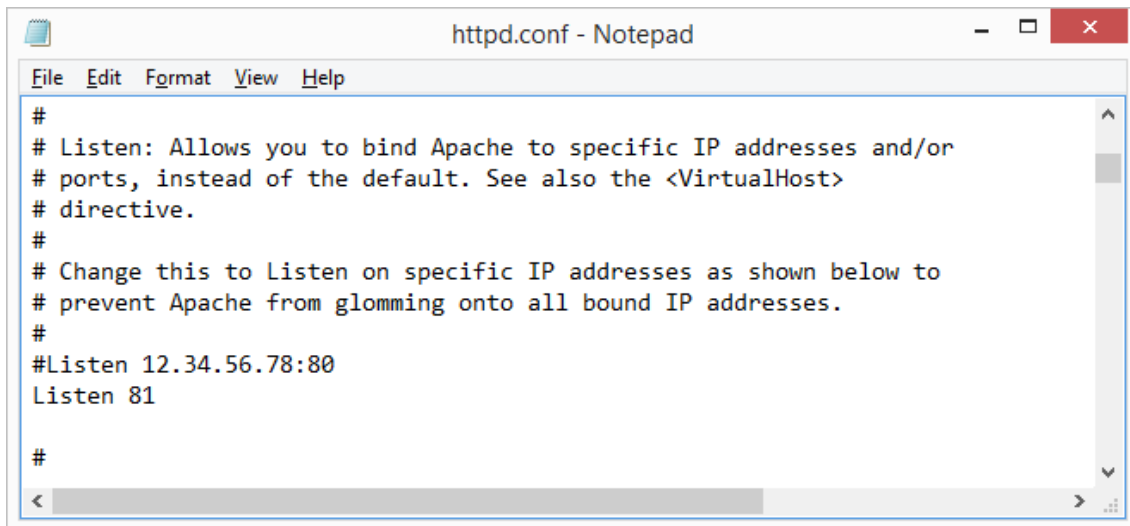


Рисунок 1.3

Після того, як проблеми будуть вирішені, необхідно запустити модулі Apache і MySQL. У разі успішного запуску, назви модулів будуть виділені зеленим кольором, а в стовпчиках PID (s) і Port (s) будуть вказані ідентифікатори запусканих процесів і номери зайнятих портів відповідно (рисунок 1.4):

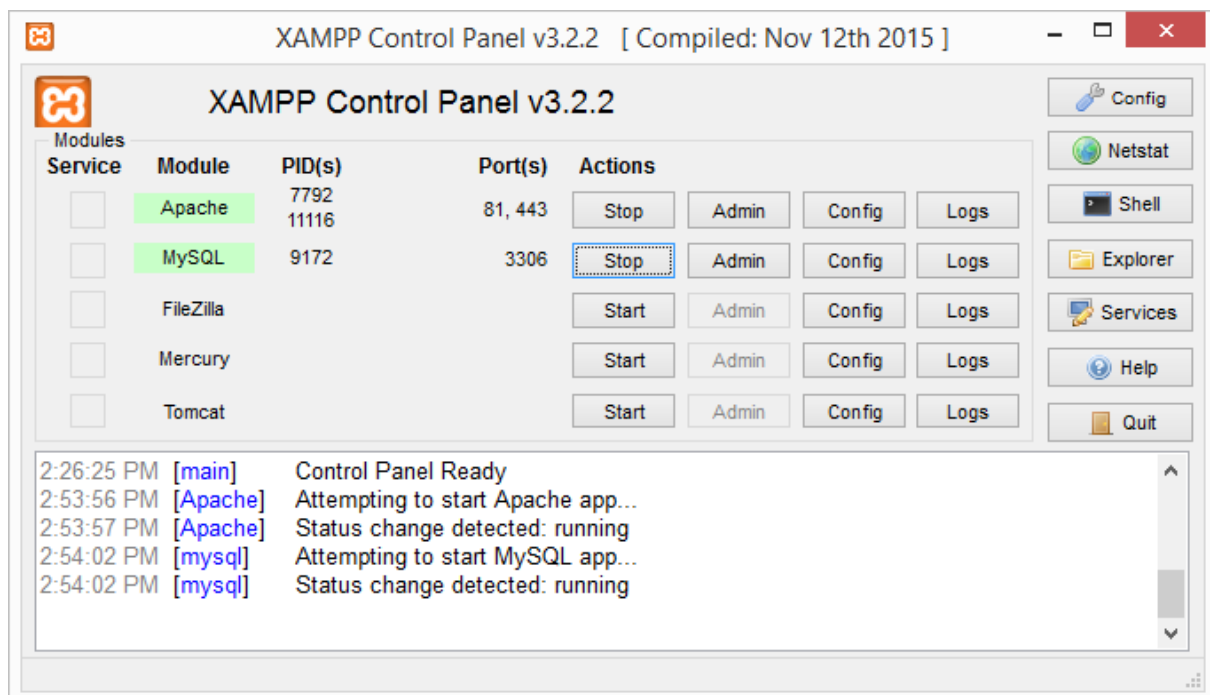


Рисунок 1.4

1.2.2 Робота з MySQL в phpMyAdmin

Додаток phpMyAdmin є web-інтерфейсом для адміністрування системи управління базами даних (СУБД) MySQL. Додаток дозволяє через браузер здійснювати адміністрування сервера MySQL, запускати команди SQL і переглядати вміст таблиць і баз даних. Додаток користується великою популярністю у web-розробників, так як дозволяє управляти СУБД MySQL без безпосереднього введення SQL команд, надаючи дружній інтерфейс.

Для запуску phpMyAdmin необхідно вибрати дію Admin для модуля MySQL в вікні управління XAMPP або перейти за адресою

`http://localhost:<port>/phpmyadmin/`

після того, як Apache і MySQL будуть успішно запущені (рисунок 1.5):

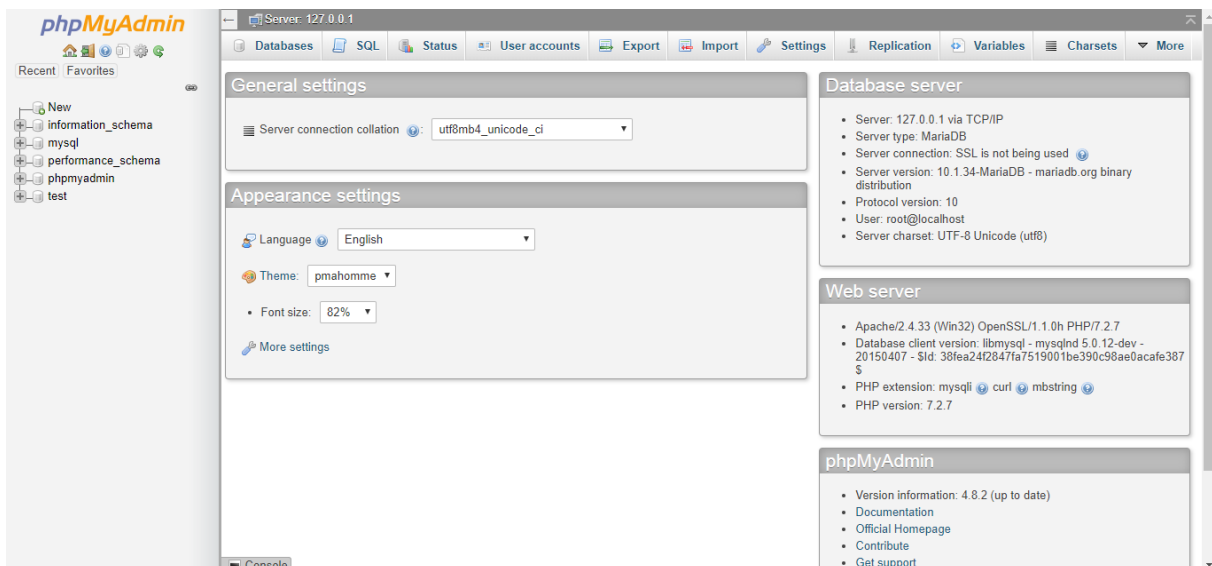
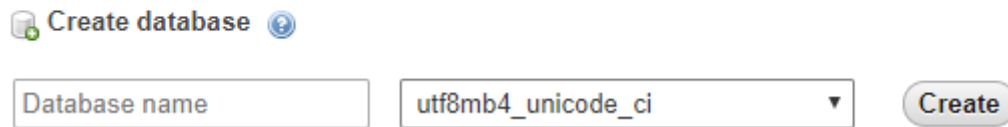


Рисунок 1.5

Для створення нової бази даних необхідно вибрати New над списком доступних баз даних. В результаті буде відкрита форма створення бази даних (рисунок 1.6):

Databases



Create database ?

Database name

utf8mb4_unicode_ci

Create

Рисунок 1.6

В поле Database name необхідно ввести назву створюваної бази даних (наприклад, delivery) і натиснути кнопку Create. Як кодування встановити utf8mb4_unicode_ci. Кодування utf8mb4 необхідно використовувати замість utf8 починаючи з версії MySQL 5.5.3, оскільки кодування utf8 вважається застарілим. В даний час для баз даних і таблиць MySQL рекомендується використовувати кодування utf8mb4_unicode_ci, позбавлене недоліків, пов'язаних з сортуванням в певних мовах.

Після створення бази даних, буде відкрито вікно перегляду структури бази даних, яка не містить в даний момент ніяких таблиць (рисунок 1.7):

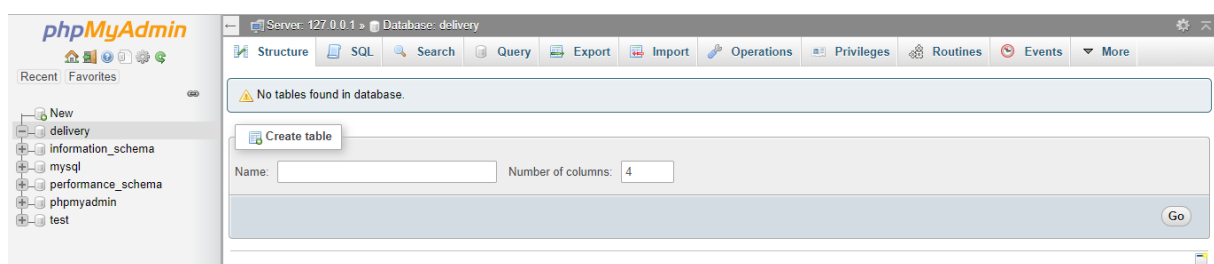


Рисунок 1.7

Для створення першої таблиці в базі даних пропонується ввести ім'я нової таблиці в поле Name, а також вказати кількість стовпців в полі Number of columns (рисунок 1.7).

Для розглянутої (як приклад) предметної області потрібно створити таблицю `supplier` з 3-ма стовпчиками (рисунок 1.8):

Name	Type	Length/Values	Default	Collation	Attributes	Null	Index	A_I
id	INT		None			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>
name	VARCHAR	50	None			<input type="checkbox"/>	---	<input type="checkbox"/>
address	VARCHAR	100	None			<input type="checkbox"/>	---	<input type="checkbox"/>

Рисунок 1.8

У формі послідовно вказати імена стовпців: `id`, `name`, `address`. В якості типу вказати для стовпця `id` тип `INT`, а для стовпців `name` і `company` – тип `VARCHAR`. Для стовпців `name` і `company` в поле `Length / Values` вказати максимальну довжину рядків в символах – 50 і 100 відповідно. Для стовпця `id` вказати в поле `Index` `PRIMARY`, а в поле `A_I` (`Auto Increment`) поставити галочку.

Після натискання кнопки `Save` буде показана структура створеної таблиці і її стовпці (рисунок 1.9):

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(50)	utf8mb4_unicode_ci		No	None			Change Drop More
3	address	varchar(100)	utf8mb4_unicode_ci		No	None			Change Drop More

Рисунок 1.9

Створити таблицю в `phpMyAdmin` можна і за допомогою команди `SQL`. Для цього необхідно перейти на вкладку `SQL` і ввести відповідну команду (рисунок 1.10):

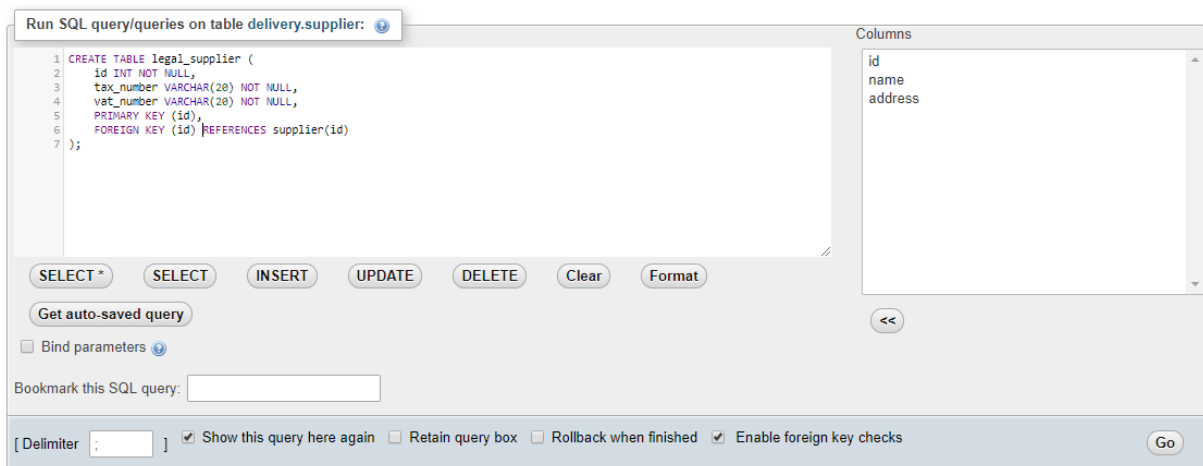


Рисунок 1.10

В результаті виконання даної команди натисканням кнопки Go, буде показаний результат успішного виконання або список помилок (рисунок 1.11):



Рисунок 1.11

Необхідно створити всі необхідні таблиці бази даних, що розробляється. Для цього можна скористатися як формами створення таблиць, так і командами SQL. Однак щоб знизити ймовірність допуску помилок при створенні таблиць бази даних, рекомендується використовувати команди SQL. Перш ніж приступати до створення таблиць, слід ознайомитися з основними типами даних СУБД MySQL за посиланням:

<https://dev.mysql.com/doc/refman/8.0/en/data-types.html>

Структуру створюваної бази даних (таблиці і відносини між ними) можна переглянути, перейшовши на вкладку Designer. Можлива структура створеної бази даних продемонстрована на малюнку (1.12):

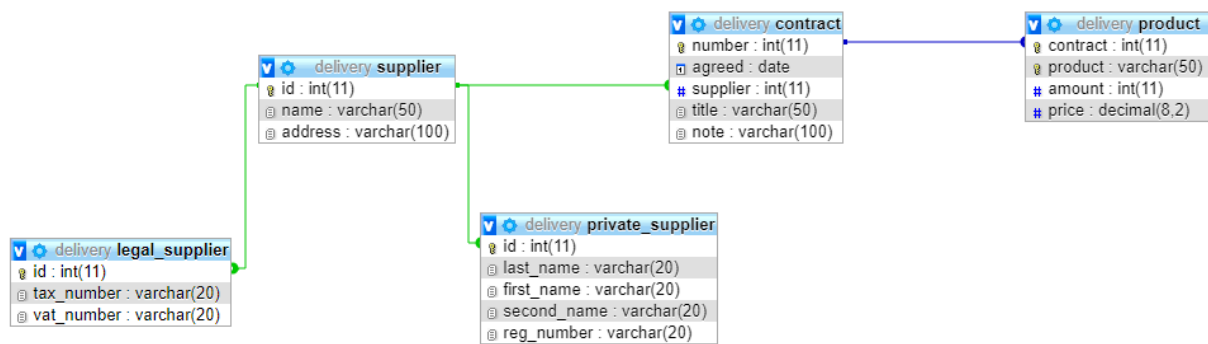


Рисунок 1.12

Для перевірки коректності і працездатності, створені таблиці бази даних необхідно заповнити тестовими наборами даних.

1.2.3 Створення облікових записів користувачів

Як приклад розглядається створення облікового запису користувача - співробітника відділу постачання. Для цього необхідно перейти на вкладку Privileges і вибрати Add user account (рисунок 1.13):

Add user account

Login Information

User name:

Use text field: supply_mngr

Host name:

Any host %

Password:

Use text field: *****

Strength: Very weak

Re-type:

Authentication Plugin

Native MySQL authentication

Generate password:

Generate

Database for user account

☐ Create database with same name and grant all privileges.
 ☐ Grant all privileges on wildcard name (username_%).
 ☐ Grant all privileges on database delivery.

Рисунок 1.13

В поля User name і Password необхідно ввести ім'я користувача supply_manager і пароль supply відповідно. В поле Host name ввести

localhost. У секції Database for user account прибрати галочки з усіх пунктів. Для збереження облікового запису натиснути кнопку Go.

Після створення облікового запису необхідно визначити привілеї на рівні окремих таблиць. Для цього потрібно виконати спеціальну команду SQL. У випадку з обліковим записом для співробітника відділу постачання, команди SQL будуть мати такий вигляд:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON delivery.contract  
TO 'supply_manager'@'localhost'
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON delivery.product  
TO 'supply_manager'@'localhost'
```

```
GRANT SELECT ON delivery.supplier TO  
'supply_manager'@'localhost'
```

```
GRANT SELECT ON delivery.legal_supplier TO  
'supply_manager'@'localhost'
```

```
GRANT SELECT ON delivery.private_supplier TO  
'supply_manager'@'localhost'
```

Для перевірки необхідно перейти на вкладку User accounts бази даних, вибрати Edit privileges для цього облікового запису, вибрати вид Database (рисунок 1.14):

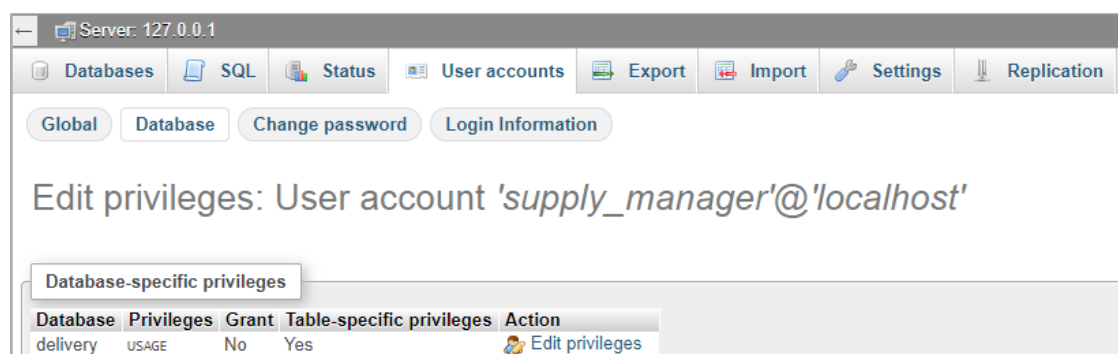


Рисунок 1.14

Для детального перегляду привілеїв по кожній таблиці бази даних вибрати Edit privileges в стовпці Action, переключитися на вигляд Table і перевірити правильність призначених для облікового запису привілеїв (рисунок 1.15):

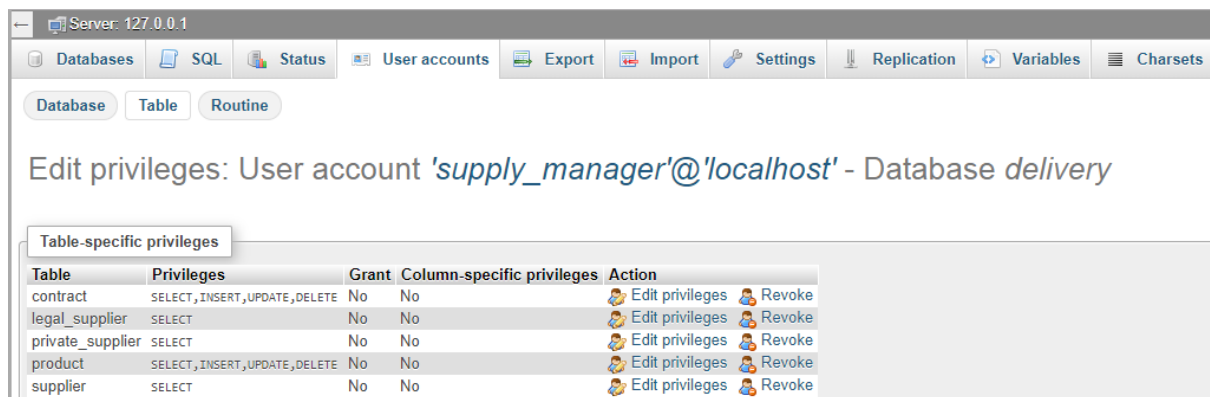


Рисунок 1.15

Додаткові відомості про керування обліковими записами користувачів в СУБД MySQL можна отримати за посиланням:

<https://dev.mysql.com/doc/refman/8.0/en/user-account-management.html>

Облікові записи необхідно створити для кожного користувача бази даних, визначеного на етапі проектування системи. Призначаються привілеї, які повинні відповідати призначеним для користувача історіям і прецедентів.

1.2.4 Створення тригерів для таблиць

Створення тригера в СУБД MySQL здійснюється за допомогою команди CREATE TRIGGER, яка має наступний синтаксис:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON table_name
FOR EACH ROW
BEGIN
...

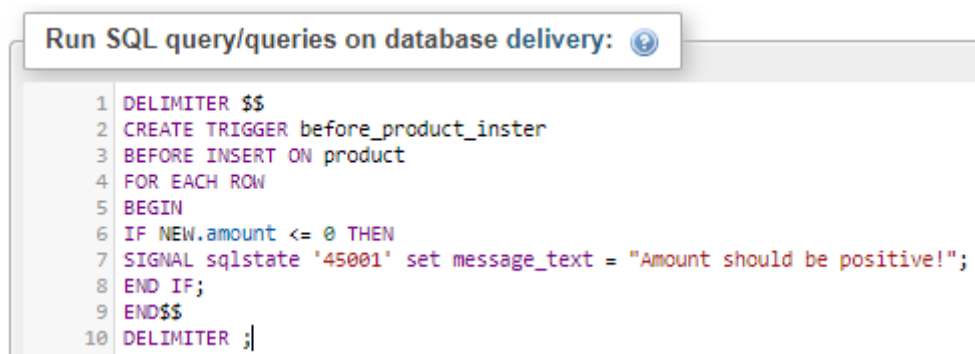
```

END;

Розглянемо синтаксис даної команди більш докладно:

- 1) ім'я тригера має відповідати угодою по іменування [trigger time] _ [table name] _ [trigger event], наприклад before_product_update;
- 2) часом активації тригера може бути BEFORE (перед виконанням зміни) або AFTER (після виконання зміни);
- 3) подіями тригера можуть бути INSERT, UPDATE або DELETE, причому для обробки кожної події необхідно створювати окремий тригер;
- 4) тригер асоціюється з певною таблицею;
- 5) команди SQL необхідно поміщати в блок між BEGIN і END, де визначається логіка тригера.

Як приклад розглядається створення тригера, призначеного для контролю вводяться записів про поставлені товари, з ім'ям before_product_insert. Для цього необхідно виконати відповідну команду SQL (рисунок 1.16):



```
1 DELIMITER $$
2 CREATE TRIGGER before_product_inster
3 BEFORE INSERT ON product
4 FOR EACH ROW
5 BEGIN
6 IF NEW.amount <= 0 THEN
7 SIGNAL sqlstate '45001' set message_text = "Amount should be positive!";
8 END IF;
9 END$$
10 DELIMITER ;|
```

Рисунок 1.16

Команда DELIMITER не відноситься до синтаксису тригера. Вона використовується для зміни обмежувача, встановленого за замовчуванням, для того, щоб передавати команду створення тригера цілком на сервер, не дозволяючи MySQL інтерпретувати кожен рядок окремо.

Для перевірки працездатності створеного тригера маєте змогу надсилати запити, що порушує задана при визначенні тригера умова:

```
INSERT INTO `supplier` (`id`, `name`, `address`) VALUES
(1, 'test', 'test');
INSERT INTO `contract` (`number`, `agreed`, `supplier`,
`title`, `note`) VALUES (1, '8-6-2018', 1, 'test', 'test');
INSERT INTO `product` (`contract`, `product`, `amount`,
`price`) VALUES (1, 'test', -5, 10);
```

В результаті виконання останнього запиту, спрацює створений раніше тригер і буде видано відповідне повідомлення про помилку (рисунок 1.17):

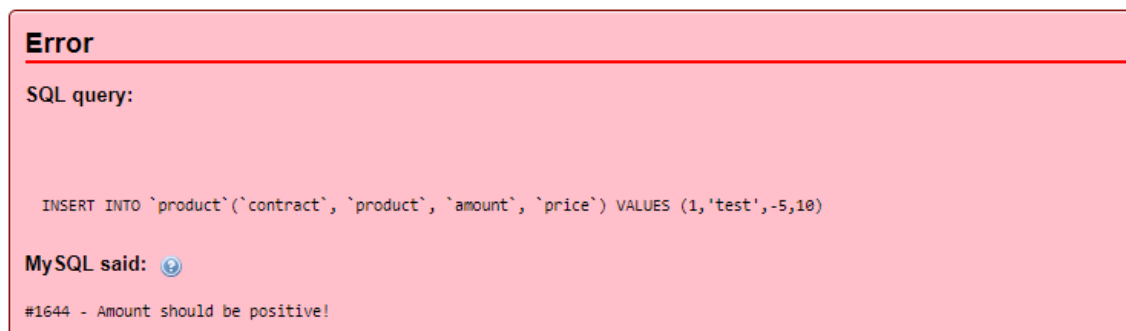


Рисунок 1.17

Створені тригери доступні для перегляду в phpMyAdmin на вкладці Triggers (рисунок 1.18):

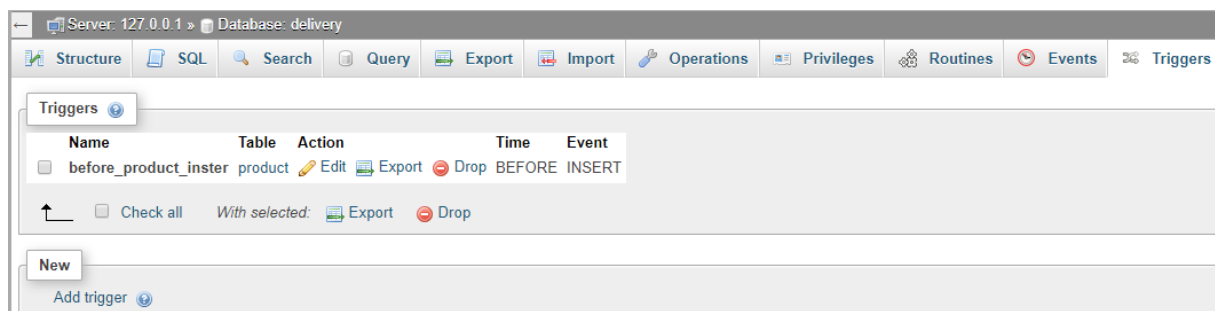


Рисунок 1.18

Детальну інформацію про тригерах в СУБД MySQL можна отримати за посиланням:

<https://dev.mysql.com/doc/refman/8.0/en/triggers.html>

Для кожної таблиці бази даних необхідно створити тригери, якщо це потрібно в залежності від особливостей заданої предметної області. Час активації і події спрацьовування тригера визначити самостійно, ґрунтуючись також на особливостях конкретної предметної області. Протестувати створені тригери за допомогою одного з методів тестування «білого ящика».

1.2.5 Створення збережених процедур і функцій

Збережені підпрограми (процедури і функції) представляють собою набір команд SQL, які можуть компілюватися і зберігатися на сервері. Таким чином, замість того, щоб зберігати часто використовуваний запит, клієнти можуть посилатися на відповідну процедуру, що зберігається. Це забезпечує кращу продуктивність, оскільки даний запит повинен аналізуватися тільки один раз і зменшується трафік між сервером і клієнтом.

Збережені підпрограми можуть бути корисні в разі, якщо численні програми клієнта написані на різних мовах або працюють на різних платформах, але потрібно використовувати одну і ту ж базу даних (рисунок 1.19):

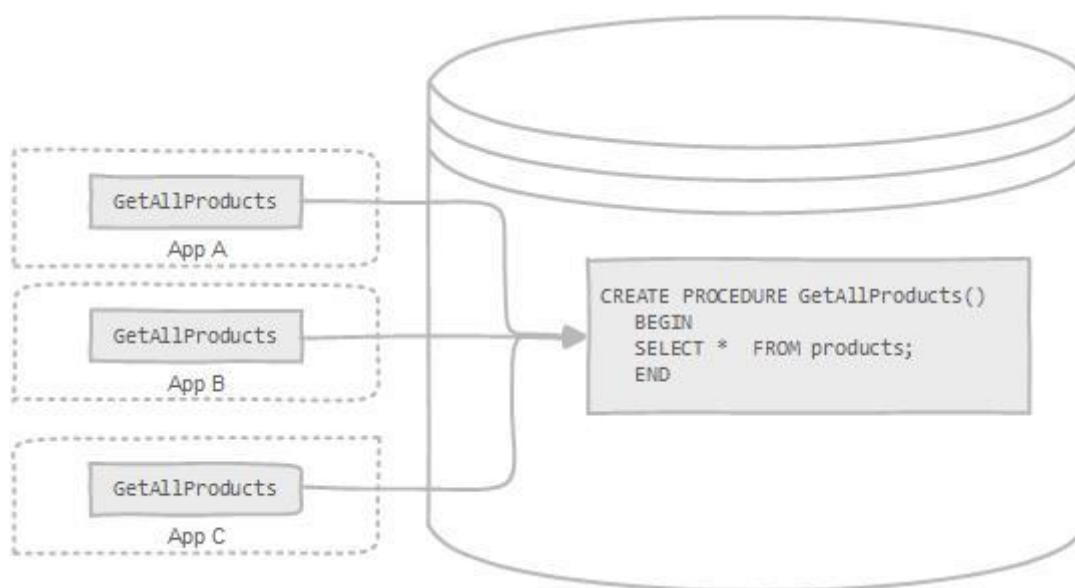


Рисунок 1.19

Збережена підпрограма є процедуру або функцію. Збережені підпрограми створюються за допомогою виразів `CREATE PROCEDURE` або `CREATE FUNCTION`. Збережена підпрограма викликається, використовуючи вираз `CALL`, причому тільки повертають значення змінні використовуються в якості вихідних. Функція може бути викликана подібно будь-якій іншій функції і може повертати скалярну величину. Збережені підпрограми можуть викликати інші збережені підпрограми.

Команди `CREATE PROCEDURE` і `CREATE FUNCTION` мають наступний синтаксис:

```
CREATE PROCEDURE procedure_name ([procedure_param[,...]])
[characteristic ...] procedure_body
```

```
CREATE FUNCTION function_name ([function_param[,...]])
RETURNS type
[characteristic...] function_body
```

```
procedure_param:
    [ IN | OUT | INOUT ] param_name type
function_param:
    param_name type
```

```
type:
```

any data type of MySQL

characteristic:

```
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
```

routine_body:

correct SQL expression.

Як приклад розглядається створення процедури, призначеної для отримання списку поставлених товарів за номером договору, з ім'ям `GetListOfSuppliedProductsByContractNumber`. Для цього необхідно виконати відповідну команду SQL (рисунок 1.20):

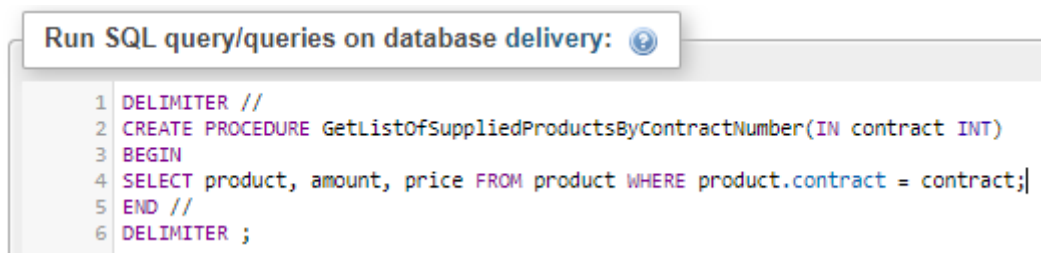


Рисунок 1.20

Створені для бази даних збережені процедури і функції доступні для перегляду на вкладці Routines (рисунок 1.21):

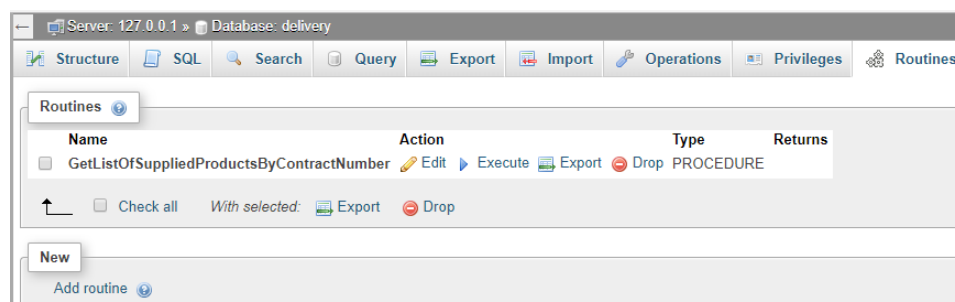
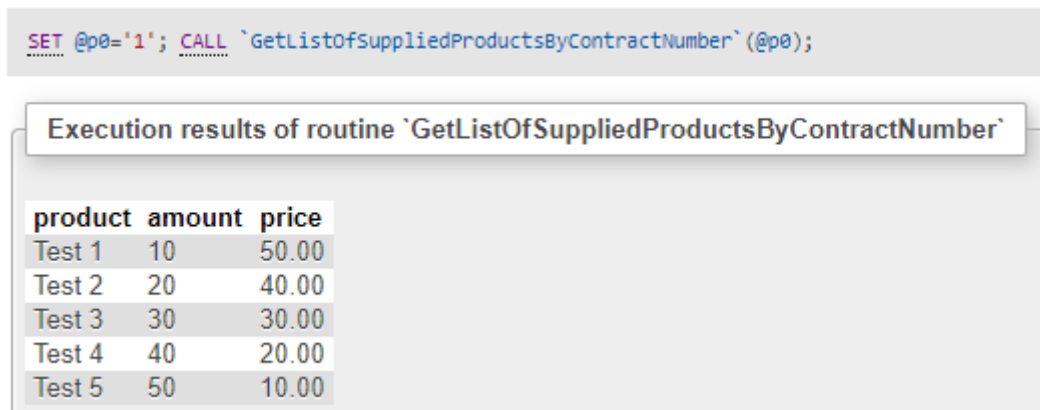


Рисунок 1.21

Для перевірки працездатності створеної процедури, що, необхідно викликати її за допомогою кнопки Execute в стовпці Action і ввести необхідне значення параметра (рисунок 1.22):



SET @p0='1'; CALL `GetListOfSuppliedProductsByContractNumber`(@p0);

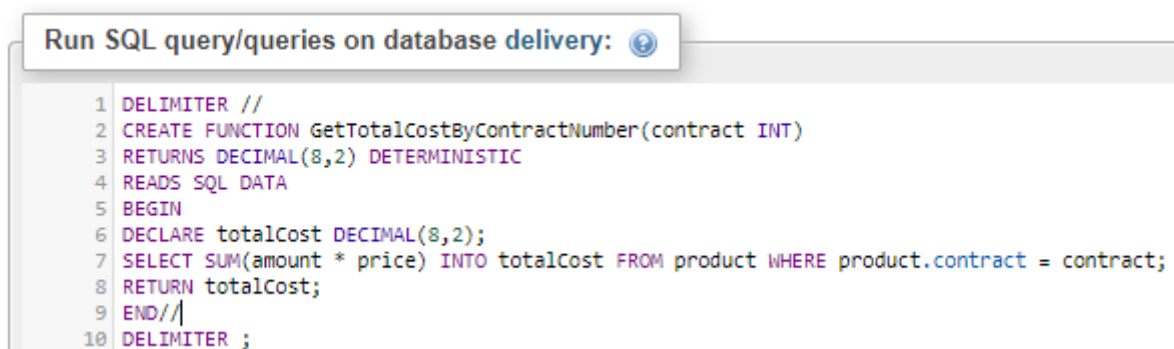
Execution results of routine `GetListOfSuppliedProductsByContractNumber`

product	amount	price
Test 1	10	50.00
Test 2	20	40.00
Test 3	30	30.00
Test 4	40	20.00
Test 5	50	10.00

Рисунок 1.22

В результаті виконання процедури виводиться також команда SQL, за допомогою якої дана процедура була викликана.

Як приклад також розглядається створення функції, призначеної для обчислення загальної вартості поставлених товарів за договором, з ім'ям GetTotalCostByContractNumber. Для цього необхідно виконати відповідну команду SQL (рисунок 1.23):



Run SQL query/queries on database delivery: ⓘ

```
1 DELIMITER //
2 CREATE FUNCTION GetTotalCostByContractNumber(contract INT)
3 RETURNS DECIMAL(8,2) DETERMINISTIC
4 READS SQL DATA
5 BEGIN
6 DECLARE totalCost DECIMAL(8,2);
7 SELECT SUM(amount * price) INTO totalCost FROM product WHERE product.contract = contract;
8 RETURN totalCost;
9 END//
10 DELIMITER ;
```

Рисунок 1.23

В результаті виконання створеної функції, результат виводиться аналогічно результату виклику процедури (рисунки 1.24):

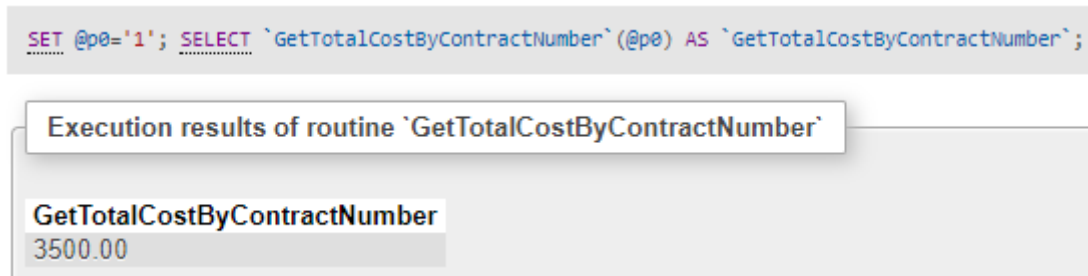


Рисунок 1.24

Детальну інформацію про збережених процедурах і функціях в СУБД MySQL можна отримати за посиланням:

<https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>

Для розробленої (згідно заданої предметної області) бази даних необхідно створити процедури і функції. Створювані процедури і функції повинні повністю задовольняти функціональним вимогам, визначеним на етапі проектування системи. Протестувати створені процедури і функції за допомогою одного з методів «білого ящика».

1.3 Робота з базою даних в мові PHP

1.3.1 Функції для роботи з базою даних

Перш ніж отримати дані з бази даних MySQL, необхідно встановити з'єднання з сервером. Для установки з'єднання використовується функція `mysqli_connect`, яка приймає такі аргументи (рисунки 1.25):

- 1) `servername` – ім'я сервера MySQL;
- 2) `username` – ім'я користувача (заданий при створенні облікового запису);
- 3) `password` – пароль користувача (також встановлений при створенні облікового запису);

4) dbname – ім'я бази даних, до якої виконується підключення.

```
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
```

Рисунок 1.25

Для виконання SQL запитів до бази даних, використовується функція `mysqli_query`, приймаючи два аргументи (рисунок 1.26):

- 1) `conn` – результат виконання функції `mysqli_connect`;
- 2) `sql` – рядкова змінна, яка містить текст команди SQL.

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);
```

Рисунок 1.26

Після того, як запит був виконаний, функція `mysqli_query` поверне результат його виконання. В даному випадку запит був на вибірку даних, тому необхідно обробити отриманий результат для виведення на екран необхідної інформації.

За допомогою функції `mysqli_num_rows` можна отримати кількість записів, отриманих в результаті виконання заданої команди `SELECT`. Ця функція приймає єдиний аргумент – результат виконання функції `mysqli_query`. Для отримання інформації по кожному записі, використовується функція `mysqli_fetch_assoc`, яка повертає асоціативний

масив, що містить пари значень «ім'я стовпця – вміст комірки». Приймає дана функція також результат виконання `mysqli_query` (рисунок 1.27):

```
if (mysqli_num_rows($result) > 0) {  
    // output data of each row  
    while($row = mysqli_fetch_assoc($result)) {  
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "  
<br>";  
    }  
} else {  
    echo "0 results";  
}
```

Рисунок 1.27

Припустимо, що замість запиту на вибірку, необхідно виконати запит на оновлення або, наприклад, додавання даних. При цьому в результаті введення некоректних даних, може спрацювати один з розроблених тригерів і видати відповідне повідомлення про помилку. Для отримання такого повідомлення можна використовувати функцію `mysqli_error`. Вона приймає єдиний аргумент – результат виконання функції `mysqli_connect` (рисунок 1.28):

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)  
VALUES ('John', 'Doe', 'john@example.com')";  
  
if (mysqli_query($conn, $sql)) {  
    echo "New record created successfully";  
} else {  
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);  
}
```

Рисунок 1.28

Після того, як всі необхідні дії з базою даних були виконані, встановлене з'єднання потрібно закрити за допомогою функції `mysqli_close`. Ця функція приймає результат виконання функції `mysqli_connect`.

Перш ніж приступати до наступних етапів виконання роботи, рекомендується ознайомитися з особливостями мови PHP за наступним посиланням:

<https://www.w3schools.com/php/default.asp>

1.3.2 Патерн проектування Repository

Патерн Repository є посередником між шаром області визначення і шаром розподілу даних, працюючи, як звичайна колекція об'єктів області визначення. Об'єкти-клієнти створюють опис запиту декларативно і направляють їх до об'єкта-сховища (Repository) для обробки. Об'єкти можуть бути додані або видалені з сховища, як ніби вони формують просту колекцію об'єктів. А код розподілу даних, прихований в об'єкті Repository, подбає про відповідні операції непомітно для розробника (рисунок 1.29):

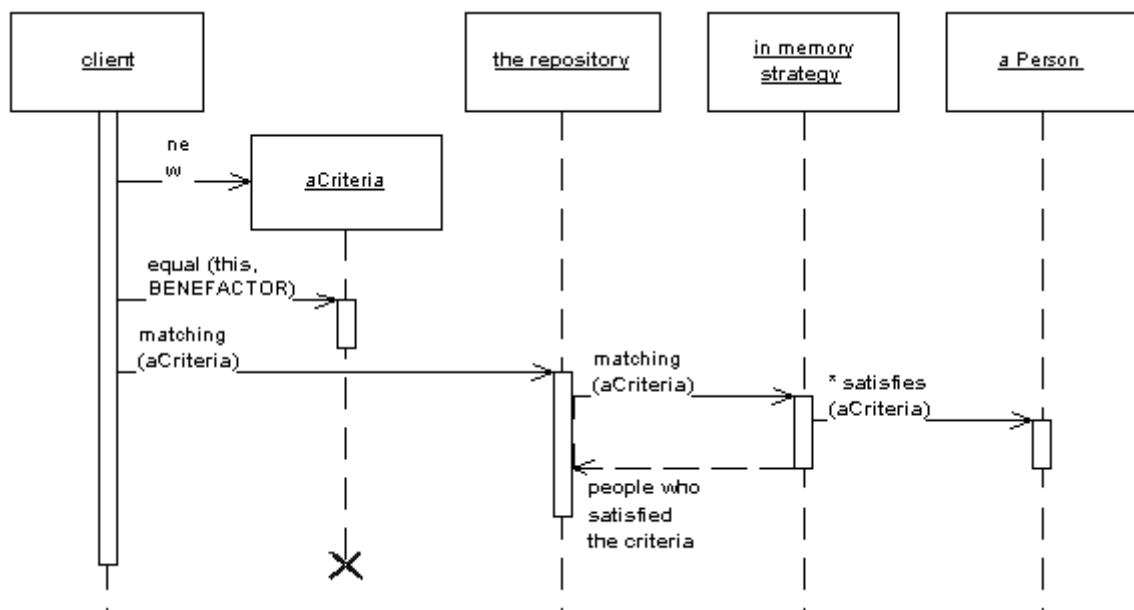


Рисунок 1.29

Патерн Repository інкапсулює об'єкти, представлені в сховищі даних і операції, вироблені над ними, надаючи об'єктно-орієнтоване уявлення реальних даних. Repository також має на меті досягнення повного поділу і

односторонньої залежності між рівнями області визначення і розподілу даних.

Як приклад, розглядається реалізація патерну Repository для роботи з об'єктами, що описують інформацію про договори:

1. Клас Contract:

```
class Contract
{
    private $number;
    private $agreed;
    private $supplier;
    private $title;
    private $note;

    public function __construct($number, $agreed, $supplier, $title, $note)
    {
        if (empty($number))
        {
            throw new Exception('Contract number is not set!');
        }

        if (empty($supplier))
        {
            throw new Exception('Supplier is not set!');
        }

        if (empty($title))
        {
            throw new Exception('Contract title is not set!');
        }

        if (empty($note))
        {
            throw new Exception('Contract note is not set!');
        }

        $this->number = $number;
        $this->agreed = $agreed;
        $this->supplier = $supplier;
        $this->title = $title;
        $this->note = $note;
    }

    public function getNumber()
    {
        return $this->number;
    }

    public function getAgreed()
    {
        return $this->agreed;
    }

    public function getSupplier()
    {
        return $this->supplier;
    }

    public function getTitle()
    {
        return $this->title;
    }

    public function getNote()
    {
        return $this->note;
    }
}
```



```
}
```

2. Інтерфейс `ContractRepositoryInterface`, що визначає поведінку об'єкта, що працює з даними про договори:

```
interface ContractRepositoryInterface
{
    public function getContractList();

    public function getContractByNumber($number);

    public function create($contract);

    public function update($contract);

    public function delete($number);
}
```

3. Реалізація інтерфейсу `ContractRepositoryInterface`, призначена для роботи з базою даних MySQL – клас `MySQLContractRepository`:

```
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Domain/Contract.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Util/MySQLConnectionUtil.php');

require_once('ContractRepositoryInterface.php');

class MySQLContractRepository implements ContractRepositoryInterface
{
    public function getContractList()
    {
        $conn = MySQLConnectionUtil::getConnection();
        $contracts = array();

        $query = 'SELECT number, agreed, supplier, title, note FROM contract';
        $result = mysqli_query($conn, $query);

        while ($row = mysqli_fetch_assoc($result))
        {
            $contract = new Contract($row['number'], $row['agreed'], $row['supplier'],
                $row['title'], $row['note']);

            array_push($contracts, $contract);
        }

        mysqli_close($conn);

        return $contracts;
    }

    public function getContractByNumber($number)
    {
        $conn = MySQLConnectionUtil::getConnection();
        $contract = NULL;

        $query = "SELECT number, agreed, supplier, title, note FROM contract
            WHERE number = {$number}";
        $result = mysqli_query($conn, $query);

        while ($row = mysqli_fetch_assoc($result))
        {
            $contract = new Contract($row['number'], $row['agreed'], $row['supplier'],
                $row['title'], $row['note']);

            break;
        }
    }
}
```

```

    }

    mysqli_close($conn);

    if ($contract == NULL)
    {
        throw new Exception("Contract with number {$number} doesn't exist!");
    }

    return $contract;
}

public function create($contract)
{
    $conn = MySQLConnectionUtil::getConnection();

    $number = $contract->getNumber();
    $agreed = $contract->getAgreed();
    $supplier = $contract->getSupplier();
    $title = $contract->getTitle();
    $note = $contract->getNote();

    $query = "INSERT INTO contract(number, agreed, supplier, title, note)
        VALUES ({ $number }, '{ $agreed }', { $supplier }, '{ $title }', '{ $note }')";
    $result = mysqli_query($conn, $query);

    if (!$result)
    {
        throw new Exception(mysqli_error($conn));
    }

    mysqli_close($conn);
}

public function update($contract)
{
    $conn = MySQLConnectionUtil::getConnection();

    $number = $contract->getNumber();
    $agreed = $contract->getAgreed();
    $supplier = $contract->getSupplier();
    $title = $contract->getTitle();
    $note = $contract->getNote();

    $query = "UPDATE contract SET agreed = '{ $agreed }', supplier = { $supplier },
        title = '{ $title }', note = '{ $note }' WHERE number = { $number }";
    $result = mysqli_query($conn, $query);

    if (!$result)
    {
        throw new Exception(mysqli_error($conn));
    }

    mysqli_close($conn);
}

public function delete($number)
{
    $conn = MySQLConnectionUtil::getConnection();

    $query = "DELETE FROM contract WHERE number = { $number }";
    $result = mysqli_query($conn, $query);

    if (!$result)
    {
        throw new Exception(mysqli_error($conn));
    }

    mysqli_close($conn);
}
}

```

4. Для установки з'єднання з базою даних MySQL використовується клас MySQLConnectionUtil:

```
class MySQLConnectionUtil
{
    public static function getConnection()
    {
        $servername = 'localhost';
        $username = $_SESSION['username'];
        $password = $_SESSION['password'];
        $database = 'delivery';

        $conn = mysqli_connect($servername, $username, $password, $database);

        if (!$conn)
        {
            throw new Exception(mysqli_connect_error());
        }

        return $conn;
    }
}
```

Створені класи і інтерфейси необхідно розташувати в каталозі хатрр/htdocs. Зразковий вигляд структури каталогів і файлів представлений на рисунку 1.30:

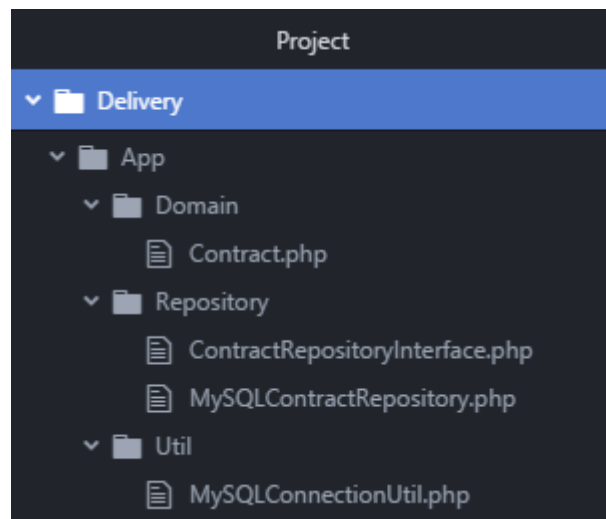


Рисунок 1.30

Для заданої предметної області необхідно реалізувати всі необхідні класи-колекції об'єктів, використовуючи патерн Repository. При розробці

класів для роботи з СУБД MySQL, бажано передбачити захист від SQL-ін'єкцій, попередньо ознайомившись з матеріалом по посиланню:

<http://php.net/manual/ru/security.database.sql-injection.php>

1.3.3 Патерн проектування Service Layer

Патерн Service Layer визначає для застосунка межу і набір допустимих операцій з точки зору взаємодіючих з ним клієнтських (рисунок 1.31). Він інкапсулює бізнес-логіку застосунка, керуючи транзакціями і керуючи відповідями в реалізації цих операцій.

Бізнес-застосунки зазвичай потребують різних інтерфейсах до даних, які вони зберігають, і логіці, яку реалізують:

- 1) завантажувачі даних;
- 2) призначені для користувача інтерфейси;
- 3) інтеграційні шлюзи і ін.

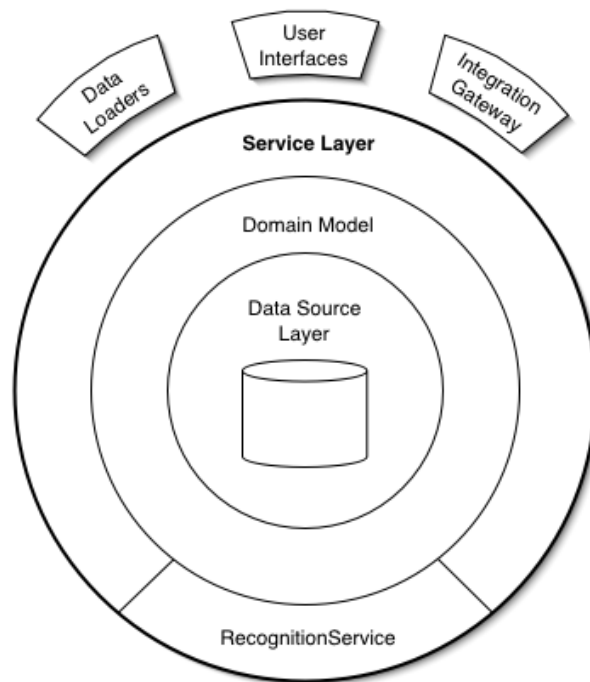


Рисунок 1.31

Дані інтерфейси часто потребують у взаємодії із застосунком для доступу і управління його даними і виконання логіки. Ці взаємодії можуть

бути складними, що використовують транзакції на декількох ресурсах і управління декількома відповідями на дію. Програмування логіки взаємодії для кожного інтерфейсу викличе більшу кількість дублювання.

Як приклад, розглядається створення класу `ContractService`, що інкапсулює бізнес-логіку роботи з договорами:

```
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Domain/Contract.php');
require_once
($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Repository/ContractRepositoryInterface.php');

class ContractService
{
    private $repository;

    public function __construct(ContractRepositoryInterface $repository)
    {
        $this->repository = $repository;
    }

    public function getAllContracts()
    {
        return $this->repository->getContractList();
    }

    public function getContractByNumber($number)
    {
        if (isset($number))
        {
            return $this->repository->getContractByNumber($number);
        }
        else
        {
            throw new Exception('Contract number is undefined!');
        }
    }

    public function createContract($number, $supplier, $title, $note)
    {
        if (isset($number, $supplier, $title, $note))
        {
            $agreed = date('Y-m-d');

            $contract = new Contract($number, $agreed, $supplier, $title, $note);

            $this->repository->create($contract);
        }
        else
        {
            throw new Exception('Please fill in all contract fields!');
        }
    }

    public function updateContract($number, $supplier, $title, $note)
    {
        if (isset($number, $supplier, $title, $note))
        {
            $contract = $this->repository->getContractByNumber($number);

            $updated = new Contract($number, $contract->getAgreed(), $supplier, $title,
$note);

            $this->repository->update($updated);
        }
        else
        {
            throw new Exception('Please fill in all contract fields!');
        }
    }
}
```

```

    }

    public function deleteContract($number)
    {
        if (isset($number))
        {
            $this->repository->delete($number);
        }
        else
        {
            throw new Exception('Contract number is undefined!');
        }
    }
}

```

Відповідно до заданої структури файлів і каталогів проекту, файл `ContractService.php` необхідно розташувати в `Delivery/App/Service`.

Для заданої предметної області необхідно реалізувати всі необхідні класи, інкапсулюючи бізнес-логіку створюваної системи на основі розглянутого патерна `Service Layer`. Протестувати створені класи за допомогою одного з методів «білого ящика».

Перевірити результати виконання методів в створених класах можна, наприклад, наступним чином (перевірка отримання інформації про договір за його номером):

```

require_once
($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Repository/MySQLContractRepository.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Service/ContractService.php');

session_start();

$_SESSION['username'] = 'supply_manager';
$_SESSION['password'] = 'supply';

class TestContractService
{
    private $repository;
    private $service;

    public function __construct()
    {
        $this->repository = new MySQLContractRepository();
        $this->service = new ContractService($this->repository);
    }

    public function shouldReturnContractByNumber()
    {
        try
        {
            print_r($this->service->getContractByNumber(1));
        }
        catch (Exception $e)
        {
            echo $e->getMessage();
        }
    }

    public function shouldThrowExceptionWhenGetContractByUndefinedNumber()

```

```

    {
        try
        {
            print_r($this->service->getContractByNumber(NULL));
        }
        catch (Exception $e)
        {
            echo $e->getMessage();
        }
    }

    public function shouldThrowExceptionWhenGetContractByInexistentNumber()
    {
        try
        {
            print_r($this->service->getContractByNumber(-1));
        }
        catch (Exception $e)
        {
            echo $e->getMessage();
        }
    }
}

$test = new TestContractService();

$test->shouldReturnContractByNumber();
$test->shouldThrowExceptionWhenGetContractByUndefinedNumber();
$test->shouldThrowExceptionWhenGetContractByInexistentNumber();

```

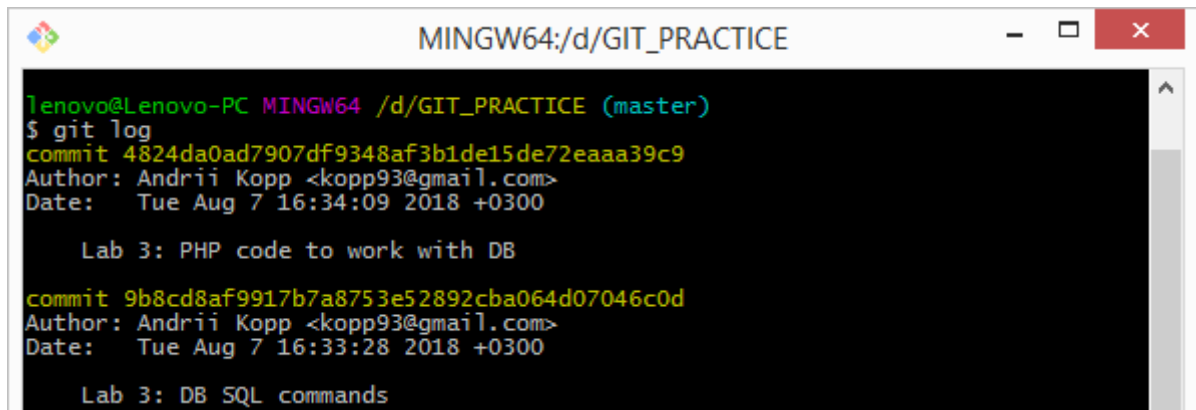
Файл з перевітками `TestContractService.php` необхідно помістити в каталог `Delivery/Test`. Перевагою при виконанні даного завдання буде використання одного з фреймворків для тестування, наприклад PHPUnit:

<https://phpunit.de/getting-started/phpunit-7.html>

У репозиторії Git необхідно зафіксувати (рисунок 1.32):

1) команди SQL, використані при створенні таблиць бази даних, тригерів, збережених процедур і функцій, а також набору тестових даних (помістити в створений в робочому каталозі підкаталог, наприклад, `D:\GIT_PRACTICE\db`);

2) вихідний код розроблених класів (помістити в створений раніше каталог `web`, наприклад `D:\GIT_PRACTICE\web\Delivery`).



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit 4824da0ad7907df9348af3b1de15de72eaaa39c9
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Aug 7 16:34:09 2018 +0300

    Lab 3: PHP code to work with DB

commit 9b8cd8af9917b7a8753e52892cba064d07046c0d
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Aug 7 16:33:28 2018 +0300

    Lab 3: DB SQL commands
```

Рисунок 1.32

Після закінчення роботи, злити гілку storage в master.

1.4 Вимоги до звіту

- 1) коротко описати основні етапи виконання роботи;
- 2) привести структуру створеної бази даних і відносин між таблицями, текст команд SQL, а також тестові випадки для тригерів, збережених процедур і функцій;
- 3) привести структуру створених інтерфейсів і класів за допомогою діаграми класів на мові моделювання UML, їх вихідний код, а також тестові випадки для методів класів, інкапсулюючих бізнес-логіку застосунка;
- 4) продемонструвати отримані результати у вигляді результатів виконання команд SQL і тестування методами «білого ящика».

1.5 Питання для самоперевірки

1. Які проблеми можуть виникнути при запуску сервера XAMPP? Яким чином можна вирішити дані проблеми?
2. Для чого використовується додаток phpMyAdmin?
3. Як створити нову базу даних в СУБД MySQL?

4. Яку кодування рекомендується використовувати? У чому перевага обраної кодування?
5. Які існують способи створення таблиць бази даних? Назвіть основні типи даних СУБД MySQL.
6. Яким чином можна переглянути структуру створеної бази даних в phpMyAdmin?
7. Для чого використовуються облікові записи користувачів? Як створити обліковий запис користувача і встановити необхідні привілеї?
8. Опишіть синтаксис команди, призначеної для створення тригера в СУБД MySQL. Для чого використовується команда DELIMITER?
9. Опишіть синтаксис команди, призначеної для створення збереженої процедури в СУБД MySQL.
10. Опишіть синтаксис команди, призначеної для створення функції в СУБД MySQL.
11. Яким чином можна встановити з'єднання з базою даних MySQL в мові PHP?
12. Яким чином можна виконати запит до бази даних MySQL в мові PHP?
13. Яким чином можна отримати результат виконання запиту до бази даних MySQL в мові PHP?
14. Яким чином можна отримати відомості про помилки, що виникають при роботі з базою даних MySQL в мові PHP?
15. Яким чином можна закрити з'єднання з базою даних MySQL в мові PHP?
16. Для чого призначений патерн проєктування Repository?
17. Для чого призначений патерн проєктування Service Layer?

2 СТВОРЕННЯ WEB-ЗАСТОСУНКІВ ЗА ДОПОМОГОЮ ФРЕЙМВОРКА BOOTSTRAP. ВИКОРИСТАННЯ ТЕХНОЛОГІЇ AJAX ДЛЯ АСИНХРОННОГО ОБМІНУ ДАНИМИ З WEB- СЕРВЕРОМ

2.1 Підготовка до виконання роботи

1. Створити нову гілку в системі управління версіями Git і назвати її pages.

2. Завантажити фреймворк Bootstrap, призначений для створення сайтів і web-застосунків, за посиланням:

<https://getbootstrap.com/docs/4.1/getting-started/download/>

Для швидкого початку роботи з Bootstrap можна також використовувати пропонований спосіб:

<https://getbootstrap.com/docs/4.1/getting-started/introduction/#quick-start>

3. Перебуваючи на створеній гілці, в каталозі Delivery створити підкаталог Web, помістивши в нього вміст завантаженого архіву з фреймворком Bootstrap і, в подальшому, створювані сторінки застосунка.

2.2 Знайомство з фреймворком Bootstrap

Фреймворк Bootstrap заснований на сучасних напрацюваннях в області HTML і CSS. Включає в себе шаблони оформлення для типографіки, web-форм, кнопок, міток, блоків навігації та інших компонентів web-інтерфейсу, включаючи JavaScript-розширення.

Основні інструменти Bootstrap:

1. Сітки – заздалегідь задані розміри колонок, які можна відразу ж використовувати (рисунок 2.1). Детальна інформація про використання сіток доступна за посиланням:

<https://getbootstrap.com/docs/4.1/layout/grid/#grid-options>

2. Шаблони – фіксований або безрозмірний шаблон документа (рисунок 2.2). Детальна інформація про використання шаблонів доступна за посиланням:

<https://getbootstrap.com/docs/4.1/layout/grid/#responsive-classes>

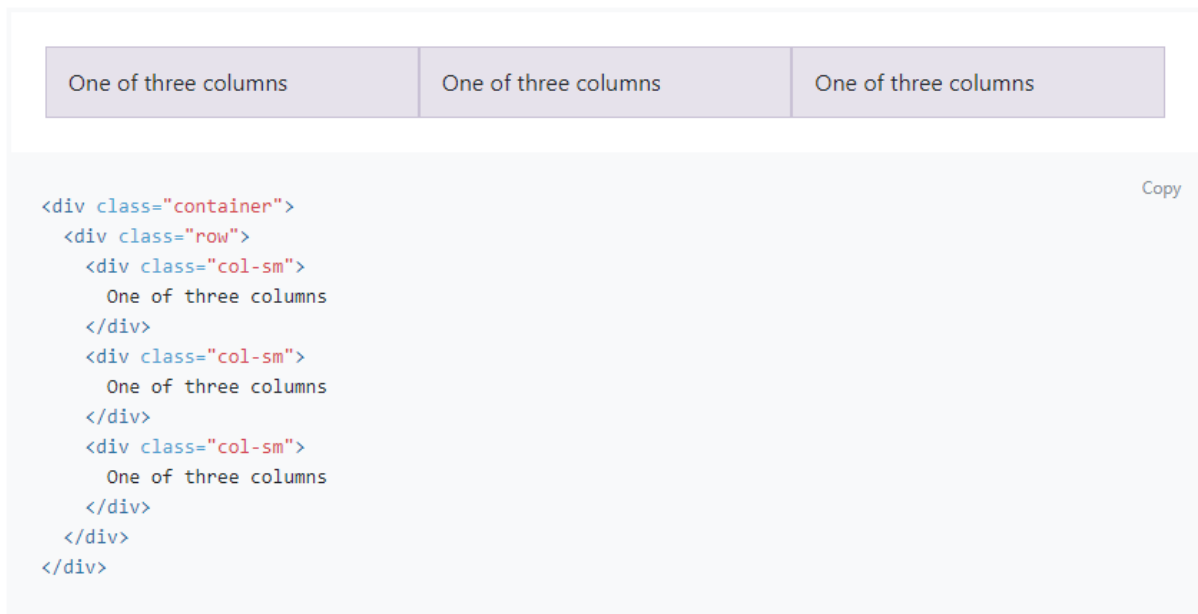


Рисунок 2.1

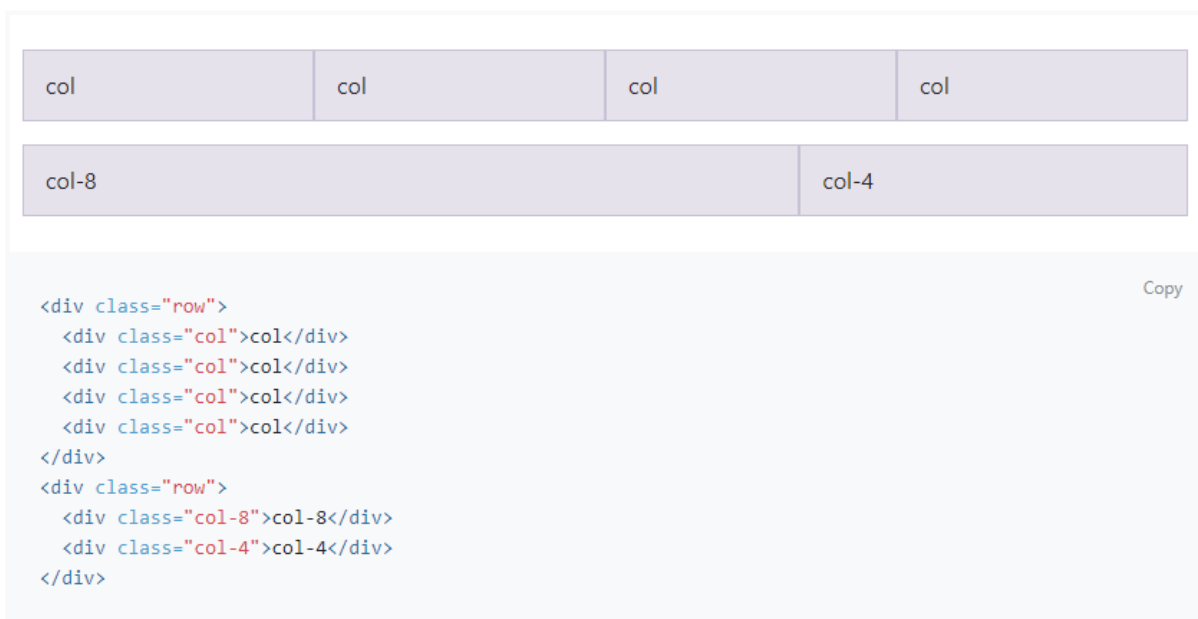


Рисунок 2.2

3. Типографіка – опис шрифтів, визначення деяких класів для шрифтів, таких як код, цитати і т.д. (рисунок 2.3). Детальна інформація про використання типографіки доступна за посиланням:

<https://getbootstrap.com/docs/4.1/content/typography/>

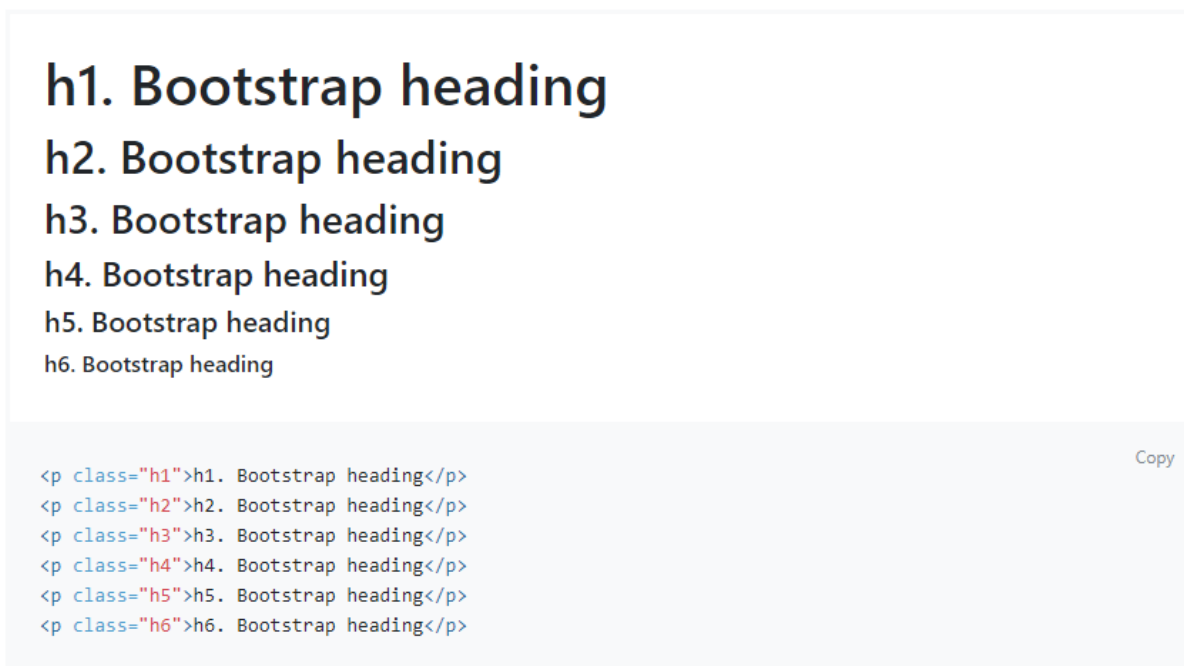


Рисунок 2.3

4. Медіа – представляє деякий спосіб упорядкування зображень та відео (рисунок 2.4). Детальна інформація про використання медіа доступна за посиланням:

<https://getbootstrap.com/docs/4.1/layout/media-object/>

5. Таблиці – засоби оформлення таблиць, аж до додавання функціональності сортування (рисунок 2.5). Детальна інформація про використання таблиць доступна за посиланням:

<https://getbootstrap.com/docs/4.1/content/tables/>

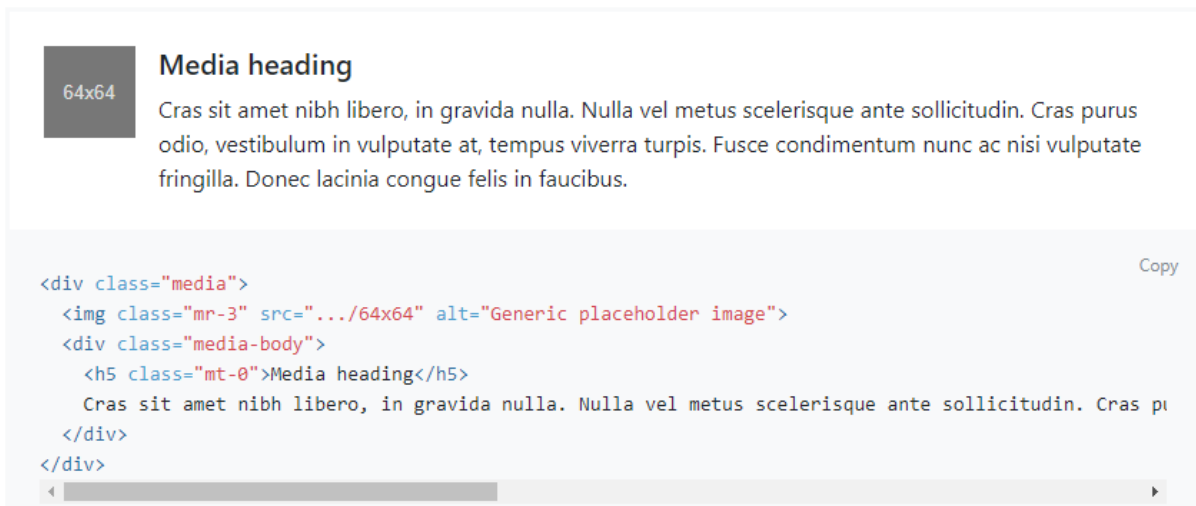


Рисунок 2.4

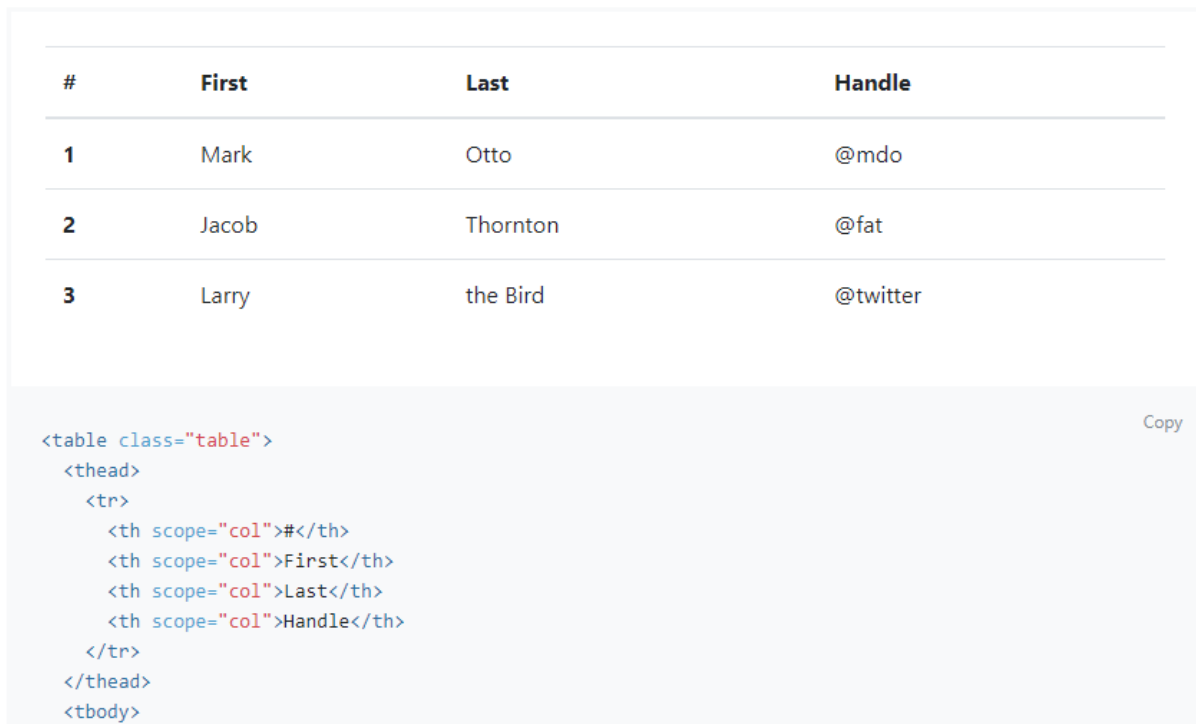


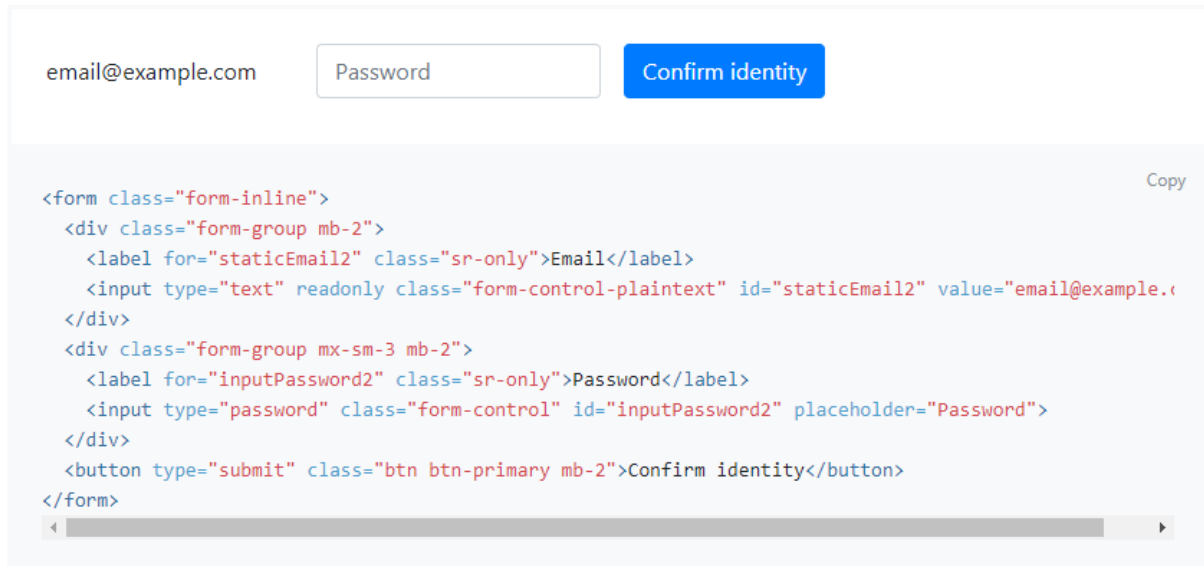
Рисунок 2.5

6. Форми — класи для оформлення форм і деяких подій, що відбуваються з ними (рисунок 2.6). Детальна інформація про використання форм доступна за посиланням:

<https://getbootstrap.com/docs/4.1/components/forms/>

7. Навігація – класи оформлення для вкладок, сторінок, меню і панелі інструментів (рисунок 2.7). Детальна інформація про використання навігації доступна за посиланням:

<https://getbootstrap.com/docs/4.1/components/navs/>



The screenshot displays a Bootstrap form with the class `form-inline`. It contains two input fields: a text input for email with the value `email@example.com` and a password input with the placeholder `Password`. Both inputs are part of `form-group` containers with the class `mb-2`. A blue button with the text `Confirm identity` is positioned to the right of the password input. Below the form, the corresponding HTML code is shown in a light blue box with a 'Copy' button.

```
<form class="form-inline">
  <div class="form-group mb-2">
    <label for="staticEmail2" class="sr-only">Email</label>
    <input type="text" readonly class="form-control-plaintext" id="staticEmail2" value="email@example.com">
  </div>
  <div class="form-group mx-sm-3 mb-2">
    <label for="inputPassword2" class="sr-only">Password</label>
    <input type="password" class="form-control" id="inputPassword2" placeholder="Password">
  </div>
  <button type="submit" class="btn btn-primary mb-2">Confirm identity</button>
</form>
```

Рисунок 2.6



The screenshot shows a Bootstrap navigation menu with the class `nav`. It contains four items: 'Active' (highlighted in blue), 'Link' (in blue), 'Link' (in blue), and 'Disabled' (in gray). Each item is a link with the class `nav-link` and the href attribute set to `#`. Below the menu, the corresponding HTML code is shown in a light blue box with a 'Copy' button.

```
<ul class="nav">
  <li class="nav-item">
    <a class="nav-link active" href="#">Active</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">Link</a>
  </li>
  <li class="nav-item">
    <a class="nav-link disabled" href="#">Disabled</a>
  </li>
</ul>
```

Рисунок 2.7

8. Сповіщення – оформлення діалогових вікон, підказок і спливаючих вікон (рисунок 2.8). Детальна інформація про використання сповіщень доступна за посиланням:

<https://getbootstrap.com/docs/4.1/components/alerts/>



Рисунок 2.8

2.3 Створення web-сторінок

2.3.1 Створення сторінки входу в додаток

Приклад найпростішої сторінки входу в додаток, створеної за допомогою фреймворка Bootstrap, представлений на малюнку 2.9:

User name

Password

Рисунок 2.9

Для створення даної сторінки була використана горизонтальна форма, шляхом використання класу `.row` для груп форми, а також класів `.col-*-*` для налаштування ширини елементів форми і їх підписів. Необхідною умовою є додавання класу `.col-form-label` до тегам `<label>` для того, щоб вони були вирівняні вертикально по центру, щодо асоційованих з ними елементів форми.

Приклад коду, використаного для створення форми входу в додаток (рисунок 2.9):

```
<form class="col-md-6 offset-md-3 mt-5" method="post">
  <div class="form-group row">
    <label for="inputEmail3" class="col-sm-2 col-form-label">User name</label>
    <div class="col-sm-10">
      <input type="text" class="form-control" id="inputUsername" name="username"
placeholder="User name">
    </div>
  </div>
  <div class="form-group row">
    <label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="inputPassword" name="password"
placeholder="Password">
    </div>
  </div>
  <div class="form-group row">
    <div class="col-sm-10">
      <input type="submit" class="btn btn-primary" value="Sign in" name="signin">
    </div>
  </div>
</form>
```

2.3.2 Робота з одними сеансами

При роботі із застосунком користувач запускає його, виконує деякі дії, і потім закриває. Це називається призначеним для користувача сеансом або сесією.

Однак web-сервер «не знає» нічого про користувача, оскільки протокол HTTP не зберігає стан. Дану проблему вирішують сеансу змінні шляхом збереження інформації про користувача, роблячи її доступною для використання на декількох сторінках (наприклад, ім'я користувача і т.д.). За замовчуванням, сеансу змінні зберігаються до тих пір, поки користувач не закриє браузер.

Для початку сеансу використовується функція `session_start()`. Сеансові змінні зберігаються в глобальному масиві `$_SESSION`. Приклад на малюнку 2.10 демонструє початок сесії і установку деяких сеансових змінних:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favourite"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Рисунок 2.10

Необхідно відзначити, що сеансу змінні не передаються окремо на кожну сторінку. Замість цього, вони витягуються з сеансу (за допомогою глобального масиву `$_SESSION`), що відкривається на початку сторінки за допомогою функції `session_start()`. Приклад на малюнку 2.11 демонструє початок сесії і звернення до сеансових змінних:

```

<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . "<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>

```

Рисунок 2.11

Для видалення всіх сеансових змінних і знищення сесії, використовуються функції `session_unset()` і `session_destroy()`. Приклад на рисунку 2.12 демонструє знищення сесії:

```

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

```

Рисунок 2.12

З урахуванням необхідності використання користувальницьких сеансів в роботі створюваного застосунка, вихідний код сторінки входу в додаток (`login.php`) буде виглядати наступним чином:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Util/MySQLConnectionUtil.php');

if (isset($_SESSION['username']))
{
    $controller->redirect($_SESSION['username']);
}

if (isset($_POST['signin']))
{

```

```

$_SESSION['username'] = $_POST['username'];
$_SESSION['password'] = $_POST['password'];

try
{
    @MySQLConnectionUtil::getConnection();

    $controller->redirect($_SESSION['username']);
}
catch (Exception $e)
{
    session_unset();
    session_destroy();

    ?><div class="alert alert-danger" role="alert"><?= $e->getMessage() ?></div><?php
}
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Contracts</title>
    <link
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
        integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
        crossorigin="anonymous">
    </head>
<body>
    <form class="col-md-6 offset-md-3 mt-5" method="post" id="loginForm">
        <div class="form-group row">
            <label for="inputEmail3" class="col-sm-2 col-form-label">User name</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="inputUsername" name="username"
placeholder="User name">
            </div>
        </div>
        <div class="form-group row">
            <label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>
            <div class="col-sm-10">
                <input type="password" class="form-control" id="inputPassword" name="password"
placeholder="Password">
            </div>
        </div>
        <div class="form-group row">
            <div class="col-sm-10">
                <input type="submit" class="btn btn-primary" value="Sign in" name="signin">
            </div>
        </div>
    </form>
    <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
</body>
</html>

```

Для коректної роботи даної сторінки, необхідно створити клас Controller, помістивши відповідний файл в каталог App/Controller.php. Вихідний код даного класу має такий вигляд:

```

<?php
class Controller
{
    private $pages;

    public function __construct()
    {
        $this->pages = array(
            'login' => 'login.php',
            'supply_manager' => 'contracts.php'
        );
    }
}

```

```

    }

    public function redirect($path)
    {
        header("location: {$this->pages[$path]}");
    }
}

$controller = new Controller();
?>

```

Відповідно, для виходу з програми, необхідно створити сторінку (logout.php), що містить виклики функцій знищення призначеного для користувача сеансу:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');

session_unset();
session_destroy();

$controller->redirect('login');
?>

```

Код стартової сторінки застосунка (index.php) буде містити перевірку наявності сеансу, і направляти користувача на сторінку входу в додаток або на головну сторінку відповідно:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');

if (isset($_SESSION['username']))
{
    $controller->redirect($_SESSION['username']);
}
else
{
    $controller->redirect('login');
}
?>

```

2.3.3 Створення сторінки роботи з даними

Як приклад сторінки роботи з даними, розглядається сторінка роботи співробітника відділу постачання з інформацією про договори (рисунок 2.13):

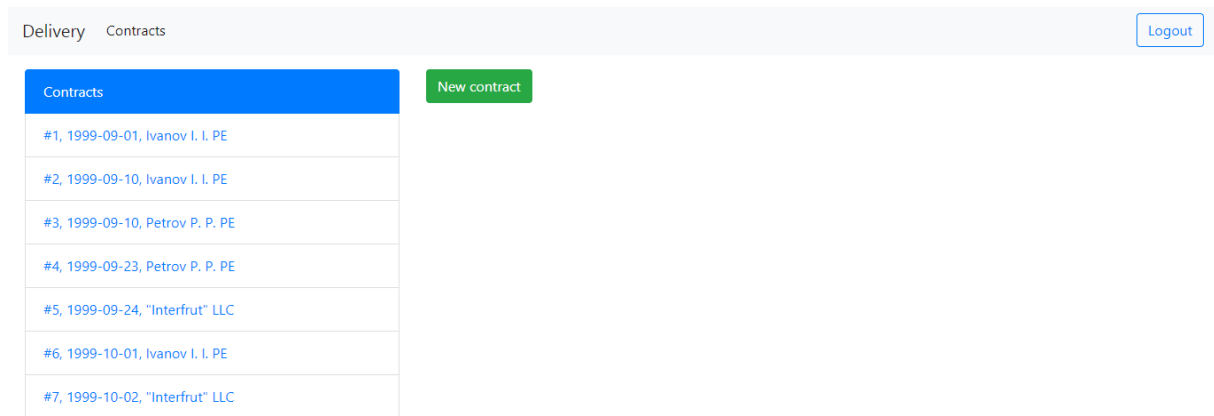


Рисунок 2.13

Для створення сторінки використовувався компонент Bootstrap – навігаційна панель, що включає в себе підкомпоненти, такі як посилання на сторінки і форма виходу з програми.

Приклад коду, використаного для створення навігаційної панелі (рисунок 2.13):

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href=".">Delivery</a>
  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="./contracts.php">Contracts</a>
      </li>
    </ul>
    <form class="form-inline my-2 my-lg-0" action="logout.php" method="post">
      <button class="btn btn-outline-primary my-2 my-sm-0"
type="submit">Logout</button>
    </form>
  </div>
</nav>
```

Для демонстрації переліку договорів використовується інший компонент Bootstrap – список. Приклад коду, використаного для створення переліку договорів (рисунок 2.13):

```
<ul class="list-group">
  <li class="list-group-item active">Contracts</li>
  <?php foreach ($service->getAllContracts() as $contract) { ?>
    <li class="list-group-item">
      <a href="contracts.php?details=<?= $contract->getNumber() ?>"
        #<?= $contract->getNumber() ?>, <?= $contract->getAgreed() ?>, <?= $contract-
>getSupplier() ?>
      </a></li>
    <?php } ?>
  </ul>
```

Для виконання дій, таких, наприклад, як створення нового договору або редагування/видалення договору, використовуються відповідні компоненти Bootstrap – кнопки і форми.

Приклад коду, використаного для створення кнопки переходу до створення нового договору (рисунок 2.13):

```
<a class="btn btn-success" href="#" role="button">New contract</a>
```

При переході по посиланню, якої є кожен пункт переліку договорів, завантажуються детальна інформація про договір, представлена у вигляді форми з відповідними кнопками редагування / видалення договору (рисунок 2.14):

The screenshot shows a web interface with a top navigation bar containing 'Delivery' and 'Contracts' tabs, and a 'Logout' button. Below the navigation bar, there is a table of contracts on the left and a details panel on the right. The table has a blue header 'Contracts' and lists seven items with IDs, dates, and names. The details panel shows fields for 'Contract number', 'Contract date', 'Supplier', 'Title', and 'Note'. The 'Note' field contains the text 'Invoice 74 from 9/11/99'. At the bottom of the details panel are 'Edit' and 'Remove' buttons.

Contracts	Contract number
#1, 1999-09-01, Ivanov I. I. PE	5
#2, 1999-09-10, Ivanov I. I. PE	Contract date
#3, 1999-09-10, Petrov P. P. PE	1999-09-24
#4, 1999-09-23, Petrov P. P. PE	Supplier
#5, 1999-09-24, "Interfrut" LLC	"Interfrut" LLC
#6, 1999-10-01, Ivanov I. I. PE	Title
#7, 1999-10-02, "Interfrut" LLC	Contract 5

Note: Invoice 74 from 9/11/99

Edit Remove

Рисунок 2.14

Приклад коду, використаного для створення форми перегляду інформації про договір (рисунок 2.14):

```
<form>
  <div class="form-group row">
    <label for="contractNumber" class="col-sm-2 col-form-label">Contract number</label>
    <div class="col-sm-10">
      <input type="text" readonly class="form-control-plaintext" id="contractNumber"
value="<%= $contract->getNumber() ?>">
    </div>
  </div>
  <div class="form-group row">
    <label for="contractDate" class="col-sm-2 col-form-label">Contract date</label>
    <div class="col-sm-10">
```

```

        <input type="text" readonly class="form-control-plaintext" id="contractDate"
value="<?= $contract->getAgreed() ?>">
    </div>
</div>
<div class="form-group row">
    <label for="supplier" class="col-sm-2 col-form-label">Supplier</label>
    <div class="col-sm-10">
        <input type="text" readonly class="form-control-plaintext" id="supplier"
value="<?= htmlspecialchars($contract->getSupplier()) ?>">
    </div>
</div>
<div class="form-group row">
    <label for="title" class="col-sm-2 col-form-label">Title</label>
    <div class="col-sm-10">
        <input type="text" readonly class="form-control-plaintext" id="title" value="<?=
htmlspecialchars($contract->getTitle()) ?>">
    </div>
</div>
<div class="form-group row">
    <label for="note" class="col-sm-2 col-form-label">Note</label>
    <div class="col-sm-10">
        <textarea class="form-control" readonly rows="5" id="note"><?=
htmlspecialchars($contract->getNote()) ?></textarea>
    </div>
</div>
</form>
<a class="btn btn-warning" href="#" role="button">Edit</a>
<a class="btn btn-danger" href="#" role="button">Remove</a>

```

Для розташування переліку договорів і форми перегляду докладної інформації по кожному договору (рисунк 2.14), використовуються відповідні класи `.col-*` фреймворка Bootstrap, призначені для створення адаптивної сітки.

Остаточний варіант вихідного коду сторінки роботи з договорами (contracts.php):

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Domain/Contract.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Repository/MySQLContractRepository.php');
;
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Service/ContractService.php');

if (!isset($_SESSION['username']))
{
    $controller->redirect('login');
}

$repository = new MySQLContractRepository();
$service = new ContractService($repository);
?>
<!DOCTYPE html>
<html>
<head>
    <title>Contracts</title>
    <link
rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdKnLPMO"
crossorigin="anonymous">
</head>

```

```

<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="/">Delivery</a>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="./contracts.php">Contracts</a>
                </li>
            </ul>
            <form class="form-inline my-2 my-lg-0" action="logout.php" method="post">
                <button class="btn btn-outline-primary my-2 my-sm-0"
type="submit">Logout</button>
            </form>
        </div>
    </nav>
    <div class="row my-3 mx-1">
        <div class="col-4">
            <ul class="list-group">
                <li class="list-group-item active">Contracts</li>
                <?php foreach ($service->getAllContracts() as $contract) { ?>
                    <li class="list-group-item">
                        <a href="contracts.php?details=<? $contract->getNumber() ?>"
#<? $contract->getNumber() ?>, <? $contract->getAgreed() ?>, <?
$contract->getSupplier() ?>
                        </a>
                    </li>
                <?php } ?>
            </ul>
        </div>
        <div class="col-8">
            <?php
            if (isset($_GET['details']))
            {
                try
                {
                    $contract = @$service->getContractByNumber($_GET['details']);
                    ?>
                    <form>
                        <div class="form-group row">
                            <label for="contractNumber" class="col-sm-2 col-form-
label">Contract number</label>
                            <div class="col-sm-10">
                                <input type="text" readonly class="form-control-plaintext"
id="contractNumber" value="<? $contract->getNumber() ?>">
                            </div>
                        </div>
                        <div class="form-group row">
                            <label for="contractDate" class="col-sm-2 col-form-label">Contract
date</label>
                            <div class="col-sm-10">
                                <input type="text" readonly class="form-control-plaintext"
id="contractDate" value="<? $contract->getAgreed() ?>">
                            </div>
                        </div>
                        <div class="form-group row">
                            <label for="supplier" class="col-sm-2 col-form-
label">Supplier</label>
                            <div class="col-sm-10">
                                <input type="text" readonly class="form-control-plaintext"
id="supplier" value="<? htmlspecialchars($contract->getSupplier()) ?>">
                            </div>
                        </div>
                        <div class="form-group row">
                            <label for="title" class="col-sm-2 col-form-label">Title</label>
                            <div class="col-sm-10">
                                <input type="text" readonly class="form-control-plaintext"
id="title" value="<? htmlspecialchars($contract->getTitle()) ?>">
                            </div>
                        </div>
                        <div class="form-group row">
                            <label for="note" class="col-sm-2 col-form-label">Note</label>
                            <div class="col-sm-10">
                                <textarea class="form-control" readonly rows="5" id="note"><?
htmlspecialchars($contract->getNote()) ?></textarea>
                            </div>
                        </div>
                    </form>
                }
            }
        }
    }

```



```

        <a class="btn btn-warning" href="#" role="button">Edit</a>
        <a class="btn btn-danger" href="#" role="button">Remove</a>
    <?php
    }
    catch (Exception $e)
    {
        ?><div class="alert alert-danger" role="alert"><?= $e->getMessage()
?></div><?php
    }
    }
    else
    {
        ?><a class="btn btn-success" href="#" role="button">New contract</a><?php
    }
    ?>
</div>
</div>
</body>
</html>

```

Помилки, що виникають при роботі користувача із застосунком, демонструються за допомогою відповідних компонентів Bootstrap – оповіщень (рисунок 2.15):

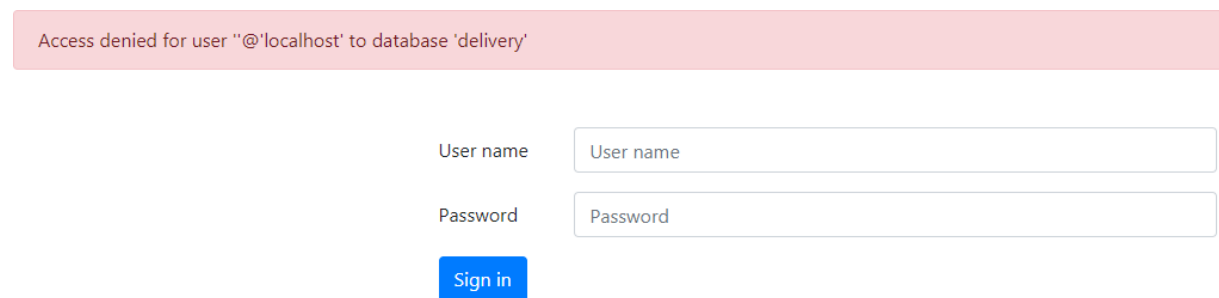


Рисунок 2.15

При виконанні роботи необхідно створити (з використанням Bootstrap) відповідні сторінки, згідно заданої предметної області. Рекомендується спочатку створити статичні прототипи сторінок і узгодити їх з викладачем. Потім, на основі створених прототипів, можна формувати динамічні PHP сторінки.

2.4 Асинхронний обмін даними з web-сервером

2.4.1 Знайомство з технологією AJAX

AJAX (Asynchronous Javascript and XML) – це підхід до побудови інтерактивних користувацьких інтерфейсів веб-застосунків, що полягає в

«фоновому» обміні даними браузера з web-сервером. В результаті, при оновленні даних web-сторінка не перезавантажується повністю, і веб-застосування стають швидше і зручніше.

У класичній моделі веб-застосування (рисунок 2.16):

- 1) користувач заходить на веб-сторінку і натискає на який-небудь її елемент;
- 2) браузер формує і відправляє запит серверу;
- 3) у відповідь сервер генерує абсолютно нову веб-сторінку і відправляє її браузеру і т. д., після чого браузер повністю перезавантажує всю сторінку.

При використанні AJAX (рисунок 2.16):

- 1) користувач заходить на веб-сторінку і натискає на який-небудь її елемент;
- 2) скрипт (на мові JavaScript) визначає, яка інформація необхідна для оновлення сторінки;
- 3) браузер відправляє відповідний запит на сервер;
- 4) сервер повертає тільки ту частину документа, на яку прийшов запит.
- 5) скрипт вносить зміни з урахуванням отриманої інформації (без повного перезавантаження сторінки).

перевагами даної технології є:

- 1) економія трафіку;
- 2) зменшення навантаження на сервер;
- 3) прискорення реакції інтерфейсу;
- 4) практично безмежні можливості для інтерактивної обробки.

Однак технологія AJAX має також і недоліки:

- 1) відсутність інтеграції зі стандартними інструментами браузера;
- 2) динамічно завантажуваний вміст недоступний пошуковикам;
- 3) старі методи обліку статистики сайтів стають неактуальними;

- 4) ускладнення проекту;
- 5) потрібно мати включений JavaScript в браузері;
- 6) проблеми з відображенням нестандартних кодувань;
- 7) низька швидкість при грубому програмуванні;
- 8) погана поведінка при ненадійному з'єднанні;
- 9) ризик фабрикації запитів іншими сайтами.

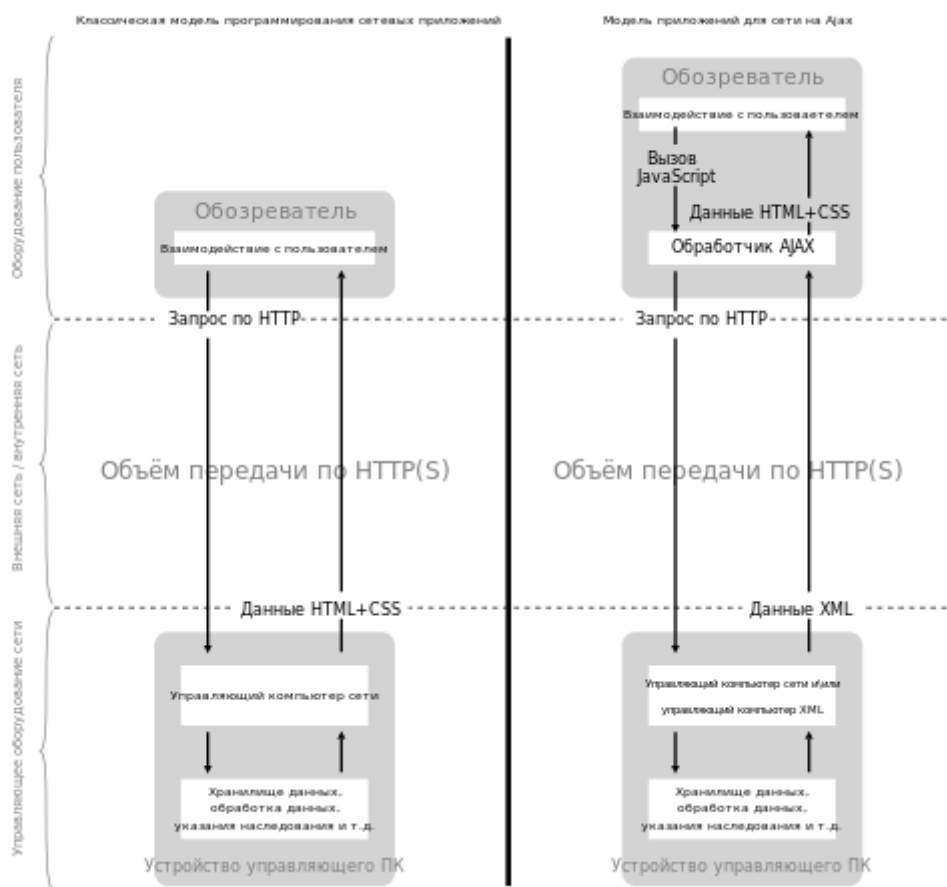


Рисунок 2.16

2.4.2 Використання AJAX на сторінці входу в додаток

Для використання асинхронного обміну даними з сервером при вході в додаток, необхідно додати нові елементи в код сторінки login.php:

```

<div id="loginAlert" hidden>
  <div class="alert alert-danger" role="alert" id="loginErrorMessage"></div>
</div>
<div class="mt-5" id="signIn" hidden>
  <center>
    <a class="btn btn-success" href="." role="button">Continue as <span
id="continueUsername"></span></a>
  </center>
</div>

```

Дані блоки будуть використовуватися для виведення інформації, отриманої від сервера (повідомлення про помилку або успішної авторизації).

В каталозі Web необхідно створити підкаталог AJAX, в якому буде поміщений файл ajax_login.php, призначений для обробки асинхронних запитів до сервера:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Util/MySQLConnectionUtil.php');

$_SESSION['username'] = $_GET['username'];
$_SESSION['password'] = $_GET['password'];

try
{
    @MySQLConnectionUtil::getConnection();

    echo json_encode(array('status' => 'true'));
}
catch (Exception $e)
{
    session_unset();
    session_destroy();

    echo json_encode(array('status' => 'false', 'message' => $e->getMessage()));
}

?>

```

Для спрощення реалізації виконання запитів до сервера буде використовуватися бібліотека jQuery:

```

$("#loginForm").submit(function(event) {
    var username = $("#inputUsername").val();
    var password = $("#inputPassword").val();

    $.getJSON("./AJAX/ajax_login.php?username=" + username + "&password=" + password,
function(result) {
    if (result.status == 'true') {
        $("#signIn").removeAttr("hidden");

        $("#continueUsername").text(username);

        $("#loginForm").hide();
        $("#loginAlert").hide();
    }
    });
});

```

```

    } else {
        $("#loginAlert").removeAttr("hidden");

        $("#loginErrorMessage").text(result.message);
    }
});

return false;
});

```

Даний код необхідно помістити в файл login.js за адресою Web/js. Після того, як JavaScript файл буде створений, його необхідно підключити в файлі сторінки входу в додаток - login.php. Приклад остаточного варіанту вихідного коду сторінки входу в додаток:

```

<?php
session_start();

require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Controller.php');
require_once($_SERVER['DOCUMENT_ROOT'].'/Delivery/App/Util/MySQLConnectionUtil.php');

if (isset($_SESSION['username']))
{
    $controller->redirect($_SESSION['username']);
}

if (isset($_POST['signin']))
{
    $_SESSION['username'] = $_POST['username'];
    $_SESSION['password'] = $_POST['password'];

    try
    {
        @MySQLConnectionUtil::getConnection();

        $controller->redirect($_SESSION['username']);
    }
    catch (Exception $e)
    {
        session_unset();
        session_destroy();

        ?><div class="alert alert-danger" role="alert"><? $e->getMessage() ?></div><?php
    }
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Contracts</title>
    <link
        rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
        integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
        crossorigin="anonymous">
</head>
<body>
    <div id="loginAlert" hidden>
        <div class="alert alert-danger" role="alert" id="loginErrorMessage"></div>
    </div>
    <div class="mt-5" id="signIn" hidden>
        <center>
            <a class="btn btn-success" href="." role="button">Continue as <span
id="continueUsername"></span></a>
        </center>
    </div>
    <form class="col-md-6 offset-md-3 mt-5" method="post" id="loginForm">
        <div class="form-group row">

```

```

        <label for="inputEmail3" class="col-sm-2 col-form-label">User name</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" id="inputUsername" name="username"
placeholder="User name">
        </div>
    </div>
    <div class="form-group row">
        <label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>
        <div class="col-sm-10">
            <input type="password" class="form-control" id="inputPassword" name="password"
placeholder="Password">
        </div>
    </div>
    <div class="form-group row">
        <div class="col-sm-10">
            <input type="submit" class="btn btn-primary" value="Sign in" name="signin">
        </div>
    </div>
</form>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="js/login.js"></script>
</body>
</html>

```

Використання технології AJAX є обов'язковою вимогою при виконанні роботи, при цьому використання бібліотеки jQuery не є обов'язковим (допускається використовувати «чистий» JavaScript або інші бібліотеки/фреймворки). При реалізації асинхронного обміну даними з сервером, необхідно врахувати можливі недоліки такого підходу.

Зафіксувати результати виконання роботи в системі управління версіями Git. Злити гілку pages в master після закінчення роботи. При необхідності, вирішити конфлікти, що виникнули.

Створити віддалений репозиторій Git, використовуючи один з безкоштовних хостингів (GitHub, GitLab і т.д.). Опублікувати результати виконання робіт в віддаленому репозиторії, використовуючи команду git push. Ознайомитися з особливостями роботи з віддаленими репозиторіями можна за посиланням:

<https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes>

2.5 Вимоги до звіту

- 1) коротко описати основні етапи виконання роботи;
- 2) привести зовнішній вигляд створених сторінок;

3) привести вихідний код створених сторінок, сценаріїв PHP і Java Script;

4) продемонструвати отримані результати у вигляді ефектів у програмному забезпеченні, надати історію змін в системі управління версіями Git, посилання на віддалений репозиторій.

2.6 Питання для самоперевірки

1. Що таке Bootstrap?
2. Основні інструменти Bootstrap?
3. Які інструменти Bootstrap були використані при виконанні роботи?
4. Яким чином вирішується проблема зберігання інформації про користувача, який працює із застосунком?
5. Що таке сеанс (сесія)? Як створити сеанс?
6. Яким чином можна зберегти дані, використовуючи сеанс?
7. Яким чином можна отримати доступ до сеансових змінних?
8. Як знищити сесію і видалити всі сеансу змінні?
9. Що таке HTTP? Назвіть методи HTTP їх основні особливості і відмінності.
10. Які методи HTTP були використані при виконанні роботи?
11. Що таке AJAX?
12. Основні переваги і недоліки технології AJAX.
13. Класична модель web-застосунків.
14. Модель застосунка при використанні AJAX.
15. Що таке jQuery?
16. За допомогою яких засобів була реалізована технологія AJAX при виконанні роботи?