

# **Fullstack web application development using MySQL, PHP, HTML, CSS and JavaScript**

## CONTENT

1 CREATION OF DATABASE USING MYSQL DBMS. WORKING WITH MYSQL DATABASE IN PHP LANGUAGE.....	4
1.1 Preparation for work.....	4
1.2 Creating a database.....	4
1.2.1 Starting and configuring XAMPP .....	4
1.2.2 Working with MySQL in phpMyAdmin .....	7
1.2.3 Creation of user accounts .....	11
1.2.4 Creating triggers for tables .....	13
1.2.5 Creation of stored procedures and functions.....	16
1.3 Working with a database in PHP .....	20
1.3.1 Functions for working with the database .....	20
1.3.2 Repository design pattern.....	22
1.3.3 Service Layer design pattern .....	27
1.4 Requirements for the report.....	31
1.5 Questions for self-testing .....	31
2 CREATION OF WEB APPLICATIONS USING THE BOOTSTRAP FRAMEWORK. USING AJAX TECHNOLOGY FOR ASYNCHRONOUS DATA EXCHANGE WITH A WEB SERVER .....	33
2.1 Preparation for work.....	33
2.2 Getting to know the Bootstrap framework.....	33
2.3 Creation of web pages .....	38
2.3.1 Creating an application login page.....	38
2.3.2 Working with single sessions.....	39

2.3.3 Creating a page for working with data .....	43
2.4 Asynchronous data exchange with the web server .....	48
2.4.1 Introduction to AJAX technology .....	48
2.4.2 Using AJAX on the application login page.....	50
2.5 Report requirements .....	53
2.6 Questions for self-testing .....	53

# 1 CREATION OF DATABASE USING MYSQL DBMS. WORKING WITH MYSQL DATABASE IN PHP LANGUAGE

## 1.1 Preparation for work

1. Create a new branch in the Git version control system and name it storage. While on the created branch, in the working directory (for example, D:\GIT\_PRACTICE) create a subdirectory in which all further work will be performed (for example, D:\GIT\_PRACTICE\db).

2. Download and install XAMPP - a cross-platform web server assembly containing the Apache HTTP server, the MySQL relational database management system, the PHP language interpreter and a large number of additional libraries that allow you to run a full-fledged web server.

## 1.2 Creating a database

### 1.2.1 Starting and configuring XAMPP

After the XAMPP server is installed, it is necessary to start the server control panel and make sure that there are no errors (Figure 1.1):

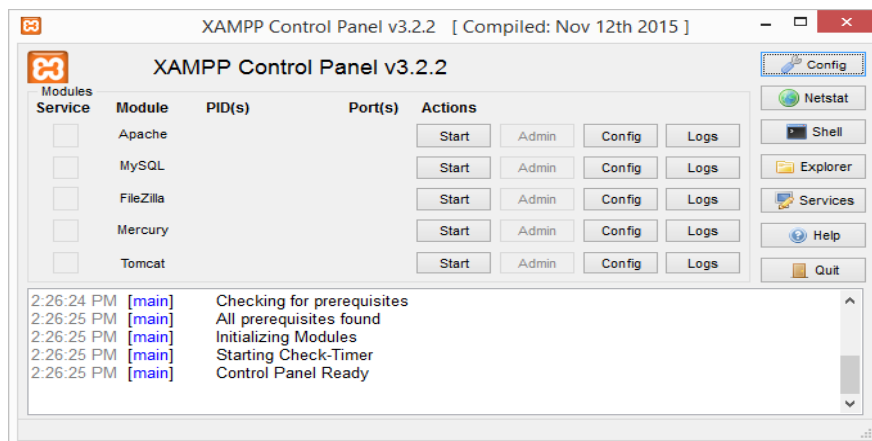


Figure 1.1

Most of the errors that occur when starting XAMPP or its modules (Apache, MySQL, etc.) are related to the fact that the ports used are already occupied by some programs or services.

To solve this problem, you can disable the programs and/or services occupying the necessary ports, or reconfigure them. Another way to solve the problem is to debug the ports of XAMPP services. To do this, you need to open the Config -> Service and Port Settings window and specify free ports for use by various XAMPP services (Figure 1.2).

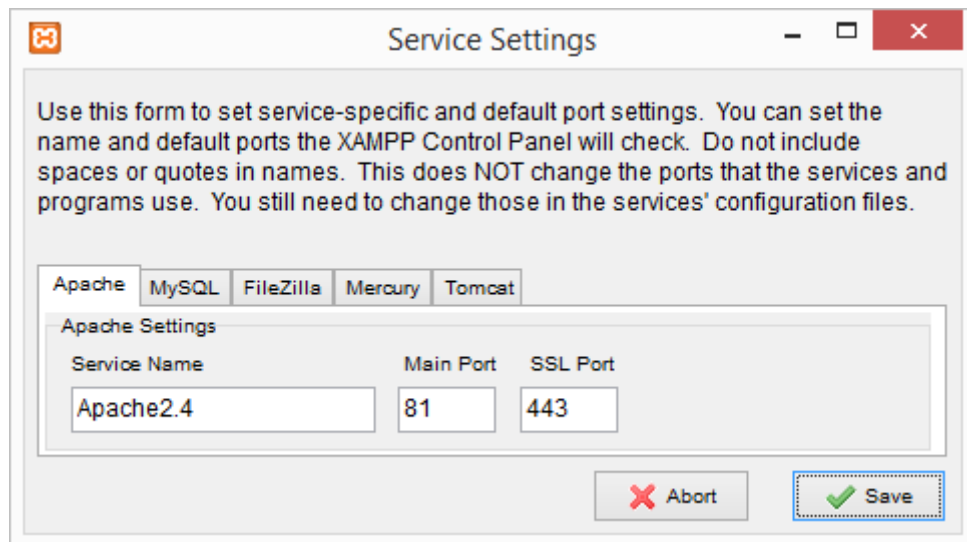
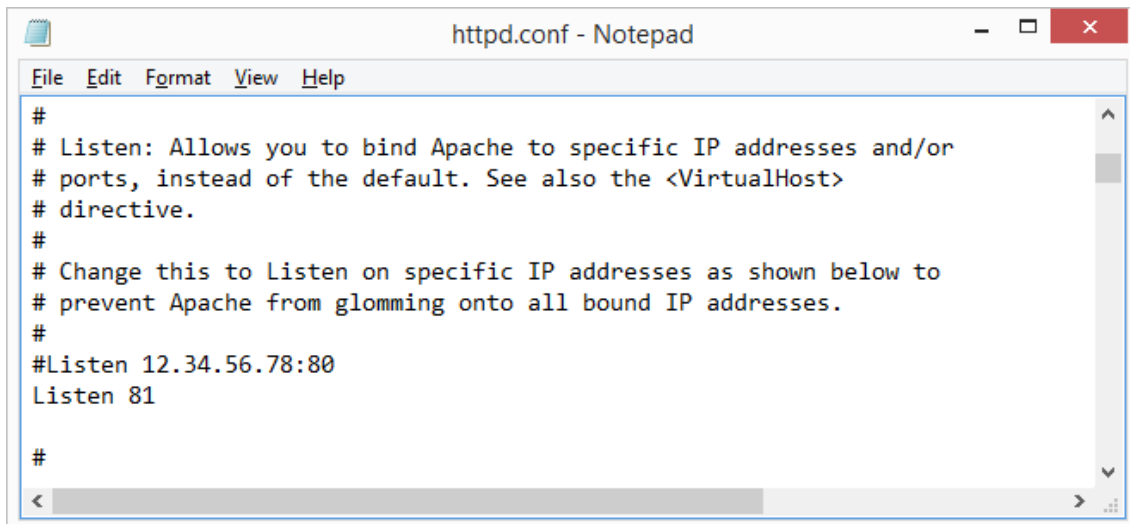


Figure 1.2

Alternatively, you may need to manually configure Apache to bind to a specific port. To do this, select the Config action for the Apache module, select the Apache item (httpd.conf) and replace the port number in the line (Figure 1.3):

```
Listen XX # where XX is the default port
```



```
#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 81

#
```

Figure 1.3

After the problems are solved, you need to start the Apache and MySQL modules. In case of successful launch, the names of the modules will be highlighted in green, and the PID (s) and Port (s) columns will indicate the identifiers of the running processes and the numbers of the occupied ports, respectively (Figure 1.4):

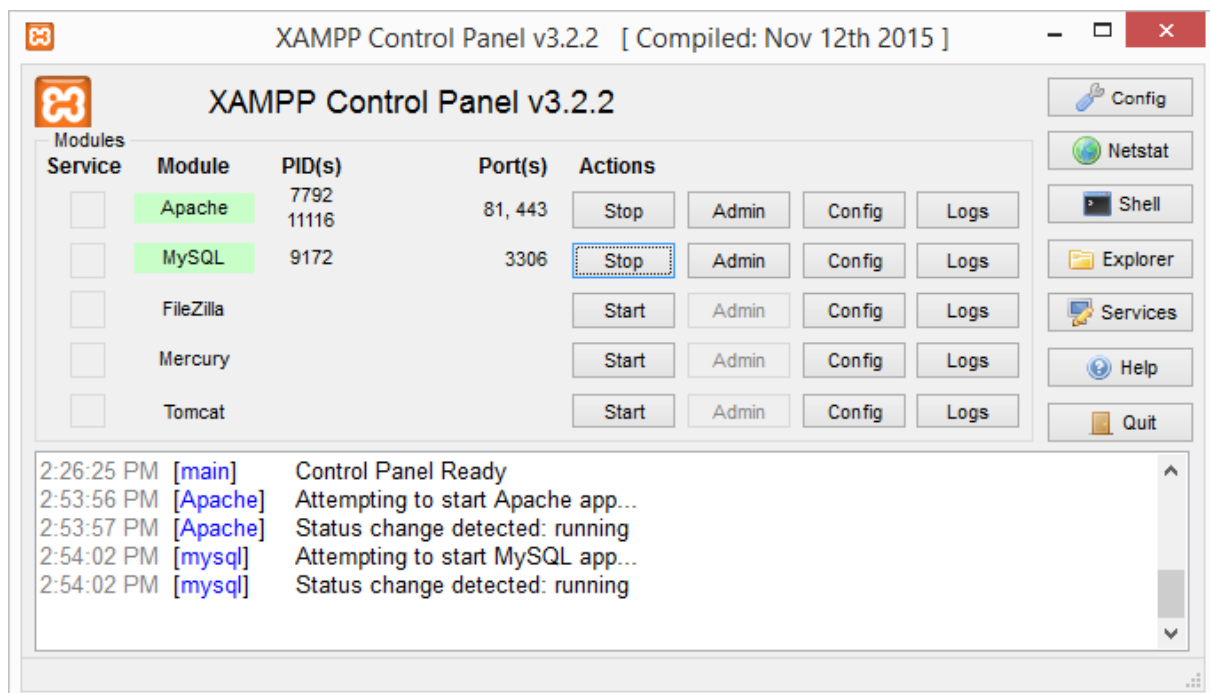


Figure 1.4

### 1.2.2 Working with MySQL in phpMyAdmin

The phpMyAdmin application is a web interface for the administration of the MySQL database management system (DBMS). The application allows you to administer the MySQL server, run SQL commands and view the contents of tables and databases through a browser. The application is very popular among web developers, as it allows you to manage the MySQL DBMS without directly entering SQL commands, providing a friendly interface.

To start phpMyAdmin, you need to select the Admin action for the MySQL module in the XAMPP control window or go to the address

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

after Apache and MySQL are successfully started (Figure 1.5):

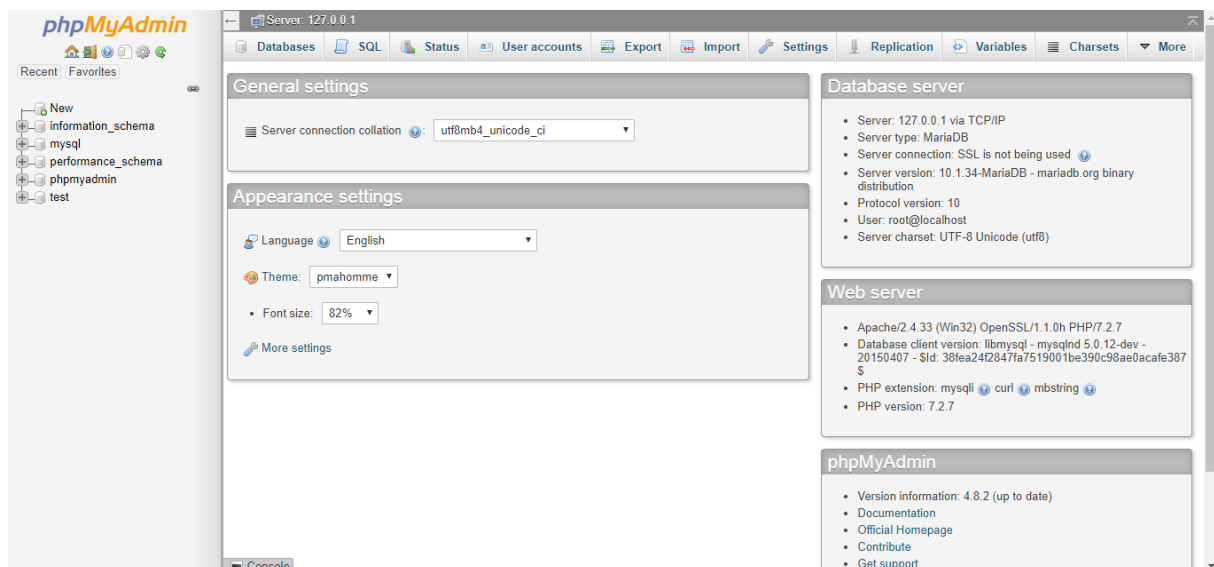


Figure 1.5

To create a new database, select New above the list of available databases. As a result, the database creation form will be opened (Figure 1.6):

# Databases

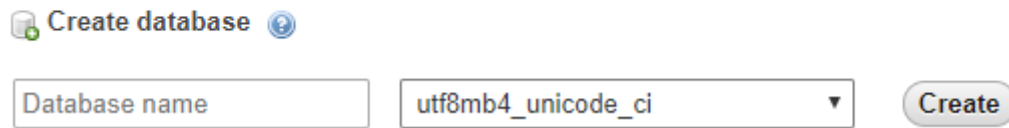


Figure 1.6

In the Database name field, enter the name of the database to be created (for example, delivery) and click the Create button. How to set the encoding to utf8mb4\_unicode\_ci. The utf8mb4 encoding must be used instead of utf8 starting with MySQL version 5.5.3, as the utf8 encoding is considered deprecated. Currently, for MySQL databases and tables, it is recommended to use the utf8mb4\_unicode\_ci encoding, which is free from the disadvantages associated with collation in certain languages.

After creating the database, a window will open for viewing the database structure, which does not currently contain any tables (Figure 1.7):

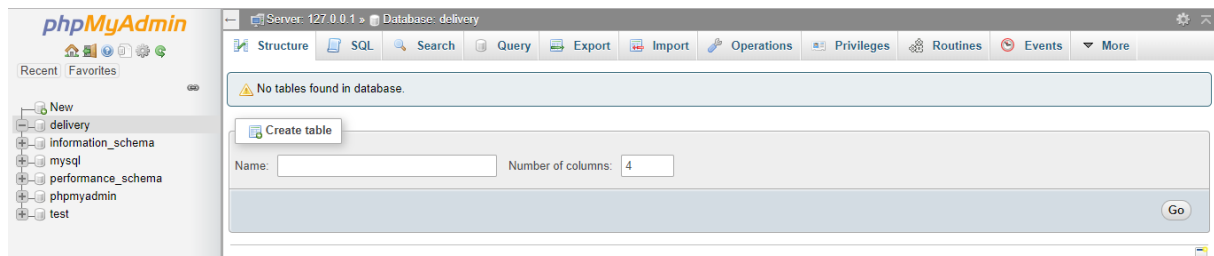


Figure 1.7

To create the first table in the database, it is suggested to enter the name of the new table in the Name field, and also specify the number of columns in the Number of columns field (Figure 1.7).

For the considered (as an example) subject area, you need to create a supplier table with 3 columns (Figure 1.8):



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

Figure 1.8

In the form, specify the names of the columns in sequence: id, name, address. As the type, specify the INT type for the id column, and the VARCHAR type for the name and company columns. For the name and company columns, in the Length / Values field, specify the maximum length of lines in characters - 50 and 100, respectively. For the id column, enter the Index PRIMARY field, and put a check mark in the A\_I (Auto Increment) field.

After pressing the Save button, the structure of the created table and its columns will be displayed (Figure 1.9):

Table structure

Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	<b>id</b>	int(11)			No	None		AUTO_INCREMENT	Change  Drop  More
<input type="checkbox"/> 2	<b>name</b>	varchar(50)	utf8mb4_unicode_ci		No	None			Change  Drop  More
<input type="checkbox"/> 3	<b>address</b>	varchar(100)	utf8mb4_unicode_ci		No	None			Change  Drop  More

Figure 1.9

You can also create a table in phpMyAdmin using the SQL command. To do this, go to the SQL tab and enter the appropriate command (Figure 1.10):

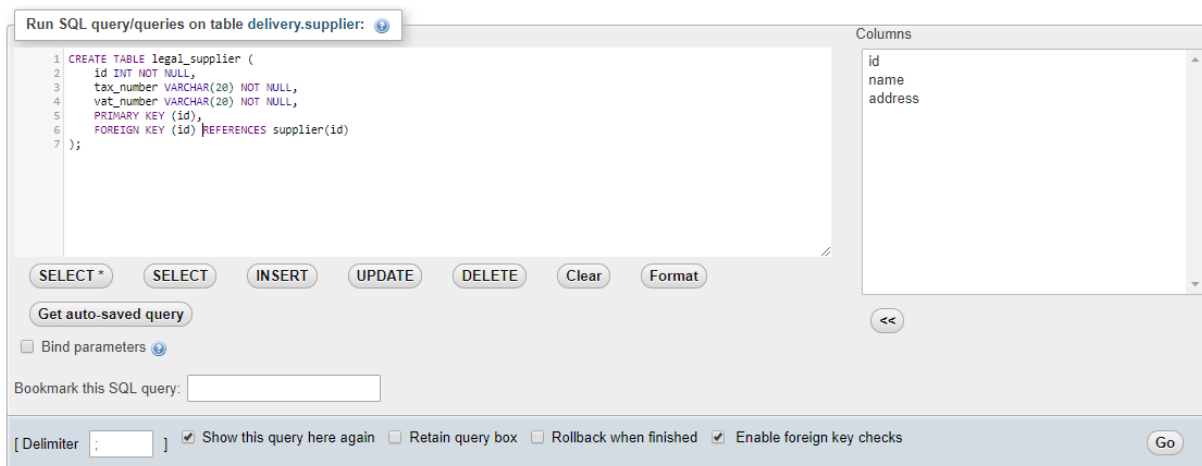


Figure 1.10

As a result of executing this command by pressing the Go button, the result of successful execution or a list of errors will be displayed (Figure 1.11):

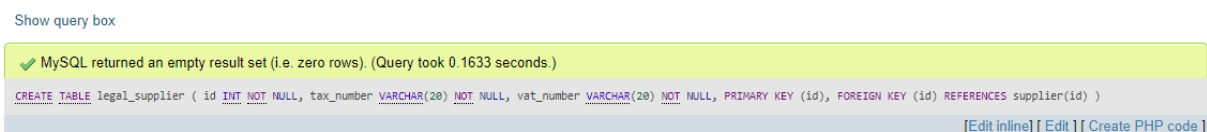


Figure 1.11

All necessary tables of the database under development must be created. For this, you can use both table creation forms and SQL commands. However, to reduce the possibility of errors when creating database tables, it is recommended to use SQL commands. Before starting to create tables, you should familiarize yourself with the main data types of the MySQL DBMS by following the link:

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

The structure of the created database (tables and relationships between them) can be viewed by going to the Designer tab. The possible structure of the created database is shown in figure (1.12):

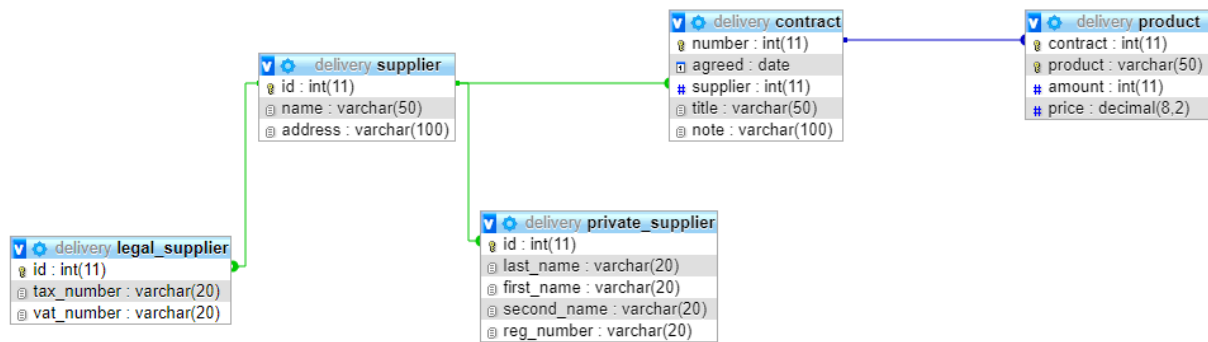


Figure 1.12

To check the correctness and functionality, the created database tables must be filled with test data sets.

### 1.2.3 Creation of user accounts

As an example, we consider the creation of a user account - an employee of the supply department. To do this, go to the Privileges tab and select Add user account (Figure 1.13):

#### Add user account

The 'Add user account' form is divided into two sections: 'Login Information' and 'Database for user account'.

**Login Information:**

- User name: Use text field: supply\_mgr
- 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.

Figure 1.13

In the fields User name and Password, you must enter the username supply\_manager and the password supply, respectively. Enter localhost in the

Host name field. In the Database for user account section, uncheck all items. Click the Go button to save your account.

After creating an account, you need to define privileges at the level of individual tables. To do this, you need to execute a special SQL command. In the case of a supply employee account, the SQL commands would look like this:

```
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'
```

To check, you need to go to the User accounts tab of the database, select Edit privileges for this account, select the Database view (Figure 1.14):

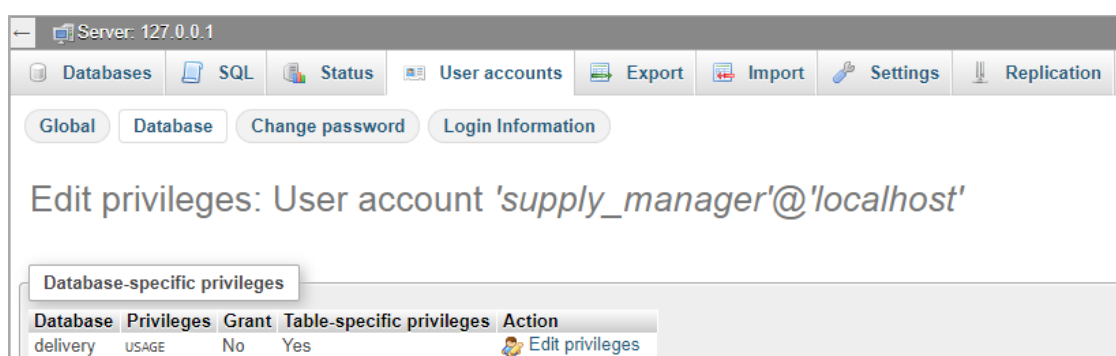


Figure 1.14

For a detailed view of the privileges for each database table, select Edit privileges in the Action column, switch to the Table view and check the correctness of the privileges assigned to the account (Figure 1.15):

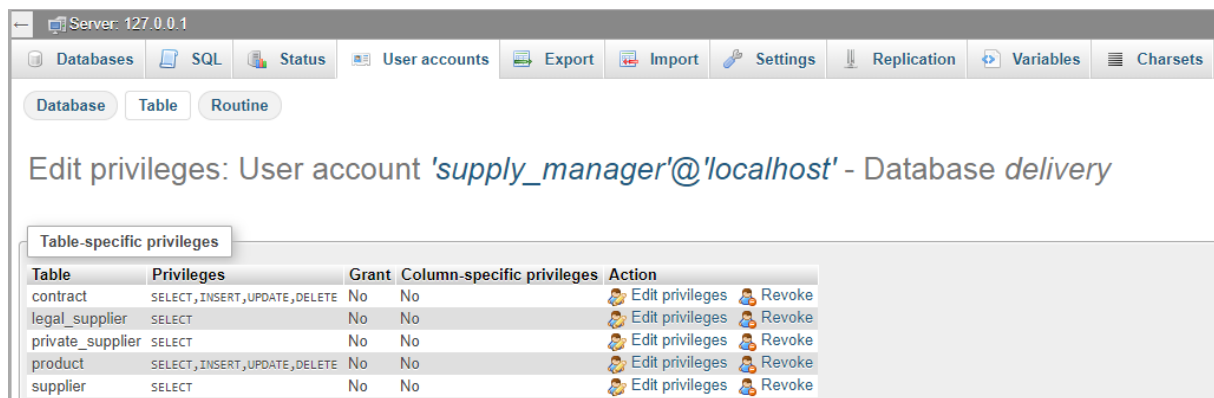


Figure 1.15

Additional information about managing user accounts in the MySQL DBMS can be obtained from the link:

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

Accounts must be created for each database user defined during system design. Privileges are assigned that must match the user's assigned stories and precedents.

### 1.2.4 Creating triggers for tables

Creating a trigger in the MySQL DBMS is carried out using the CREATE TRIGGER command, which has the following syntax:

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

Let's consider the syntax of this command in more detail:

1) the trigger name must match the naming convention [trigger time] \_ [table name] \_ [trigger event], for example before\_product\_update;

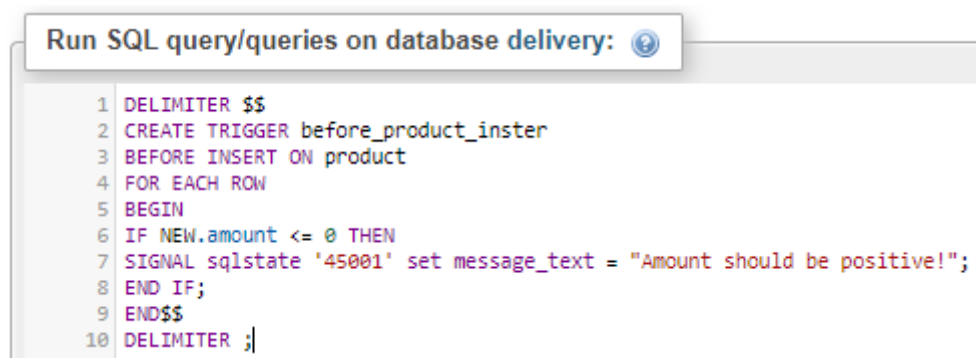
2) the trigger activation time can be BEFORE (before the change) or AFTER (after the change);

3) trigger events can be INSERT, UPDATE, or DELETE, and a separate trigger must be created to process each event;

4) the trigger is associated with a certain table;

5) SQL commands must be placed in the block between BEGIN and END, where the trigger logic is defined.

As an example, the creation of a trigger designed to control entered records about delivered goods, with the name before\_product\_insert, is considered. For this, it is necessary to execute the corresponding SQL command (Figure 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 ;
```

Figure 1.16

The DELIMITER command is not part of the trigger syntax. It is used to override the default delimiter to pass the trigger creation command entirely to the server, preventing MySQL from interpreting each row individually.

To check the operability of the created trigger, you can send requests that violate the condition specified when defining the trigger:

```
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);
```

As a result of the execution of the last request, the previously created trigger will fire and an appropriate error message will be issued (Figure 1.17):

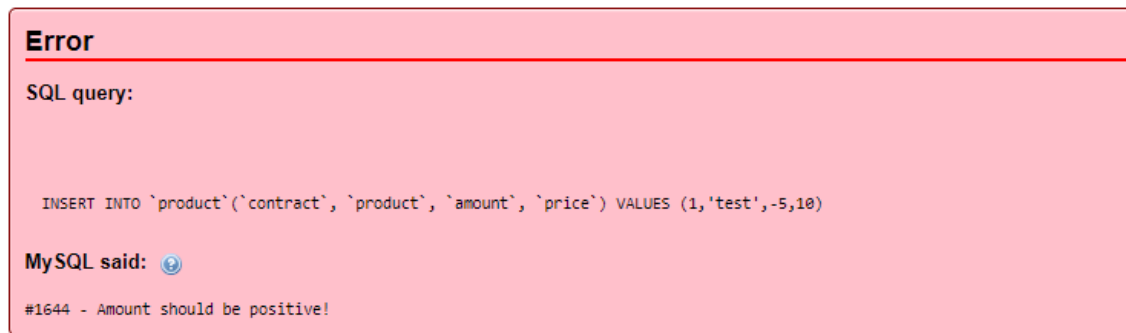


Figure 1.17

Created triggers are available for viewing in phpMyAdmin on the Triggers tab (Figure 1.18):

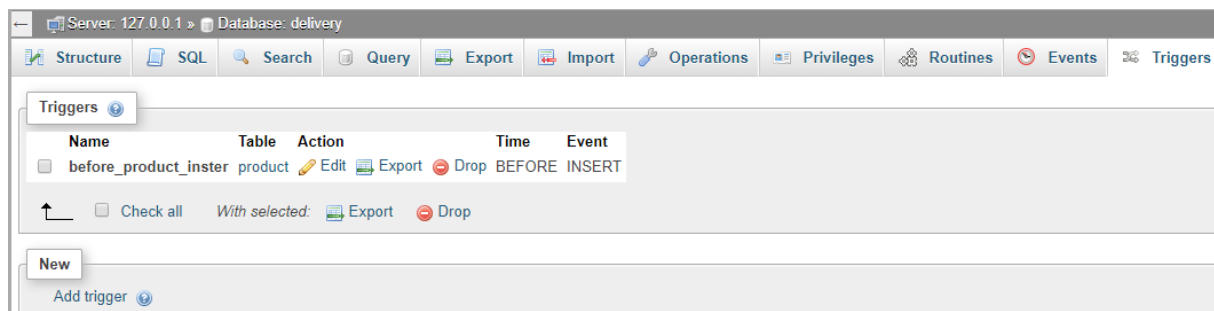


Figure 1.18

Detailed information about triggers in the MySQL DBMS can be obtained from the link:

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

Triggers must be created for each database table, if required depending on the specifics of the given subject area. The time of activation and triggering

events of the trigger can be determined independently, also based on the specifics of the specific subject area. Test the created triggers using one of the "white box" testing methods.

### 1.2.5 Creation of stored procedures and functions

Stored routines (procedures and functions) are a set of SQL commands that can be compiled and stored on the server. Thus, instead of storing a frequently used query, clients can refer to the corresponding stored procedure. This provides better performance because a given request has to be analyzed only once and the traffic between the server and the client is reduced.

Stored routines can be useful if multiple client applications are written in different languages or run on different platforms, but need to use the same database (Figure 1.19):

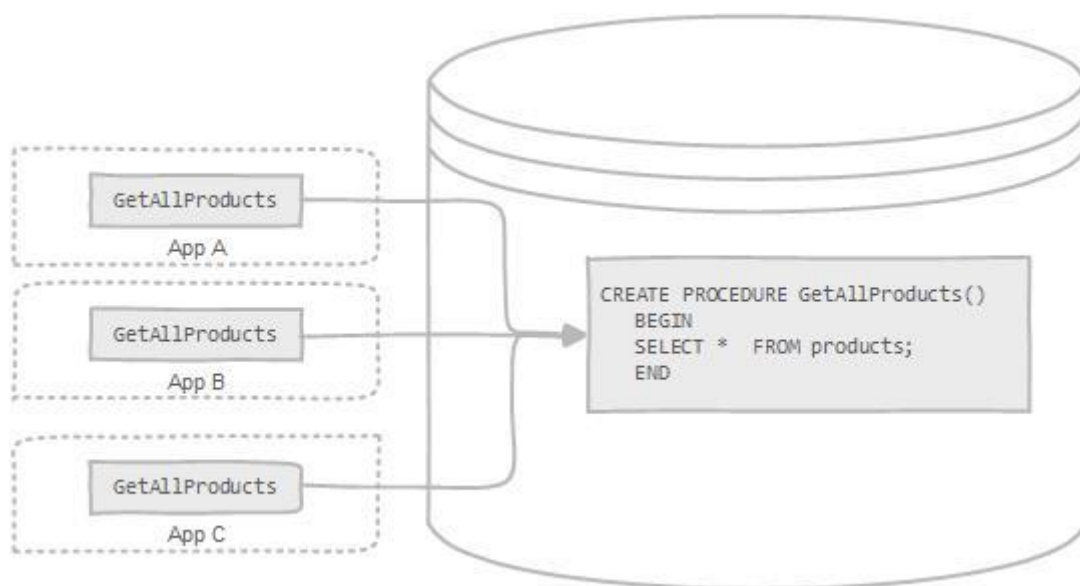


Figure 1.19

A stored routine is a procedure or function. Stored routines are created using `CREATE PROCEDURE` or `CREATE FUNCTION` statements. A stored routine is called using a `CALL` statement, and only the return values of variables



are used as outputs. The function can be called like any other function and can return a scalar value. Stored routines can call other stored routines.

**CREATE PROCEDURE** and **CREATE FUNCTION** commands have the following syntax:

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

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

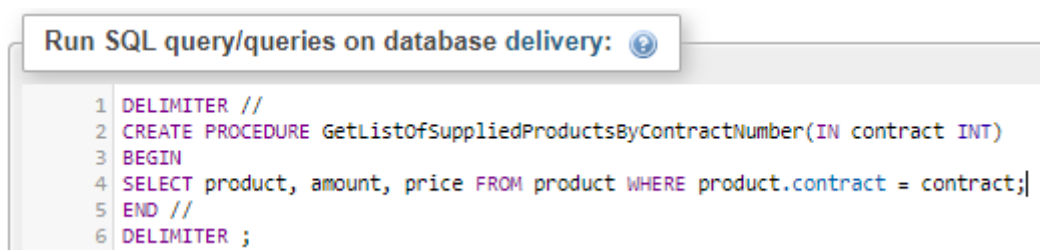
procedure_param:
    [ IN | OUT | INOUT ] param_name type
function_param:
    param_name type

type:
    any data type of MySQL

characteristics:
    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.
```

As an example, the creation of a procedure designed to obtain a list of supplied products by contract number, with the name `GetListOfSuppliedProductsByContractNumber`, is considered. For this, it is necessary to execute the corresponding SQL command (Figure 1.20):



```
Run SQL query/queries on database delivery:

1 DELIMITER //
2 CREATE PROCEDURE GetListOfSuppliedProductsByContractNumber(IN contract INT)
3 BEGIN
4 SELECT product, amount, price FROM product WHERE product.contract = contract;|
5 END //
6 DELIMITER ;
```

Figure 1.20

Stored procedures and functions created for the database are available for viewing on the Routines tab (Figure 1.21):

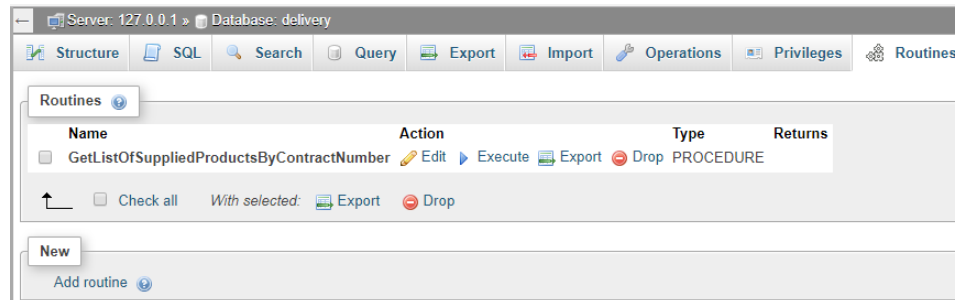


Figure 1.21

To check the operability of the created procedure, it is necessary to call it using the Execute button in the Action column and enter the required parameter value (Figure 1.22):

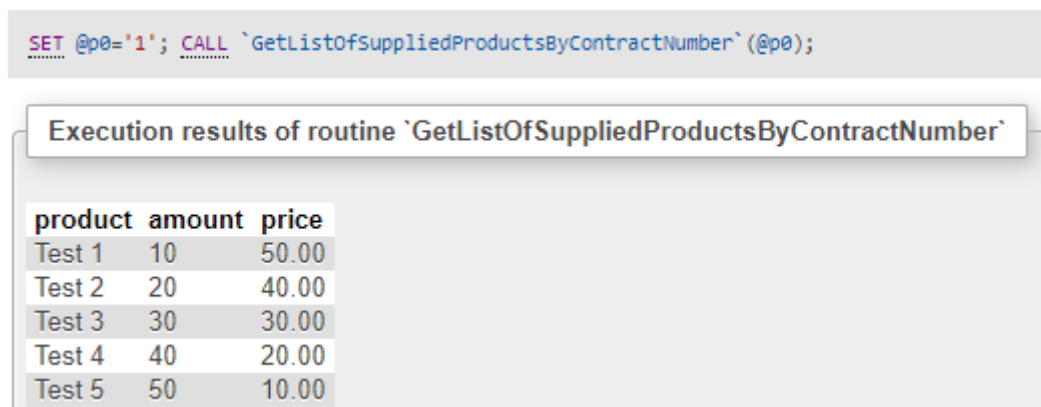
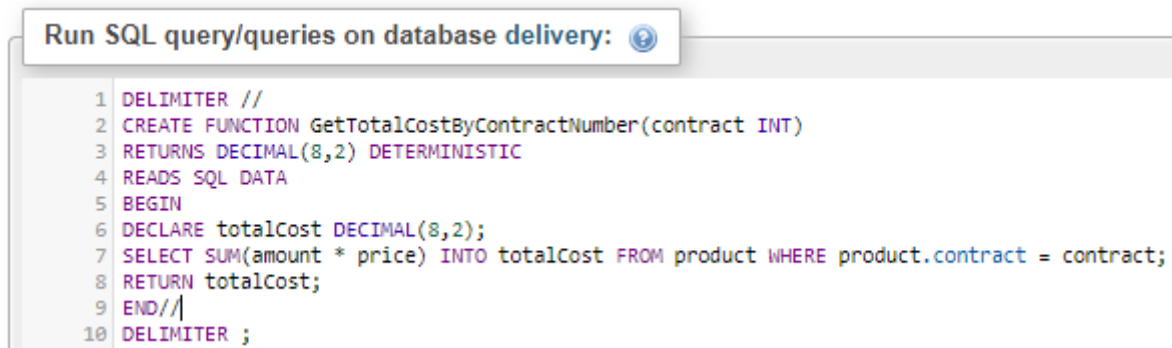


Figure 1.22

As a result of the execution of the procedure, the SQL command with which this procedure was called is also displayed.

As an example, the creation of a function designed to calculate the total cost of delivered goods under the contract, with the name

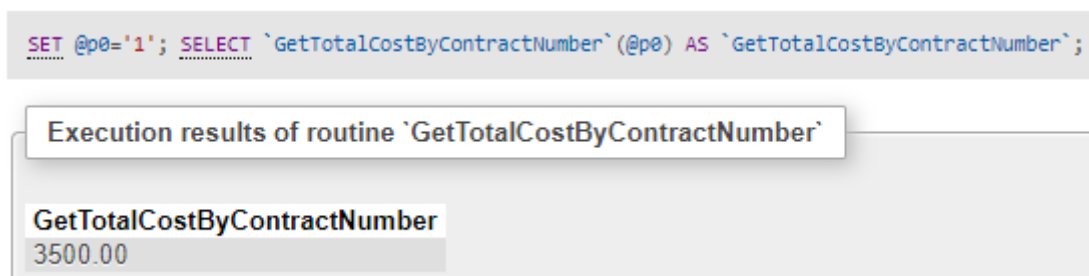
GetTotalCostByContractNumber, is also considered. For this, it is necessary to execute the corresponding SQL command (Figure 1.23):



```
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 ;
```

Figure 1.23

As a result of the execution of the created function, the result is displayed similar to the result of the procedure call (Figure 1.24):



```
SET @p0='1'; SELECT `GetTotalCostByContractNumber`(@p0) AS `GetTotalCostByContractNumber`;
```

Execution results of routine `GetTotalCostByContractNumber`

GetTotalCostByContractNumber
3500.00

Figure 1.24

Detailed information about stored procedures and functions in the MySQL DBMS can be obtained from the link:

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

Procedures and functions must be created for the developed (according to the given subject area) database. The created procedures and functions must fully satisfy the functional requirements defined at the system design stage. Test the created procedures and functions using one of the "white box" methods.

## 1.3 Working with a database in PHP

### 1.3.1 Functions for working with the database

Before getting data from the MySQL database, you need to establish a connection with the server. To establish a connection, the `mysqli_connect` function is used, which accepts the following arguments (Figure 1.25):

- 1) `servername` – the name of the MySQL server;
- 2) `username` – user name (specified when creating an account);
- 3) `password` – user password (also set when creating an account);
- 4) `dbname` – the name of the database to which the connection is made.

```
$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());
}
```

Figure 1.25

To execute SQL queries to the database, the `mysqli_query` function is used, accepting two arguments (Figure 1.26):

- 1) `conn` – the result of executing the `mysqli_connect` function;
- 2) `sql` is a string variable that contains the text of the SQL command.

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

Figure 1.26

After the query has been executed, the `mysqli_query` function will return the result of its execution. In this case, the request was for data sampling, so it is

necessary to process the received result to display the necessary information on the screen.

Using the `mysqli_num_rows` function, you can get the number of records obtained as a result of executing a given `SELECT` command. This function accepts a single argument - the result of the `mysqli_query` function. To obtain information on each record, the `mysqli_fetch_assoc` function is used, which returns an associative array containing pairs of "column name - cell content" values. This function also accepts the result of executing `mysqli_query` (Figure 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";  
}
```

Figure 1.27

Suppose that instead of a fetch request, you need to perform an update request or, for example, add data. At the same time, as a result of entering incorrect data, one of the developed triggers may work and issue a corresponding error message. To receive such a message, you can use the `mysqli_error` function. It accepts a single argument - the result of the `mysqli_connect` function (Figure 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);  
}
```

Figure 1.28

After all the necessary operations with the database have been completed, the established connection must be closed using the `mysqli_close` function. This function accepts the result of the `mysqli_connect` function.

Before starting the next stages of work, it is recommended to familiarize yourself with the features of the PHP language at the following link:

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

### 1.3.2 Repository design pattern

The Repository pattern is an intermediary between the domain layer and the data distribution layer, working like a normal collection of domain objects. Client objects create a request description declaratively and send them to the repository object (Repository) for processing. Objects can be added to or removed from the repository as if they formed a simple collection of objects. And the data distribution code, hidden in the Repository object, will take care of the relevant operations inconspicuously for the developer (Figure 1.29):

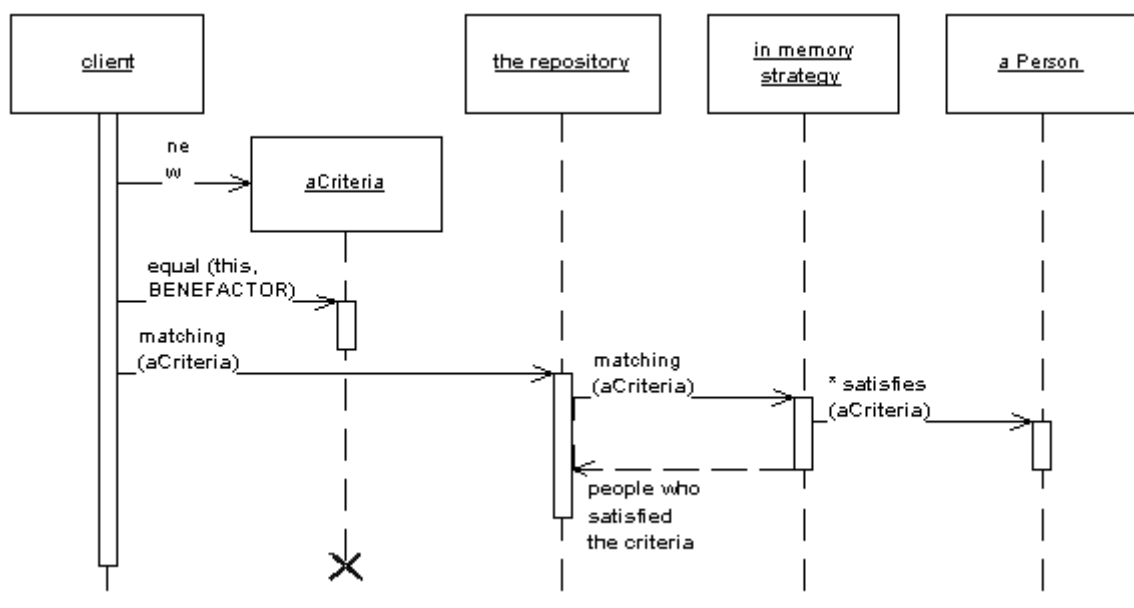


Figure 1.29

The Repository pattern encapsulates the objects represented in the data repository and the operations performed on them, providing an object-oriented representation of real data. The Repository also aims to achieve complete separation and one-way dependence between the levels of data definition and distribution.

As an example, the implementation of the Repository pattern for working with objects describing information about contracts is considered:

### 1. Contract class:

```
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. The ContractRepositoryInterface interface, which defines the behavior of the object that works with contract data:**

```

interface ContractRepositoryInterface
{
public function getContractList();

public function getContractByNumber($number);

public function create($contract);

public function update($contract);

public function delete($number);
}

```

**3. Implementation of the ContractRepositoryInterface interface, intended for working with the MySQL database - MySQLContractRepository class:**

```

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. The MySQLConnectionUtil class is used to establish a connection to the MySQL database:

```
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;  
}  
}
```

The created classes and interfaces must be located in the xampp/htdocs directory. A sample view of the structure of directories and files is presented in Figure 1.30:

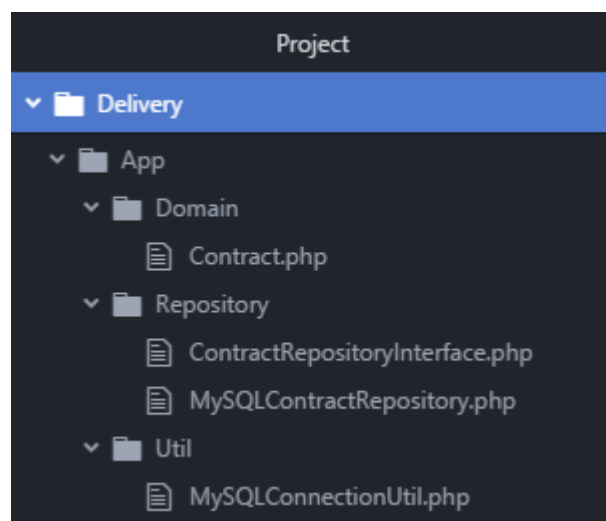


Figure 1.30

For a given subject area, it is necessary to implement all necessary classes-collections of objects using the Repository pattern. When developing

classes for working with the MySQL DBMS, it is advisable to provide protection against SQL injections by first reading the material on the link:

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

### 1.3.3 Service Layer design pattern

The Service Layer pattern defines for the application a limit and a set of permissible operations from the point of view of the client interacting with it (Figure 1.31). It encapsulates the business logic of the application, managing transactions and managing responses in the implementation of these operations.

Business applications typically require different interfaces to the data they store and the logic they implement:

- 1) data loaders;
- 2) user interfaces;
- 3) integration gateways, etc.

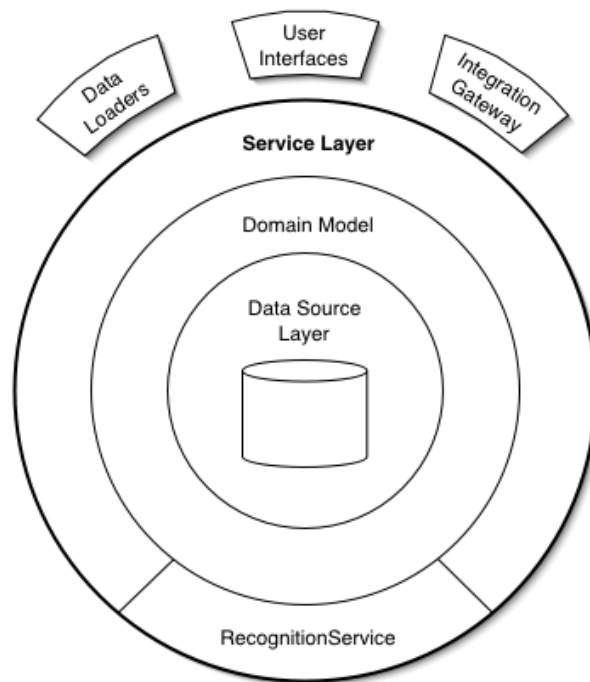


Figure 1.31

These interfaces often require interaction with the application to access and manage its data and execute logic. These interactions can be complex, using

transactions across multiple resources and managing multiple responses to an action. Programming interaction logic for each interface will cause more duplication.

As an example, the creation of the `ContractService` class, which encapsulates the business logic of working with contracts, is considered:

```
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('Ymd');

            $contract = new Contract($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!');
    }
}
}

```

According to the file and directory structure of the project, the ContractService.php file must be located in Delivery/App/Service.

For a given subject area, it is necessary to implement all the necessary classes encapsulating the business logic of the system being created based on the considered Service Layer pattern. Test the created classes using one of the "white box" methods.

***Check the execution results*** methods in the created classes can be used, for example, as follows (checking receipt of information about the contract by its number):

```

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();

```

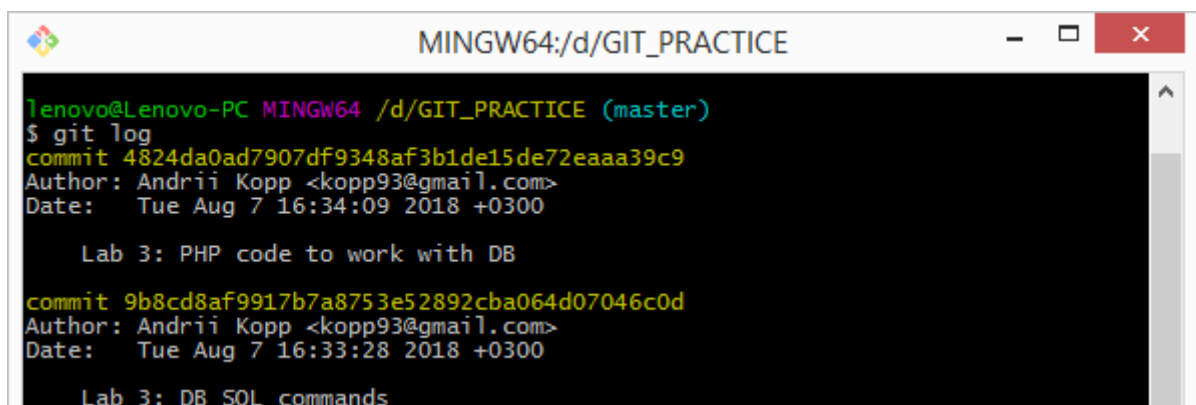
The test file TestContractService.php must be placed in the Delivery/Test directory. The advantage when performing this task will be the use of one of the frameworks for testing, for example PHPUnit:

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

In the Git repository, it is necessary to fix (Figure 1.32):

1) SQL commands used when creating database tables, triggers, stored procedures and functions, as well as a set of test data (place in a subdirectory created in the working directory, for example, D:\GIT\_PRACTICE\db);

2) the source code of the developed classes (place in the previously created web directory, for example D:\GIT\_PRACTICE\web\Delivery).



```

MINGW64:/d/GIT_PRACTICE

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

```

Figure 1.32

After finishing work, merge the storage branch into master.

#### **1.4 Requirements for the report**

- 1) briefly describe the main stages of work performance;
- 2) provide the structure of the created database and relationships between tables, the text of SQL commands, as well as test cases for triggers, stored procedures and functions;
- 3) show the structure of the created interfaces and classes using a class diagram in the UML modeling language, their source code, as well as test cases for class methods encapsulating the business logic of the application;
- 4) demonstrate the obtained results in the form of the results of executing SQL commands and testing using "white box" methods.

#### **1.5 Questions for self-testing**

1. What problems can occur when starting the XAMPP server? How can these problems be solved?
2. What is the phpMyAdmin application used for?
3. How to create a new database in the MySQL DBMS?
4. What encoding is recommended to use? What is the advantage of the selected encoding?
5. What are the ways to create database tables? Name the main data types of the MySQL DBMS.
6. How can you view the structure of the created database in phpMyAdmin?
7. What are user accounts used for? How to create a user account and set the necessary privileges?

8. Describe the syntax of the command designed to create a trigger in the MySQL DBMS. What is the DELIMITER command used for?

9. Describe the syntax of the command designed to create a stored procedure in the MySQL DBMS.

10. Describe the syntax of the command designed to create a function in the MySQL DBMS.

11. How can you establish a connection to a MySQL database in PHP?

12. How can you execute a request to the MySQL database in PHP?

13. How can you get the result of a query to the MySQL database in PHP?

14. How can you get information about errors that occur when working with the MySQL database in PHP?

15. How can you close a connection to a MySQL database in PHP?

16. What is the Repository design pattern for?

17. What is the Service Layer design pattern for?



## **2 CREATION OF WEB APPLICATIONS USING THE BOOTSTRAP FRAMEWORK. USING AJAX TECHNOLOGY FOR ASYNCHRONOUS DATA EXCHANGE WITH A WEB SERVER**

### **2.1 Preparation for work**

1. Create a new branch in the Git version control system and call it pages.
2. Download the Bootstrap framework, designed for creating sites and web applications, using the link:

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

To quickly start working with Bootstrap, you can also use the suggested method:

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

3. While on the created branch, in the Delivery directory, create a Web subdirectory, placing the contents of the downloaded archive with the Bootstrap framework into it and, subsequently, the created application pages.

### **2.2 Getting to know the Bootstrap framework**

The Bootstrap framework is based on modern developments in HTML and CSS. Includes design templates for typography, web forms, buttons, labels, navigation blocks and other web interface components, including JavaScript extensions.

Basic Bootstrap Tools:

1. Grids are predefined column sizes that can be used immediately (Figure 2.1). Detailed information on the use of grids is available at the link:

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

2. Templates – a fixed or dimensionless document template (Figure 2.2). Detailed information on the use of templates is available at the link:

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

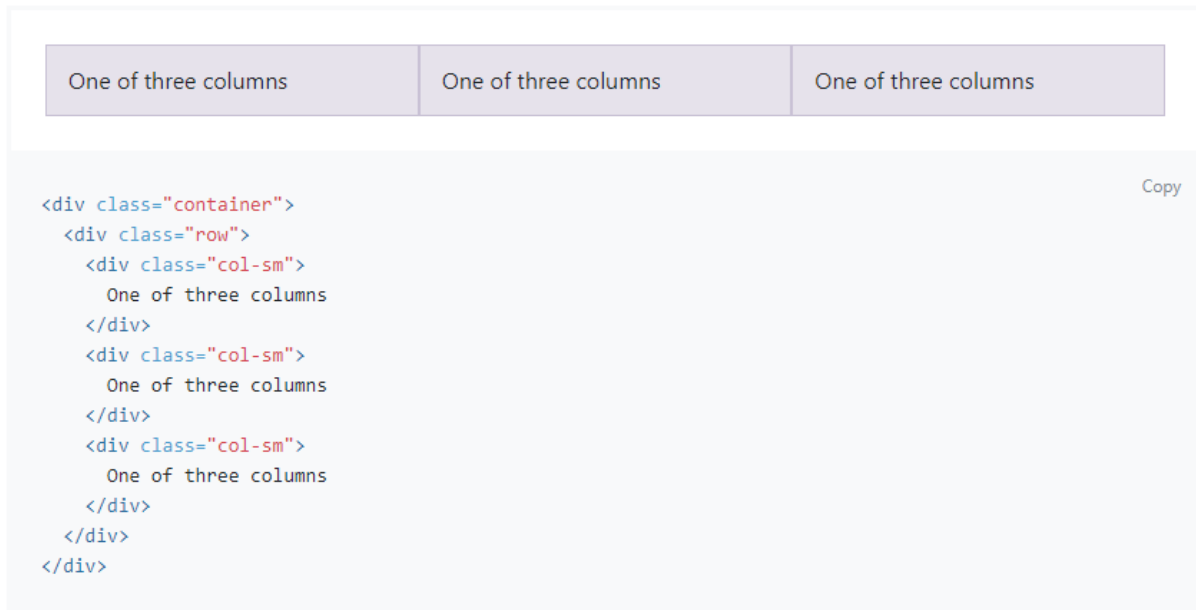


Figure 2.1

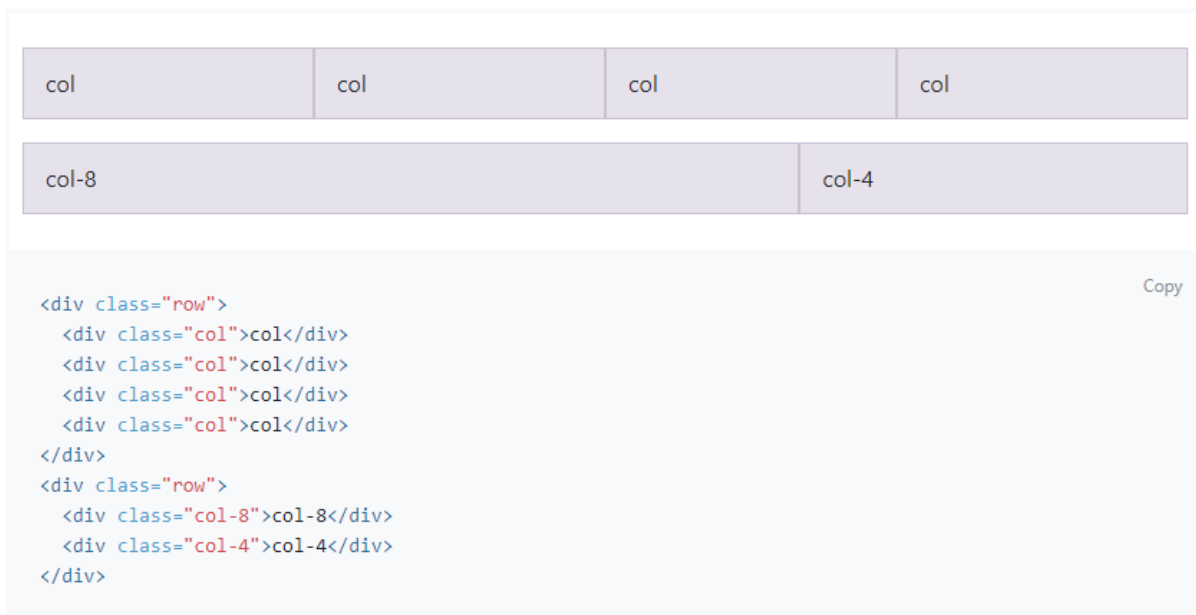


Figure 2.2

3. Typography – description of fonts, definition of some classes for fonts, such as code, quotes, etc. (Figure 2.3). Detailed information on the use of typography is available at the link:

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

**h1. Bootstrap heading**

**h2. Bootstrap heading**

**h3. Bootstrap heading**

**h4. Bootstrap heading**

**h5. Bootstrap heading**

**h6. Bootstrap heading**

```
<p class="h1">h1. Bootstrap heading</p>
<p class="h2">h2. Bootstrap heading</p>
<p class="h3">h3. Bootstrap heading</p>
<p class="h4">h4. Bootstrap heading</p>
<p class="h5">h5. Bootstrap heading</p>
<p class="h6">h6. Bootstrap heading</p>
```

Copy

Figure 2.3

4. Media – represents some way of organizing images and videos (Figure 2.4). Detailed information on the use of media is available at the following link:

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

5. Tables – means of designing tables, up to the addition of sorting functionality (Figure 2.5). Detailed information on the use of tables is available at the following link:

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

64x64

### Media heading

Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.

```
<div class="media">
  
  <div class="media-body">
    <h5 class="mt-0">Media heading</h5>
    Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin. Cras pi
  </div>
</div>
```

Copy

Figure 2.4

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
```

Copy

Figure 2.5

6. Forms – classes for designing forms and some events that occur with them (Figure 2.6). Detailed information on the use of forms is available at the following link:

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

7. Navigation – design classes for tabs, pages, menus and toolbars (Figure 2.7). Detailed information on the use of navigation is available at the link:

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

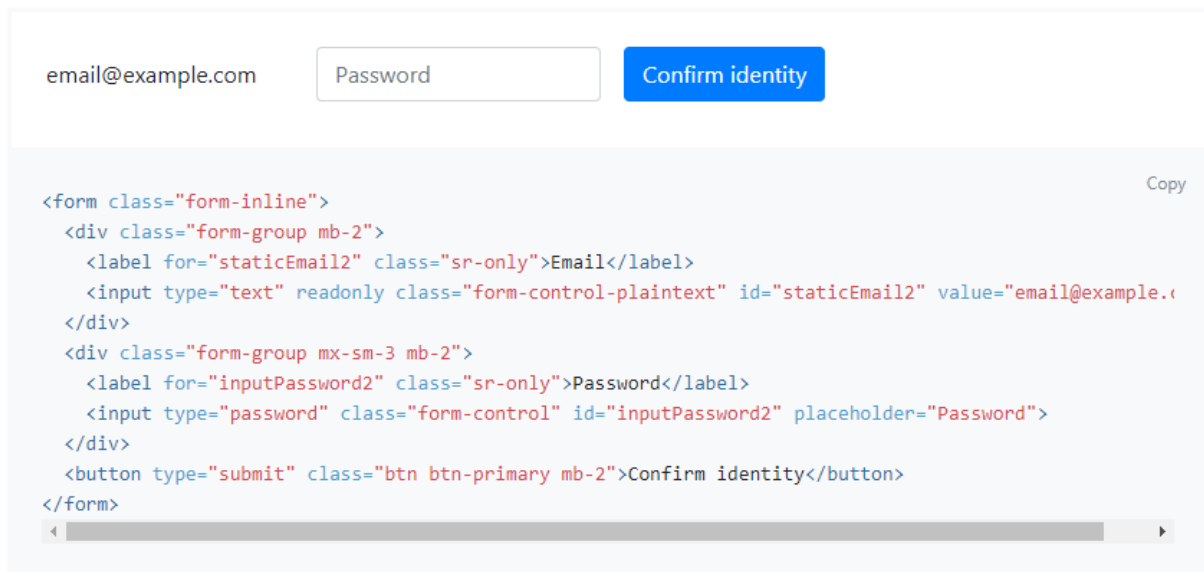


Figure 2.6



Figure 2.7

8. Notifications – design of dialog boxes, prompts and pop-up windows (Figure 2.8). Detailed information on the use of notifications is available at the link:

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



Figure 2.8

## 2.3 Creation of web pages

### 2.3.1 Creating an application login page

An example of the simplest application login page, created using the Bootstrap framework, is presented in Figure 2.9:

The figure shows a simple application login page. It consists of two input fields, one for "User name" and one for "Password". The "User name" field is a text input, and the "Password" field is a password input. Below the input fields is a blue "Sign in" button. The labels "User name" and "Password" are positioned to the left of their respective input fields. The entire form is styled with a light gray background and a white border.

Figure 2.9

A horizontal form was used to create this page, by using the `.row` class for form groups, as well as the `.col-*-*` classes for setting the width of the form elements and their captions. A prerequisite is to add the `.col-form-label` class to the `<label>` tags so that they are vertically centered relative to their associated form elements.

An example of the code used to create an application login form (Figure 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 Working with single sessions

When working with an application, the user starts it, performs some actions, and then closes it. This is called a user-defined session or session.

However, the web server "knows" nothing about the user, since the HTTP protocol does not store state. This problem is solved by session variables by saving information about the user, making it available for use on several pages (for example, the user's name, etc.). By default, session variables are saved until the user closes the browser.

The `session_start()` function is used to start a session. Session variables are stored in the `$_SESSION` global array. The example in Figure 2.10 demonstrates starting a session and setting some session variables:

```

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

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

</body>
</html>

```

Figure 2.10

It should be noted that session variables are not transferred separately to each page. Instead, they are retrieved from the session (using the `$_SESSION` global array) that is opened at the beginning of the page using the `session_start()` function. The example in Figure 2.11 demonstrates the beginning of a session and access to session variables:

```

<?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>

```

Figure 2.11



To remove all session variables and destroy the session, the `session_unset()` and `session_destroy()` functions are used. The example in Figure 2.12 demonstrates session destruction:

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

// destroy the session
session_destroy();
?>
```

Figure 2.12

Taking into account the need to use user sessions in the work of the created application, the source code of the application login page (`login.php`) will look like this:

```
<?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>
<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>

```

For the correct operation of this page, it is necessary to create the Controller class by placing the corresponding file in the App/Controller.php directory. The source code of this class looks like this:

```

<?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();
?>

```

Accordingly, to exit the program, it is necessary to create a page (logout.php) that contains calls to the functions of destroying the session assigned to the user:

```

<?php
session_start();

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

```

```

session_unset();
session_destroy();

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

```

The code of the start page of the application (index.php) will contain a check for the presence of a session, and direct the user to the application login page or to the main page, respectively:

```

<?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 Creating a page for working with data

As an example of a work page with data, consider the work page of an employee of the supply department with information about contracts (Figure 2.13):

Delivery	Contracts	Logout
Contracts	New contract	
#1, 1999-09-01, Ivanov I. I. PE		
#2, 1999-09-10, Ivanov I. I. PE		
#3, 1999-09-10, Petrov P. P. PE		
#4, 1999-09-23, Petrov P. P. PE		
#5, 1999-09-24, "Interfrut" LLC		
#6, 1999-10-01, Ivanov I. I. PE		
#7, 1999-10-02, "Interfrut" LLC		

Figure 2.13

The Bootstrap component was used to create the page - a navigation panel that includes sub-components such as links to pages and a form for exiting the program.

An example of the code used to create a navigation panel (Figure 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>
```

Another Bootstrap component, the list, is used to display the list of contracts. An example of the code used to create a list of contracts (Figure 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>
```

Appropriate Bootstrap components - buttons and forms - are used to perform actions such as creating a new contract or editing/deleting a contract.

An example of the code used to create a button to go to the creation of a new contract (Figure 2.13):

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

When clicking on the link, which is each item in the list of contracts, detailed information about the contract is loaded, presented in the form of a form with the corresponding buttons for editing / deleting the contract (Figure 2.14):

Delivery
Contracts

Logout

Contracts

#1, 1999-09-01, Ivanov I. I. PE

#2, 1999-09-10, Ivanov I. I. PE

#3, 1999-09-10, Petrov P. P. PE

#4, 1999-09-23, Petrov P. P. PE

#5, 1999-09-24, "Interfrut" LLC

#6, 1999-10-01, Ivanov I. I. PE

#7, 1999-10-02, "Interfrut" LLC

Contract number5

Contract date1999-09-24

Supplier"Interfrut" LLC

TitleContract 5

Note

Invoice 74  
from 9/11/99

Edit

Remove

Figure 2.14

An example of the code used to create a form for viewing contract information (Figure 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="<?=$contract->getSupplier() ?>">
</div>
</div>
<div class="form-group row">
<label for="title" class="col-sm-2 col-form-label">Title</label>
<div class="col-sm-10">
<input type="text" readonly class="form-control-plaintext" id="title" value="<?=$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"><?=$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>

```

For the location of the list of contracts and the form for viewing detailed information on each contract (Figure 2.14), the corresponding `.col-*` classes of the Bootstrap framework are used, designed to create an adaptive grid.

The final version of the source code of the contract page (`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']);
    }
    catch (Exception $e)
    {
        $controller->redirect('error');
    }
}
?>
<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="<?=$contract->getSupplier() ?>">
</div>
</div>
<div class="form-group row">
<label for="title" class="col-sm-2 col-form-label">Title</label>
<div class="col-sm-10">
<input type="text" readonly class="form-control-plaintext" id="title" value="<?=$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"><?=$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>

```

Errors that occur when the user works with Zastsoun are demonstrated using the corresponding Bootstrap components - alerts (Figure 2.15):

Access denied for user '@'localhost' to database 'delivery'

User name

User name

Password

Password

Sign in

Figure 2.15

When performing the work, it is necessary to create (using Bootstrap) the appropriate pages, according to the given subject area. It is recommended to first create static prototypes of the pages and coordinate them with the teacher. Then, based on the created prototypes, you can create dynamic PHP pages.

## **2.4 Asynchronous data exchange with the web server**

### **2.4.1 Introduction to AJAX technology**

AJAX (Asynchronous Javascript and XML) is an approach to building interactive user interfaces of web applications, which consists in the "background" exchange of data between the browser and the web server. As a result, when updating data, the web page is not completely reloaded, and web applications become faster and more convenient.

In the classic web application model (Figure 2.16):

- 1) the user enters the web page and clicks on any of its elements;
- 2) the browser forms and sends a request to the server;
- 3) in response, the server generates a completely new web page and sends it to the browser, etc., after which the browser completely reloads the entire page.

When using AJAX (Figure 2.16):

- 1) the user enters the web page and clicks on any of its elements;
- 2) a script (in JavaScript) determines what information is needed to update the page;
- 3) the browser sends a corresponding request to the server;
- 4) the server returns only that part of the document for which the request came.
- 5) the script makes changes taking into account the received information (without completely reloading the page).



***advantages*** of this technology are:

- 1) saving traffic;
- 2) reducing the load on the server;
- 3) acceleration of the interface response;
- 4) virtually unlimited possibilities for interactive processing.

However, AJAX technology also has disadvantages:

- 1) lack of integration with standard browser tools;
- 2) dynamically loaded content is unavailable to search engines;
- 3) old methods of accounting for site statistics become irrelevant;
- 4) complication of the project;
- 5) JavaScript must be enabled in the browser;
- 6) problems with displaying non-standard encodings;
- 7) low speed with rough programming;
- 8) bad behavior with an unreliable connection;
- 9) the risk of requests being fabricated by other sites.

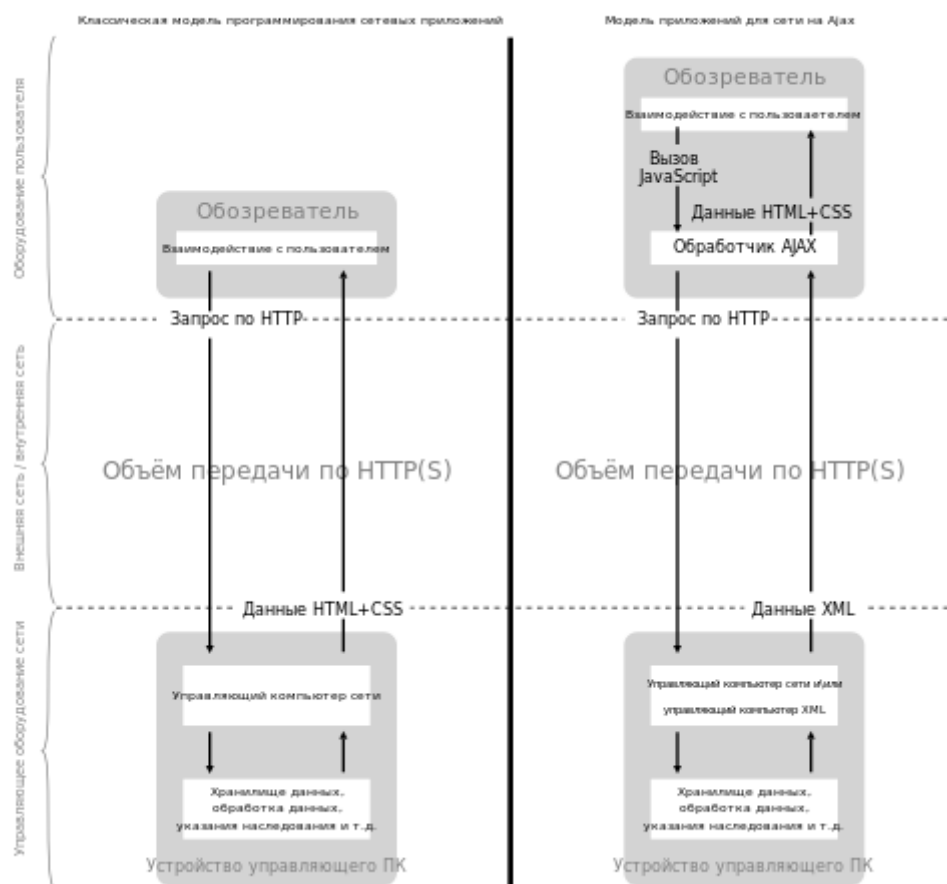


Figure 2.16

## 2.4.2 Using AJAX on the application login page

To use asynchronous data exchange with the server when entering the application, you need to add new elements to the code of the login.php page:

```
<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>
```

These blocks will be used to display information received from the server (error message or successful authorization).

In the Web directory, you need to create an AJAX subdirectory in which the ajax\_login.php file will be placed, designed for processing asynchronous requests to the server:

```
<?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()));
}

?>
```

To simplify the implementation of requests to the server, the jQuery library will be used:

```
$("#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;
});
```

This code must be placed in the login.js file at Web/js. After the JavaScript file is created, it must be connected in the file of the application's

login page - login.php. An example of the final version of the source code of the application login page:

```
<?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>
<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>
```

The use of AJAX technology is a mandatory requirement when performing the work, while the use of the jQuery library is optional (it is allowed to use "pure" JavaScript or other libraries/frameworks). When implementing asynchronous data exchange with the server, it is necessary to take into account the possible disadvantages of this approach.

**to fix** the results of work in the Git version control system. Merge the pages branch into master after finishing work. If necessary, resolve conflicts that have arisen.

**Create a remote repository** Git using one of the free hosts (GitHub, GitLab, etc.). Publish the results of the work to the remote repository using the git push command. You can familiarize yourself with the features of working with remote repositories by following the link:

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

## **2.5 Report requirements**

- 1) briefly describe the main stages of work performance;
- 2) show the appearance of the created pages;
- 3) provide the source code of the created pages, PHP and Java Script scripts;
- 4) demonstrate the obtained results in the form of effects in the software, provide a history of changes in the Git version control system, a link to a remote repository.

## **2.6 Questions for self-testing**

1. What is Bootstrap?
2. Basic Bootstrap tools?

3. What Bootstrap tools were used during the work?
4. How is the problem of storing information about the user working with the application solved?
5. What is a session (session)? How to create a session?
6. How can data be saved using a session?
7. How can session variables be accessed?
8. How to destroy a session and delete all session variables?
9. What is HTTP? Name the HTTP methods, their main features and differences.
10. What HTTP methods were used in the execution of the work?
11. What is AJAX?
12. The main advantages and disadvantages of AJAX technology.
13. Classic model of web applications.
14. Application model when using AJAX.
15. What is jQuery?
16. By what means was the AJAX technology implemented when performing the work?