Домашняя работа

по предмету: Основы PHP (семинары)

Выполнил: Байбородин Александр

Группа: Программист | 17.04.2024

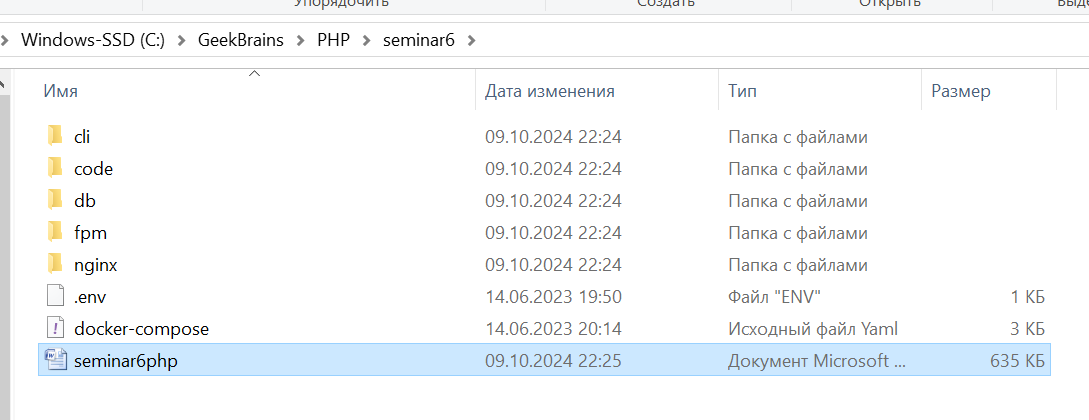
### Урок 6. Работа с БД

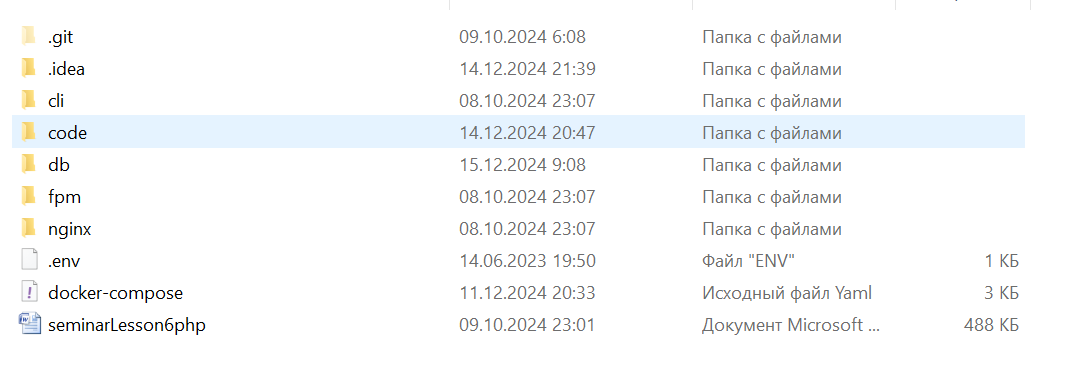
Задание

Мы стали работать с исключениями. Создайте в Render логику обработки исключений так, чтобы она встраивалась в общий шаблон. Вызов будет выглядеть примерно так:  
  
try{  
$app = new Application();  
echo $app->run();  
}  
catch(Exception $e){  
echo Render::renderExceptionPage($e);  
}  
  
— Создайте метод обновления пользователя новыми данными. Например,  
  
/user/update/?id=42&name=Петр  
  
Такой вызов обновит имя у пользователя с ID 42. Обратите внимание, что остальные поля не меняются. Также помните, что пользователя с ID 42 может и не быть в базе.  
  
— Создайте метод удаления пользователя из базы. Учитывайте, что пользователя может не быть в базе  
/user/delete/?id=42

Решение:

Сформируем структуру каталогов и файлов проекта:





Сформируем докер-файл, для сборки и запуска контейнера:

# версия синтаксиса

version: '3'

# в этом блоке мы описываем контейнеры, которые будут запускаться

services:

  #контейнер с Nginx

  nginx:

    build:

      context: ./nginx

      dockerfile: Dockerfile

    image: myapp/nginx

    container\_name: webserver

    # проброс портов

    ports:

      - "80:80"

    volumes:

       - ./code:/data/mysite.local

    networks:

      - app-network

  #Контейнер с PHP-FPM, назовём его app

  application:

    # Если нет секции build, то система будет искать образ в репозиториях

    build:

      context: ./fpm

      dockerfile: Dockerfile

    image: myapp/php # имя будущего образа

    container\_name: application # имя контейнера после запуска

    volumes:

       - ./code:/data/mysite.local

    # мы можем создать для контейнеров внутреннюю сеть

    networks:

      - app-network

   #Контейнер с PHP-FPM, назовём его app

  cli:

    # Если нет секции build, то система будет искать образ в репозиториях

    build:

      context: ./cli

      dockerfile: Dockerfile

    image: myapp/php-cli # имя будущего образа

    container\_name: cli # имя контейнера после запуска

    volumes:

       - ./code:/data/mysite.local

    # мы можем создать для контейнеров внутреннюю сеть

    networks:

      - app-network

  #Контейнер с БД

  database:

    image: mysql:5.7

    container\_name: database # имя контейнера после запуска

    environment:

      MYSQL\_DATABASE: ${DB\_NAME} # имя нашей БД

      MYSQL\_USER: ${DB\_USER} # имя пользователя, с которым будет подключаться

      MYSQL\_PASSWORD: ${DB\_PASSWORD} # пароль для пользователя

      MYSQL\_ROOT\_PASSWORD: ${DB\_ROOT\_PASSWORD} # администраторский пароль

    ports:

      - "3306:3306"

    volumes:

        - ./db:/var/lib/mysql

    # мы можем создать для контейнеров внутреннюю сеть

    networks:

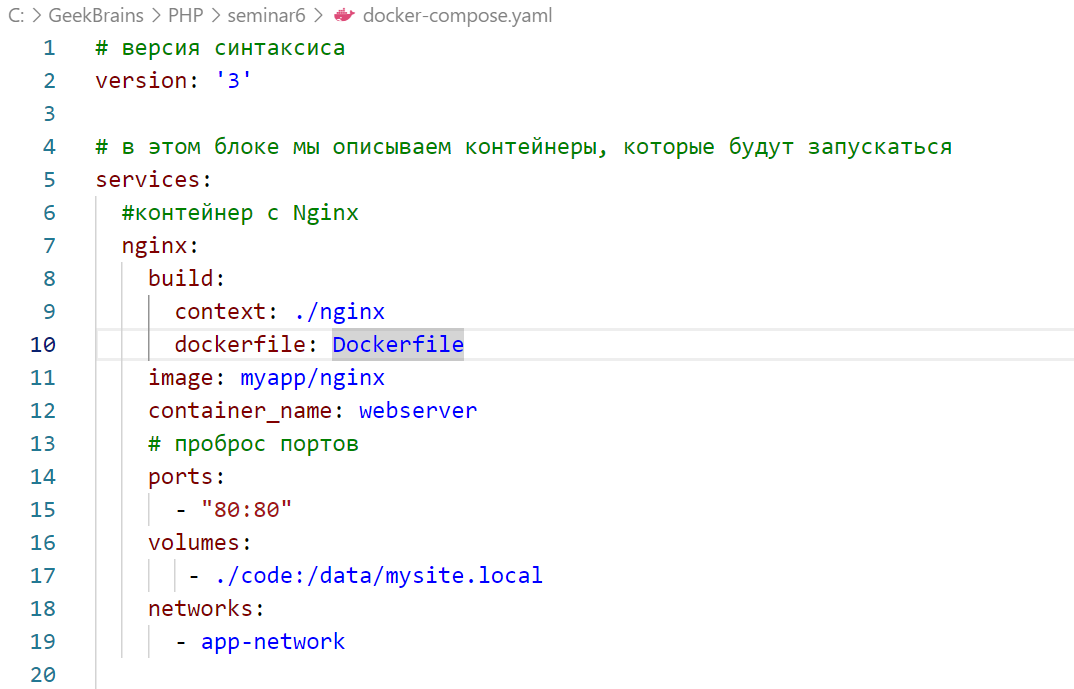
      - app-network

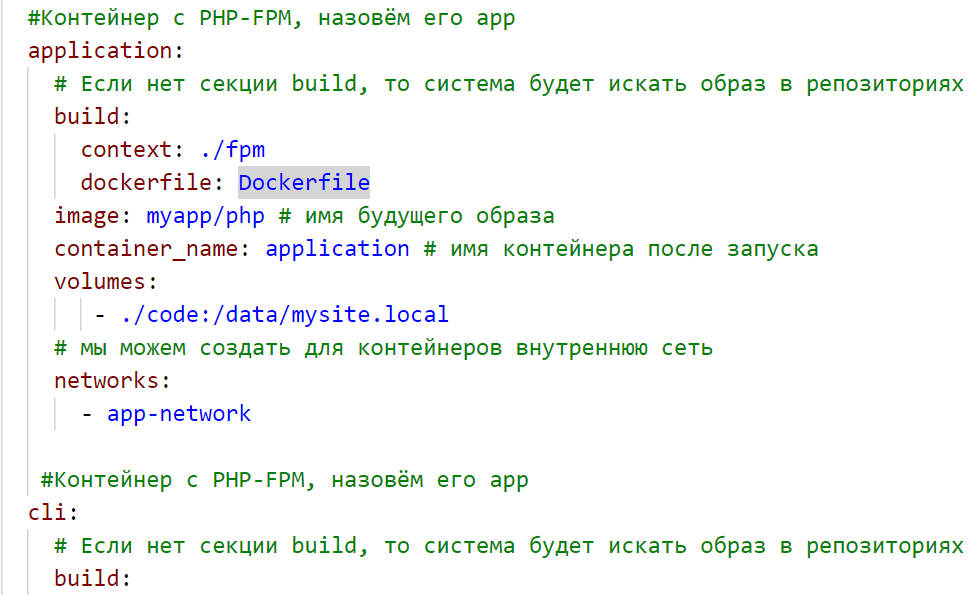
#Docker Networks

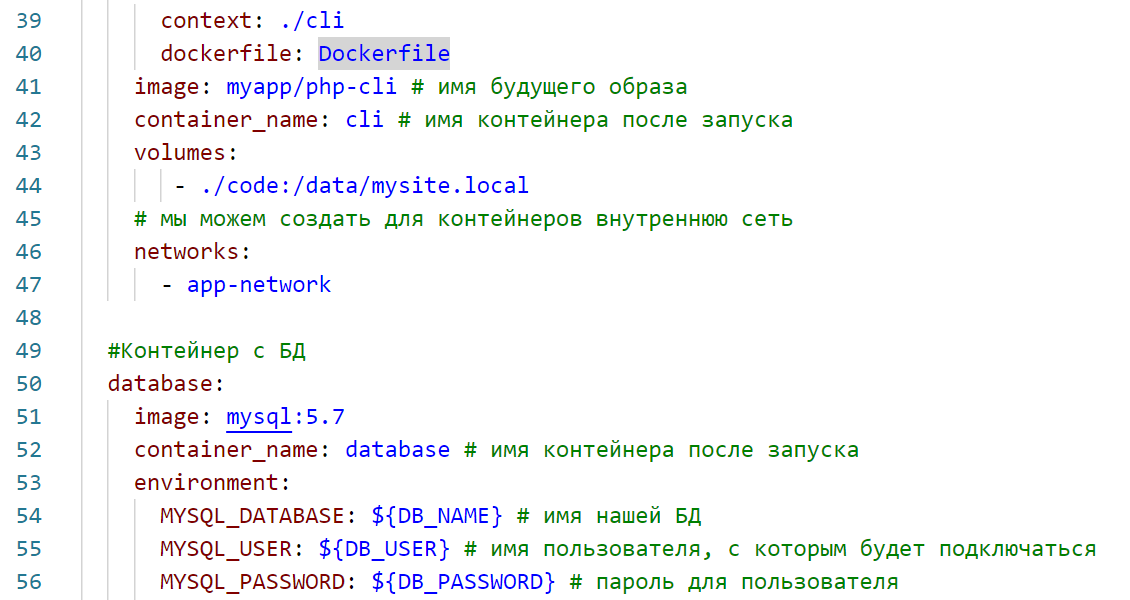
networks:

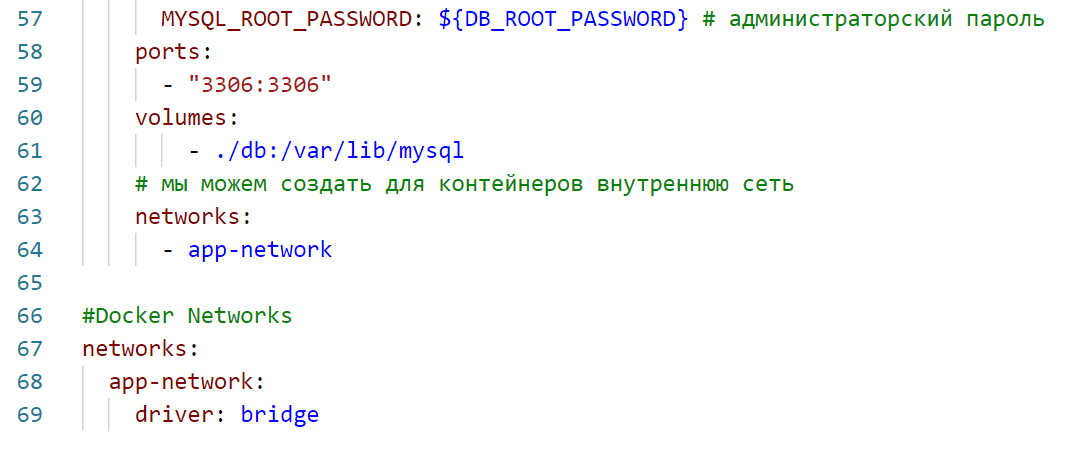
  app-network:

    driver: bridge









Создадим файл .env (для создания строк соединения c базой mysql при запуске yaml-файла через docker-compose командой docker-compose -f docker-compose.yaml up) с содержимым:

DB\_NAME=application1

DB\_USER=application\_user

DB\_PASSWORD=geekbrains!23

DB\_ROOT\_PASSWORD=GeekBrains&56

Эти строки необходимы, так как данные эти строк используются в строках файла docker-compose.yaml:

#Контейнер с БД

  database:

    image: mysql:5.7

    container\_name: database # имя контейнера после запуска

    environment:

      MYSQL\_DATABASE: ${DB\_NAME} # имя нашей БД

      MYSQL\_USER: ${DB\_USER} # имя пользователя, с которым будет подключаться

      MYSQL\_PASSWORD: ${DB\_PASSWORD} # пароль для пользователя

      MYSQL\_ROOT\_PASSWORD: ${DB\_ROOT\_PASSWORD} # администраторский пароль

    ports:

      - "3306:3306"

    volumes:

      - ./db:/var/lib/mysql

    # мы можем создать для контейнеров внутреннюю сеть

    networks:

      - app-network

Теперь sql-скриптом создадим таблицу users в базе mysql:

-- Создание базы данных (если она еще не существует)

CREATE DATABASE IF NOT EXISTS application1;

-- Использование базы данных

USE application1;

-- Создание таблицы users

CREATE TABLE IF NOT EXISTS users (

id\_user INT(11) AUTO\_INCREMENT PRIMARY KEY,

user\_name VARCHAR(45) NOT NULL,

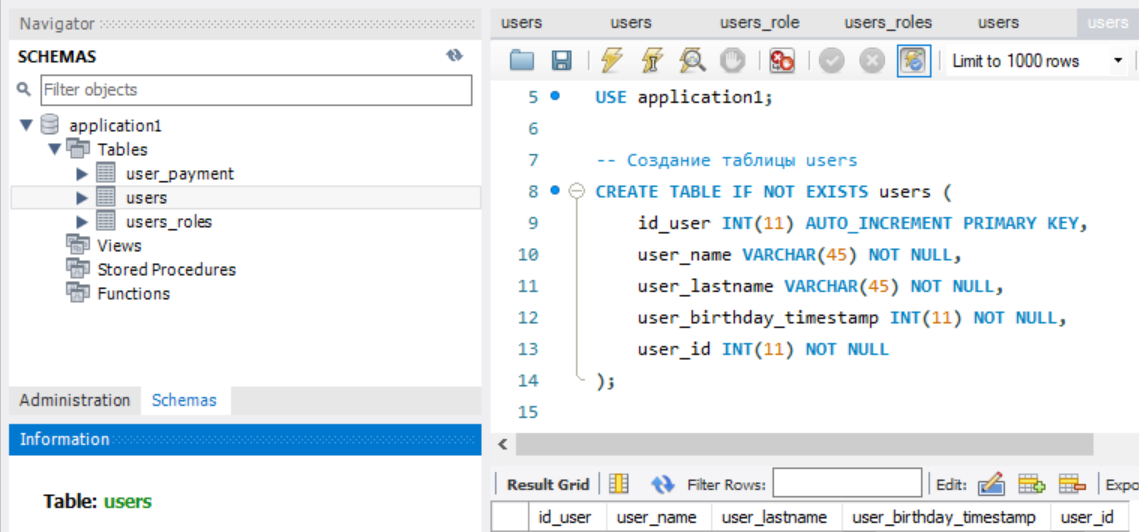
user\_lastname VARCHAR(45) NOT NULL,

user\_birthday\_timestamp INT(11) NOT NULL,

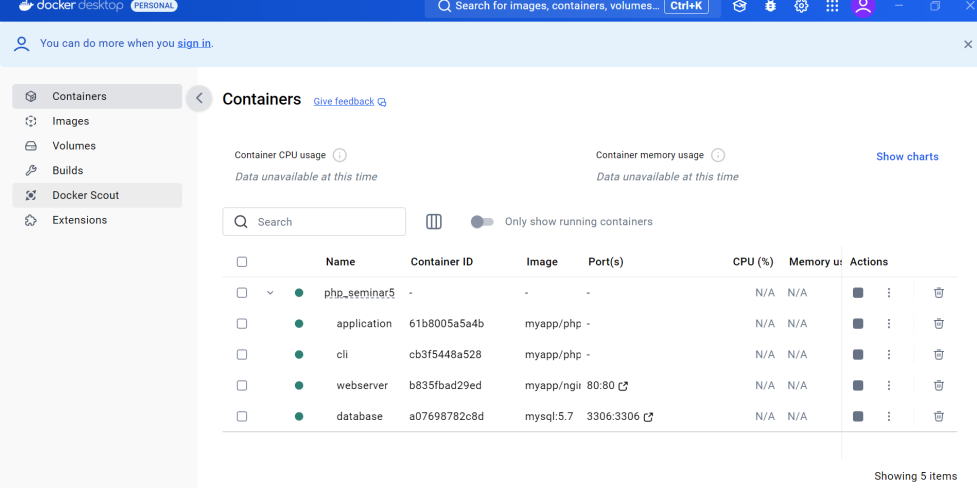
user\_id INT(11) NOT NULL

);

Видим, что таблица users создалась:



Запускаем контейнеры командой docker-compose -f docker-compose.yaml up и видим, что контейнеры запущены:



После создания скриптом таблицы users создадим код в файле index.php для тестирования работы PHP, соединения с базой данных, добавления, редактирования и удаления через GET-запросы строк:

<?php

echo "Серевер c PHP работает<br>";

$servername = "database";

$username = "application\_user";

$password = "geekbrains!23";

$dbname = "application1";

try {

    $dsn = "mysql:host=$servername;dbname=$dbname;charset=utf8mb4";

    $options = [

        PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

        PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

        PDO::MYSQL\_ATTR\_INIT\_COMMAND => "SET NAMES utf8mb4",

    ];

    $conn = new PDO($dsn, $username, $password, $options);

    echo "Соединение с БД MySQL установлено<br>";

} catch (PDOException $e) {

    die("Connection failed: " . $e->getMessage());

}

function saveUser($name, $lastname, $birthday) {

    global $conn;

    try {

        // Преобразуем дату рождения в timestamp

        $timestamp = strtotime($birthday);

        // Вставляем данные о пользователе

        $stmt = $conn->prepare("INSERT INTO users (user\_name, user\_lastname, user\_birthday\_timestamp) VALUES (:name, :lastname, :timestamp)");

        $stmt->execute([

            ':name' => $name,

            ':lastname' => $lastname,

            ':timestamp' => $timestamp

        ]);

        return "Пользователь c именем " . $name . " и " .  " фамилией " . $lastname . " сохранён";

    } catch (PDOException $e) {

        return "Ошибка при сохранении пользователя: " . $e->getMessage();

    }

}

function updateUser($id, $name) {

    global $conn;

    if (!is\_numeric($id)) {

        return "ID должен быть числом.";

    }

    try {

        // Проверяем существование пользователя

        $stmt = $conn->prepare("SELECT \* FROM users WHERE id\_user=:id");

        $stmt->execute(['id' => $id]);

        if ($stmt->rowCount() == 0) {

            return "Пользователь с таким ID не найден.";

        }

        // Обновляем поле name

        $stmt = $conn->prepare("UPDATE users SET user\_name=:name WHERE id\_user=:id");

        $stmt->execute(['name' => $name, 'id' => $id]);

        return "Имя пользователя c id " . $id . " успешно обновлено.";

    } catch (PDOException $e) {

        return "Ошибка при обновлении пользователя: " . $e->getMessage();

    }

}

function deleteUser($id) {

    global $conn;

    if (!is\_numeric($id)) {

        return "ID должен быть числом.";

    }

    try {

        // Проверяем существование пользователя

        $stmt = $conn->prepare("SELECT \* FROM users WHERE id\_user=:id");

        $stmt->execute(['id' => $id]);

        if ($stmt->rowCount() == 0) {

            return "Пользователь с таким ID не найден.";

        }

        // Удаление пользователя

        $stmt = $conn->prepare("DELETE FROM users WHERE id\_user=:id");

        $stmt->execute(['id' => $id]);

        return "Пользователь с ID ." . $id . "удалён";

    } catch (PDOException $e) {

        return "Ошибка при удалении пользователя: " . $e->getMessage();

    }

}

// Определяем маршрут

$path = parse\_url($\_SERVER['REQUEST\_URI'], PHP\_URL\_PATH);

//echo $path;

switch ($path) {

    case '/user/save/':

        if ($\_SERVER['REQUEST\_METHOD'] === 'GET') {

            if (isset($\_GET['name']) && isset($\_GET['lastname']) && isset($\_GET['birthday'])) {

                echo saveUser($\_GET['name'], $\_GET['lastname'], $\_GET['birthday']);

            } else {

                echo "Не указаны обязательные параметры.";

            }

        }

        break;

    case '/user/update/':

        if ($\_SERVER['REQUEST\_METHOD'] === 'GET') {

            if (isset($\_GET['id']) && isset($\_GET['name'])) {

                echo updateUser($\_GET['id'], $\_GET['name']);

            } else {

                echo "Не указаны обязательные параметры.";

            }

        }

        break;

    case '/user/delete/':

        if ($\_SERVER['REQUEST\_METHOD'] === 'GET') {

            if (isset($\_GET['id'])) {

                echo deleteUser($\_GET['id']);

            } else {

                echo "Не указан параметр ID.";

            }

        }

        break;

    default:

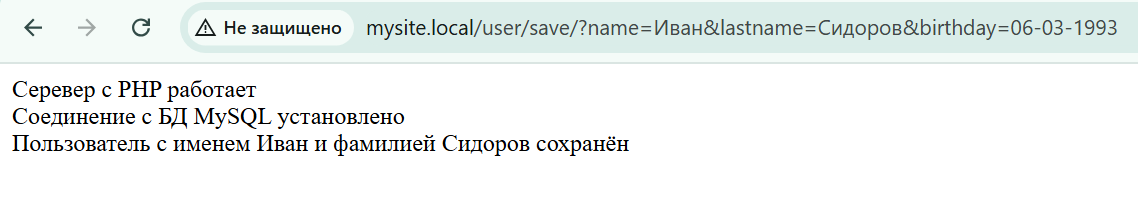
        echo "Неверный маршрут.";

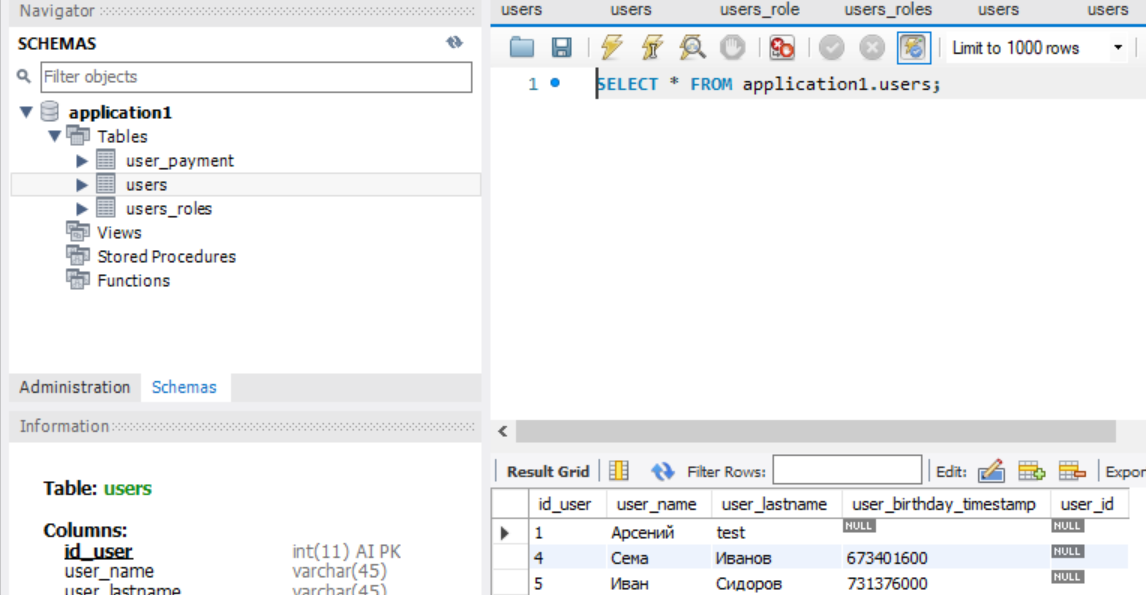
        break;

}

Теперь попробуем протестировать добавление пользователя с именем и фамилией в таблицу users через GET-запрос, видим, что пользователь добавлен и есть в таблице users:

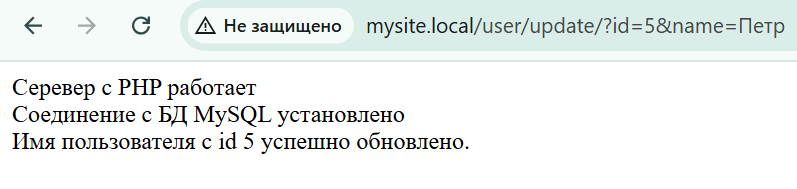
http://mysite.local/user/save/?name=Иван&lastname=Сидоров&birthday=06-03-1993



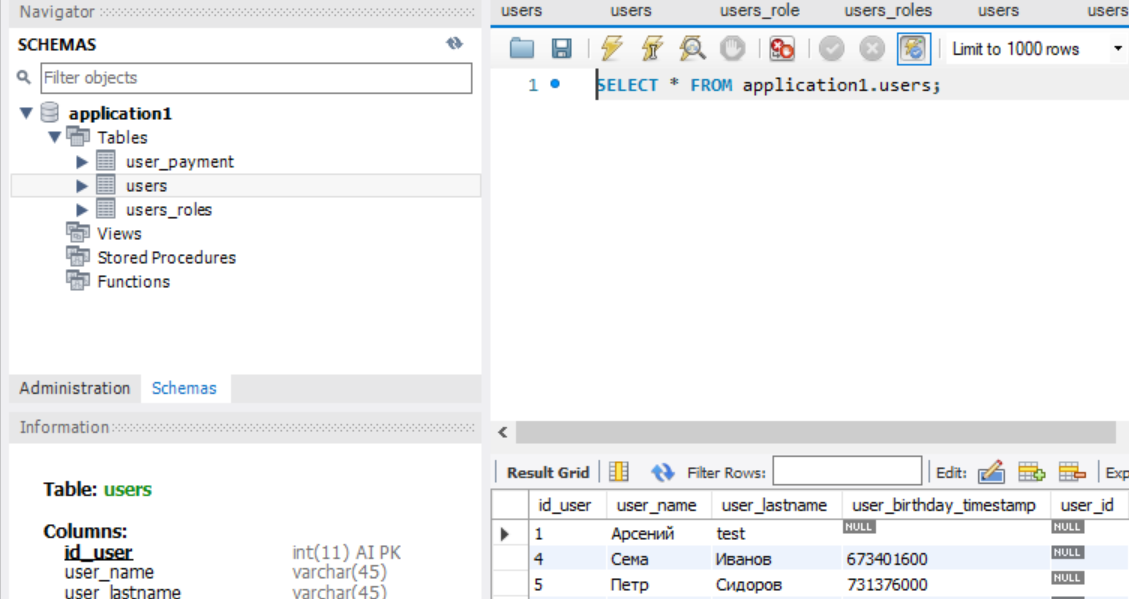


Теперь протестируем GET-запрос обновления имени пользователя:

<http://mysite.local/user/update/?id=5&name=Петр>

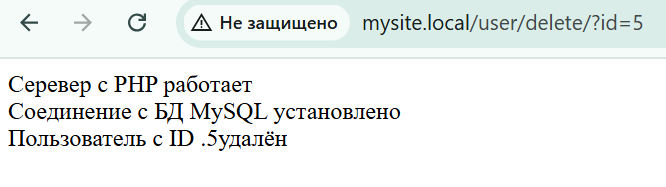


Видим, что в базе имя поменялось:

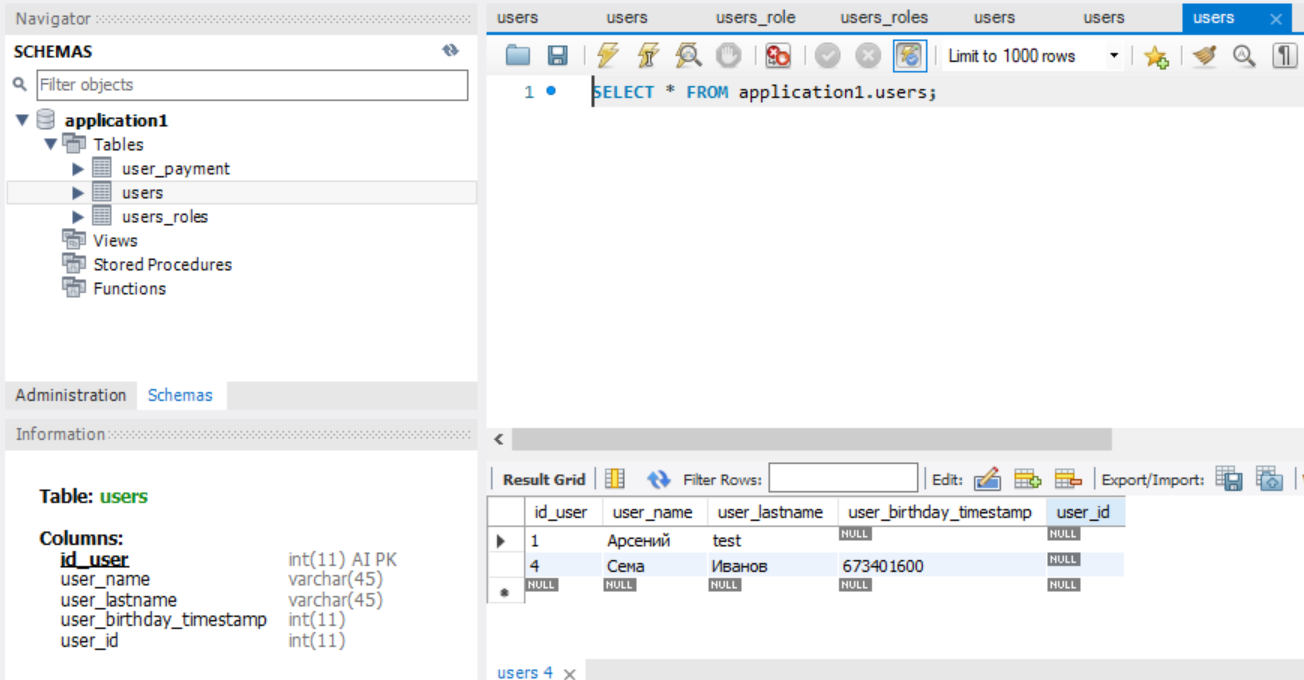


Теперь протестируем удаление через GET-запрос пользователя:

<http://mysite.local/user/delete/?id=5>



Видим, что пользователь удален:



Создадим логику обработки исключений в файле index.php, изменив файл index.php:

require\_once('./vendor/autoload.php');

use Geekbrains\Application1\Application\Application;

use Geekbrains\Application1\Application\Render;

try{

     $app = new Application();

     echo $app->run();

 }

 catch(Exception $e){

     echo Render::renderExceptionPage($e);

 }

Также создадим файл с классом приложения: src/Application/Application.php:

<?php

namespace Geekbrains\Application1\Application;

use Exception;

use Geekbrains\Application1\Infrastructure\Config;

use Geekbrains\Application1\Infrastructure\Storage;

class Application {

    private const APP\_NAMESPACE = 'Geekbrains\Application1\Domain\Controllers\\';

    private string $controllerName;

    private string $methodName;

    public static Config $config;

    public static Storage $storage;

    public function \_\_construct(){

        self::$config = new Config();

        self::$storage = new Storage();

    }

    public function run(): string{

    $routeArray = explode('/', $\_SERVER['REQUEST\_URI']);

    if (isset($routeArray[1]) && $routeArray[1] !== '') {

        $controllerName = $routeArray[1];

    } else {

        $controllerName = 'page';

    }

    $this->controllerName = self::APP\_NAMESPACE . ucfirst($controllerName) . 'Controller';

    if (class\_exists($this->controllerName)) {

        // Пытаемся вызвать метод

        if (isset($routeArray[2]) && $routeArray[2] !== '') {

            $methodName = $routeArray[2];

        } else {

            $methodName = 'index';

        }

        $this->methodName = 'action' . ucfirst($methodName);

        if (method\_exists($this->controllerName, $this->methodName)) {

            $controllerInstance = new $this->controllerName();

            // Разбираем параметры

            $params = [];

            $requestParams = $\_GET;

            foreach ($requestParams as $key => $value) {

                if ($key === 'id') {

                    $params[] = intval($value); // Приводим значение id к типу int

                } else {

                    $params[] = $value;

                }

            }

            // Передаем параметры в метод контроллера

            return call\_user\_func\_array([$controllerInstance, $this->methodName], $params);

        } else {

            throw new Exception("Метод {$this->methodName} не существует", 404);

        }

    } else {

        throw new Exception("Класс {$this->controllerName} не существует", 404);

    }

}

}

Создадим также файл с классом прорисовки данных на веб-странице Render srs\Application\Render:

<?php

namespace Geekbrains\Application1;

use Twig\Loader\FilesystemLoader;

use Twig\Environment;

class Render {

    private string $viewFolder = '/src/Views/';

    private FilesystemLoader $loader;

    private Environment $environment;

    public function \_\_construct(){

        $this->loader = new FilesystemLoader(dirname(\_\_DIR\_\_) . $this->viewFolder);

        $this->environment = new Environment($this->loader, [

           // 'cache' => $\_SERVER['DOCUMENT\_ROOT'].'/cache/',

        ]);

    }

    public function renderPage(string $contentTemplateName = 'page-index.twig', array $templateVariables = []) {

        $template = $this->environment->load('/layouts/main.twig');

        $templateVariables['content\_template\_name'] = $contentTemplateName;

        $templateVariables['title'] = 'имя страницы';

        return $template->render($templateVariables);

    }

}

Строки для подключения к базе MySQL из файла index.php перенесем в файл src\Infrastructure\config.php:

[database]

DSN = "mysql:dbname=application1;host=database"

USER = "application\_user"

PASSWORD = "geekbrains!23"

Для обработки-парсинга данных этого ini-файла создадим класс Config в файле src/Infrastructure/Config.php:

<?php

namespace Geekbrains\Application1\Infrastructure;

class Config {

    private string $defaultConfigFile = "/src/config/config.ini";

    private array $applicationConfiguration = [];

    public function \_\_construct(){

        $address = $\_SERVER['DOCUMENT\_ROOT'] . $this->defaultConfigFile;

        if(file\_exists($address) && is\_readable($address)){

            $this->applicationConfiguration = parse\_ini\_file($address, true);

        }

        else {

            throw new \Exception("Файл конфигурации не найден");

        }

    }

    public function get(): array {

        return $this->applicationConfiguration;

    }

}

Для соединения с базой данных строки соединения и работы с базой MySQL создадим класс хранилища Storage в файле src\Infrastructure\storage.php:

<?php

namespace Geekbrains\Application1\Infrastructure;

use Geekbrains\Application1\Application\Application;

use PDO;

class Storage

{

    private static ?Storage $instance = null; // Статическое свойство для хранения экземпляра

    private PDO $connection;

    public function \_\_construct() {

        $this->connection = new PDO(

            Application::$config->get()['database']['DSN'],

            Application::$config->get()['database']['USER'],

            Application::$config->get()['database']['PASSWORD'],

            array(

                PDO::MYSQL\_ATTR\_INIT\_COMMAND => "SET NAMES utf8"

            )

        );

    }

    public static function getInstance(): Storage {

        if (self::$instance === null) {

            self::$instance = new Storage();

        }

        return self::$instance;

    }

    public function findUserById(int $id): ?array {

        $stmt = $this->connection->prepare("SELECT \* FROM users WHERE id\_user = :id");

        $stmt->execute(['id' => $id]);

        return $stmt->fetch(PDO::FETCH\_ASSOC);

    }

    public function addUser(string $name, string $lastname, int $birthdayTimestamp): bool {

        $stmt = $this->connection->prepare("INSERT INTO users (user\_name, user\_lastname, user\_birthday\_timestamp) VALUES (:name, :lastname, :timestamp)");

        return $stmt->execute([

            ':name' => $name,

            ':lastname' => $lastname,

            ':timestamp' => $birthdayTimestamp

        ]);

    }

    public function updateUserName(int $id, string $newName): bool {

        $stmt = $this->connection->prepare("UPDATE users SET user\_name=:name WHERE id\_user=:id");

        return $stmt->execute([

            ':name' => $newName,

            ':id' => $id

        ]);

    }

    public function deleteUser(int $id): bool {

        $stmt = $this->connection->prepare("DELETE FROM users WHERE id\_user=:id");

        return $stmt->execute([':id' => $id]);

    }

    public function get(): PDO {

        return $this->connection;

    }

}

Также создадим файл UserController.php с классом контроллера работы с пользователями UserController в котором обрабатываются действия по добавлению пользователя, изменению его имени и удалению пользователя:

<?php

namespace Geekbrains\Application1\Domain\Controllers;

use Geekbrains\Application1\Infrastructure\Storage;

use Exception;

class UserController

{

    /\*\* @var Storage \*/

    private $storage;

    public function \_\_construct()

    {

        $this->storage = Storage::getInstance();

    }

    /\*\*

     \* Действие для добавления нового пользователя.

     \*

     \* @param string $name Имя пользователя.

     \* @param string $lastname Фамилия пользователя.

     \* @param string $birthday Дата рождения пользователя.

     \* @throws Exception

     \* @return string Сообщение о результате операции.

     \*/

    public function actionSave(string $name, string $lastname, string $birthday): string

    {

        try {

            // Преобразуем дату рождения в timestamp

            $timestamp = strtotime($birthday);

            // Вставляем данные о пользователе

            $result = $this->storage->addUser($name, $lastname, $timestamp);

            return "Пользователь с именем $name и фамилией $lastname сохранён.";

        } catch (\Throwable $e) {

            throw new Exception("Ошибка при сохранении пользователя: " . $e->getMessage(), 500, $e);

        }

    }

    /\*\*

     \* Действие для обновления имени пользователя.

     \*

     \* @param int $id Идентификатор пользователя.

     \* @param string $newName Новое имя пользователя.

     \* @throws Exception

     \* @return string Сообщение о результате операции.

     \*/

    public function actionUpdate(int $id, string $newName): string

    {

        try {

            // Проверяем существование пользователя

            $user = $this->storage->findUserById($id);

            if (!$user) {

                throw new Exception("Пользователь с таким ID не найден.", 404);

            }

            // Обновляем имя пользователя

            $this->storage->updateUserName($id, $newName);

            return "Имя пользователя с ID $id успешно обновлено на $newName.";

        } catch (\Throwable $e) {

            throw new Exception("Ошибка при обновлении пользователя: " . $e->getMessage(), 500, $e);

        }

    }

    /\*\*

     \* Действие для удаления пользователя.

     \*

     \* @param int $id Идентификатор пользователя.

     \* @throws Exception

     \* @return string Сообщение о результате операции.

     \*/

    public function actionDelete(int $id): string

    {

        try {

            // Проверяем существование пользователя

            $user = $this->storage->findUserById($id);

            if (!$user) {

                throw new Exception("Пользователь с таким ID не найден.", 404);

            }

            // Удаление пользователя

            $this->storage->deleteUser($id);

            return "Пользователь с ID $id был удалён.";

        } catch (\Throwable $e) {

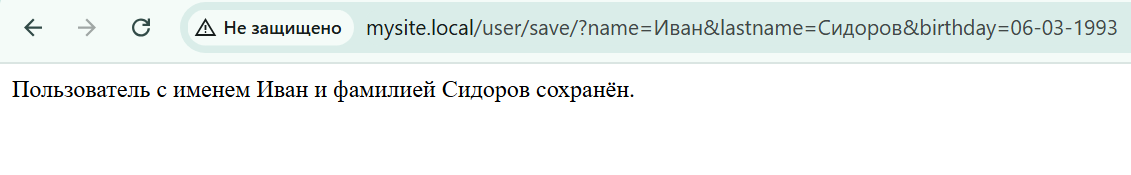
            throw new Exception("Ошибка при удалении пользователя: " . $e->getMessage(), 500, $e);

        }

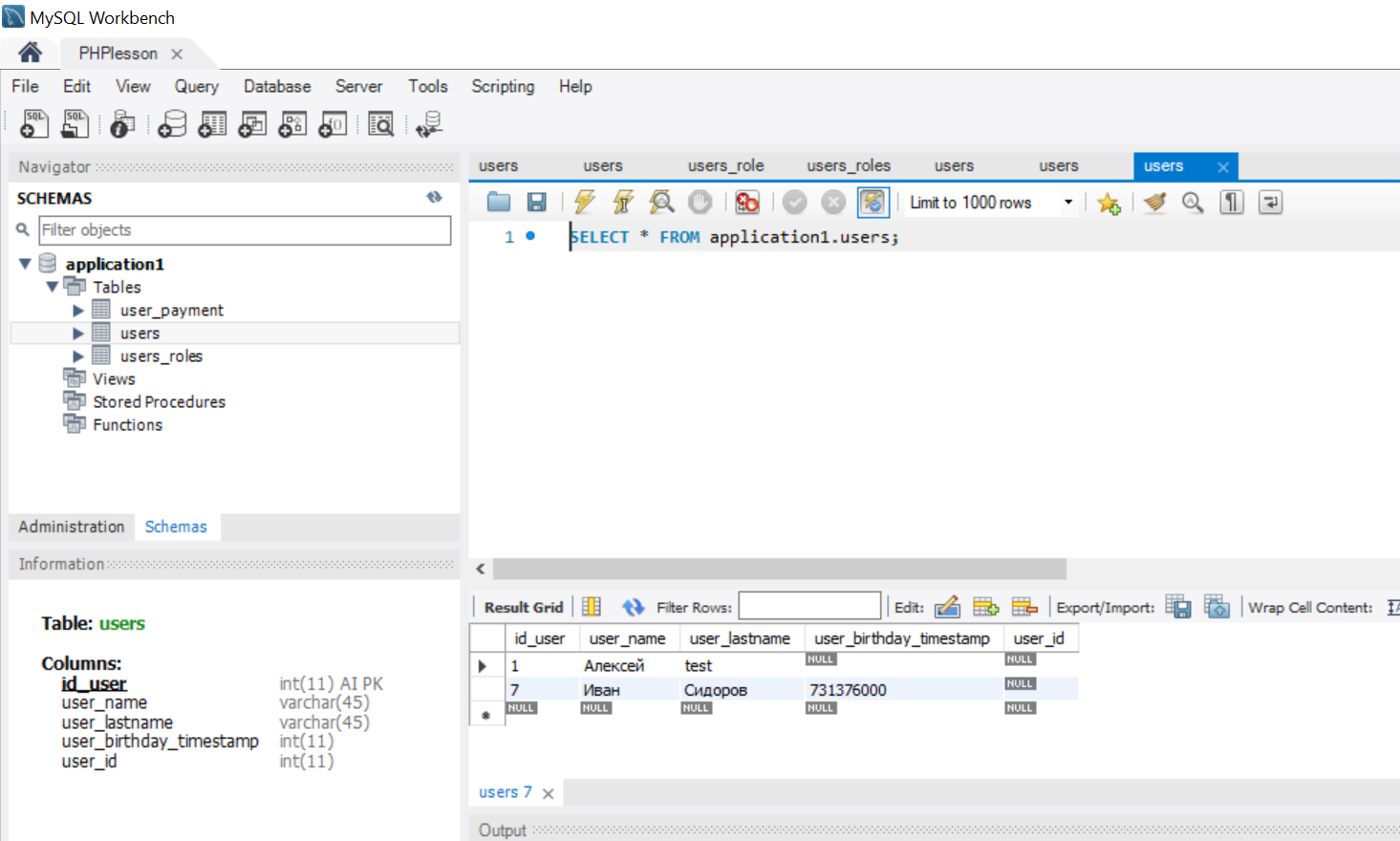
    }

}

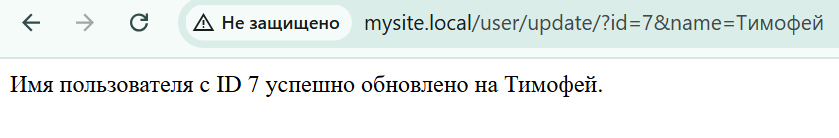
Протестируем теперь добавление пользователя через GET-запрос http://mysite.local/user/save/?name=Иван&lastname=Сидоров&birthday=06-03-1993



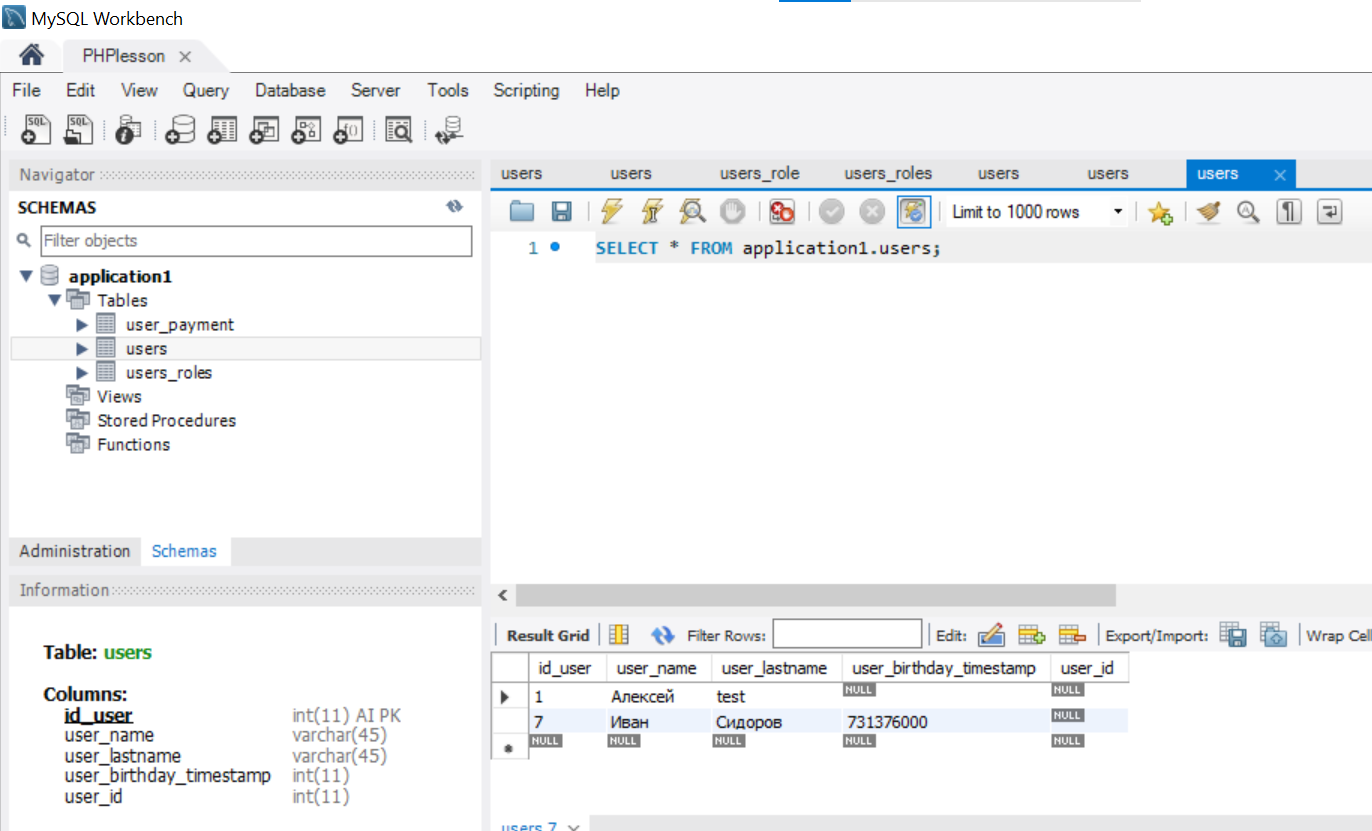
Видим, что пользователь Иван Сидоров сохранен в базе MySQL:



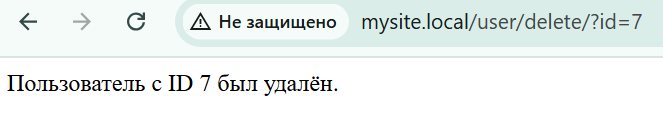
Теперь протестируем изменение имени пользователя GET-запросом http://mysite.local/user/update/?id=7&name=Тимофей:



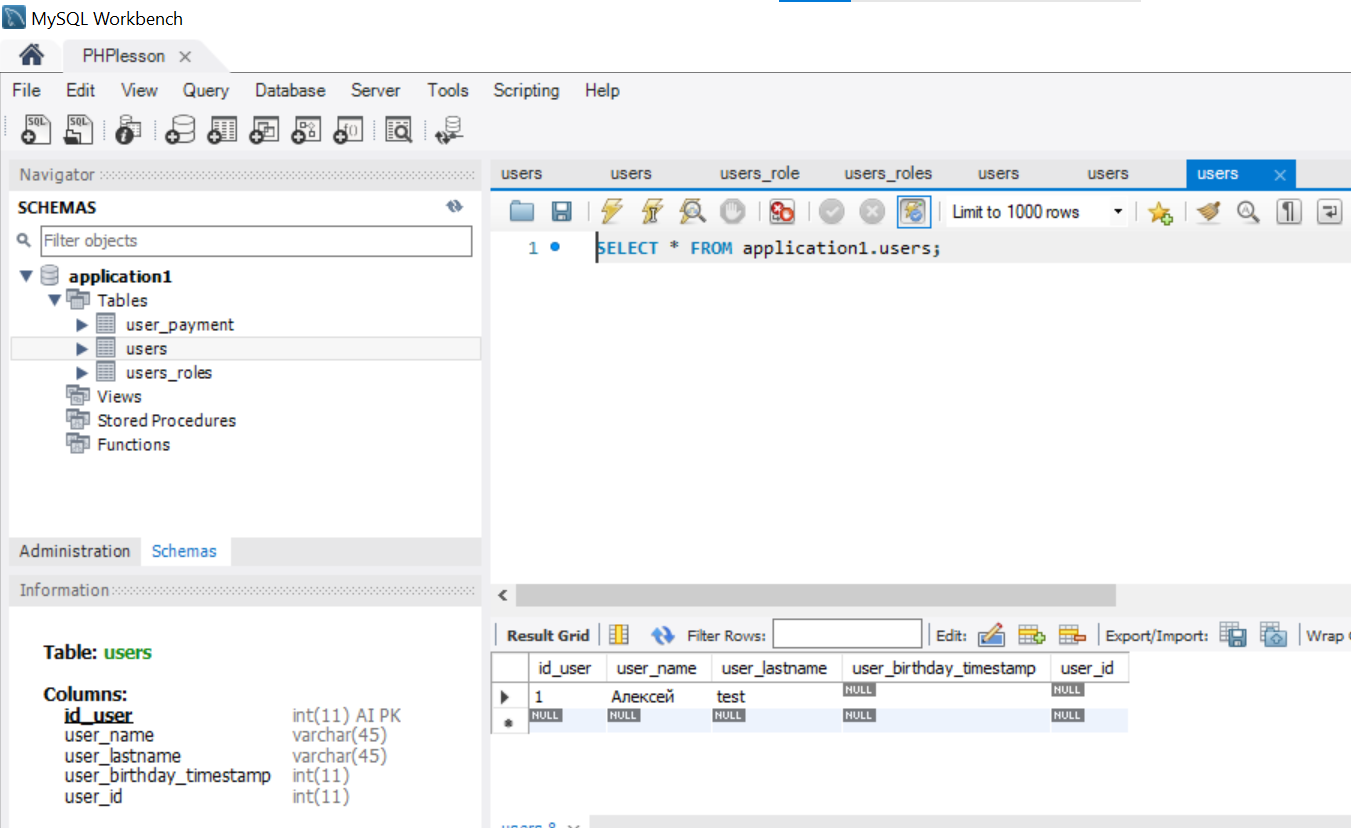
Видим, что в базе MySQL имя пользователя с ID=7 поменялось с Иван на Тимофей:



Теперь протестируем GET-запрос удаления пользователя http://mysite.local/user/delete/?id=7:



Видим, что из базы MySQL пользователь с ID=7 также удален:



Также по условию задачи кроме GET-запросов нужно было использовать обработку исключений через код:

try{  
 $app = new Application();  
 echo $app->run();  
}  
catch(Exception $e){  
 echo Render::renderExceptionPage($e);  
}

Данный код добавлен в файле index.php:

<?php

require\_once('./vendor/autoload.php');

use Geekbrains\Application1\Application\Application;

use Geekbrains\Application1\Application\Render;

try{

     $app = new Application();

     echo $app->run();

 }

 catch(Exception $e){

     echo Render::renderExceptionPage($e);

 }

?>

