Цель работы:Изучить платформу Node js и фреймворк Express.

Задачи: Продолжить разработку веб приложения из Л.р.№6: продумать и реализовать структуру БД для хранения данных разрабатываемого приложения. Добавить подключение к БД в разрабатываемое приложение. Преобразовать страницы со статическим контентом в шаблоны и реализовать их заполнение информацией, хранящейся в БД.

Ход работы:

server.js:

```
const express = require('express');
const bodyParser = require('body-parser');
const mysql = require('mysql');
const path = require('path');

const app = express();
const port = 3000;

// MySQL database connection
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '1111',
  database: 'dz2'
});

db.connect((err) => {
  if (err) {
    throw err;
  }
  console.log('MySQL Connected...');
});

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'public')));

// Handle SQL query from form
app.post('/execute-sql', (req, res) => {
  const command = req.body.command;
  db.query(command, (error, results) => {
    if (error) {
      res.send(Ошибка выполнения запроса: ${error.sqlMessage}`);
    } else {
      let output = '<h2>Результаты запроса:</h2>';
      if (results.length > 0) {
        output += '<table border="1"><tr>';
        for (const key in results[0]) {
          output += `<th>${key}</th>`;
        }
        output += '</tr>';
        results.forEach(row => {
          output += '<tr>';
          for (const key in row) {
            output += `<td>${row[key]}</td>`;
          }
          output += '</tr>';
        });
        output += '</table>';
      } else {
        output += '0 результатов';
      }
      res.send(output);
    }
  });
});

// Serve the HTML file
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Database Console</title>
</head>
<body>
  <h1>Database Console</h1>

  <!-- Форма для ввода SQL-запроса -->
  <form id="sql-form" method="post" action="/execute-sql">
    <label for="command">Введите SQL-запрос:</label><br>
    <textarea id="command" name="command" rows="10" cols="100">select *
from;</textarea><br>
```

```
    <input type="submit" value="Выполнить">
  </form>

  <button id="join-btn" type="button">Сложный JOIN</button>
  <button id="select-btn" type="button">Сложный запрос</button>
  <button id="request-and-subrequest-btn" type="button">Запрос с
подзапросом</button>
  <button id="simple-select-form-btn" type="button">Простой запрос</button>
  <button id="update-btn" type="button">Выполнить UPDATE</button>
  <button id="insert-workers-btn" type="button">Выполнить INSERT
Workers</button>
  <button id="insert-ticket-btn" type="button">Выполнить INSERT Ticket</button>
  <button id="insert-ticket-lines-btn" type="button">Выполнить INSERT
Ticket_lines</button>
  <button id="delete-ticket-lines-btn" type="button">Выполнить DELETE
Ticket_lines</button>
  <button id="select-worker-material-responsibility-btn" type="button">Посмотреть
Мат. Отв. работников</button>
  <button id="select-cost-pictures-from-holes-btn" type="button">Ценность
зала</button>

  <script>
    // Получаем ссылки на элементы формы и кнопки
    const sqlForm = document.getElementById('sql-form');
    const simpleSelect = document.getElementById('simple-select-form-btn');
    const commandInput = document.getElementById('command');
    const joinBtn = document.getElementById('join-btn');
    const selectBtn = document.getElementById('select-btn');
    const updateBtn = document.getElementById('update-btn');
    const insertWorkersBtn = document.getElementById('insert-workers-btn');
    const insertTicketBtn = document.getElementById('insert-ticket-btn');
    const insertTicketLinesBtn = document.getElementById('insert-ticket-lines-btn');
    const deleteTicketLinesBtn = document.getElementById('delete-ticket-lines-btn');
    const selectWorkerMaterialResponsibility =
document.getElementById('select-worker-material-responsibility-btn');
    const selectCostPicturesFromHoles =
document.getElementById('select-cost-pictures-from-holes-btn');
    const requestAndSubrequest =
document.getElementById('request-and-subrequest-btn');

    joinBtn.addEventListener('click', function() {
      const joinCommand = `SELECT Ticket.TL_ID, Ticket.T_DATA,
Ticket.T_CODE, Visitor.V_ID, Visitor.V_FIO, workers.W_ID, workers.W_FIO
FROM Ticket
INNER JOIN Ticket_lines ON Ticket.TL_ID = Ticket_lines.TL_ID
INNER JOIN Visitor ON Ticket.T_V_ID = Visitor.V_ID AND Ticket.T_V_FIO =
Visitor.V_FIO
INNER JOIN workers ON Ticket_lines.TL_W_ID = workers.W_ID AND
Ticket.T_W_FIO = workers.W_FIO
ORDER BY Ticket.T_DATA;`;
      commandInput.value = joinCommand;
    });

    selectBtn.addEventListener('click', function() {
      const selectCommand = `SELECT workers.W_ID AS Worker_ID,
workers.W_FIO AS Worker_Name, COUNT(Ticket.TL_ID) AS Tickets_Sold,
SUM(Ticket_lines.TL_SELL) AS Total_Sales
FROM Ticket INNER
JOIN Ticket_lines ON Ticket.TL_ID = Ticket_lines.TL_ID
INNER JOIN workers ON Ticket_lines.TL_W_ID = workers.W_ID
WHERE Ticket.T_DATA BETWEEN '2023-01-01' AND '2025-01-01'
GROUP BY workers.W_FIO
ORDER BY Tickets_Sold DESC;`;
      commandInput.value = selectCommand;
    });

    updateBtn.addEventListener('click', function() {
      const updateCommand = `UPDATE Visitor
SET V_FIO = 'New name'
WHERE V_ID = 000;`;
      commandInput.value = updateCommand;
    });

    insertWorkersBtn.addEventListener('click', function() {
      const insertWorkersCommand = `INSERT INTO Workers (W_ID, W_FIO,
W_PD, W_TBL, W_SNT, W_INN, W_NP, W_PM)
VALUES (1, 'Worker name', 'Passport data', 'Position', 1, 1111111, 'Residential
address', 11223344);`;
      commandInput.value = insertWorkersCommand;
    });

    insertTicketBtn.addEventListener('click', function() {
      const insertTicketCommand = `INSERT INTO Ticket (TL_ID, T_DATA,
T_CODE, T_V_ID, T_V_FIO, T_W_FIO)
VALUES (1, '2024-06-03', 101, 2, 'Visitor name', 'Worker name');`;
      commandInput.value = insertTicketCommand;
    });

    insertTicketLinesBtn.addEventListener('click', function() {
```

```
    const insertTicketLinesCommand = `INSERT INTO Ticket_lines (TL_ID,
TL_W_ID, TL_SELL, TL_VIS)
VALUES (1, 123, 456, 789);`;
    commandInput.value = insertTicketLinesCommand;
  });

  deleteTicketLinesBtn.addEventListener('click', function() {
    const deleteTicketLinesCommand = `DELETE FROM Ticket_lines
WHERE TL_ID = 123;`;
  commandInput.value = deleteTicketLinesCommand;
  });

  selectWorkerMaterialResponsibility.addEventListener('click', function() {
    const selectWorkerMaterialResponsibility = `SELECT workers.W_ID AS
worker_id, workers.W_FIO AS worker_name, SUM(Picture.P_COST) AS
total_value
FROM workers
JOIN holes ON workers.W_ID = holes.H_W_ID
JOIN Picture ON holes.H_ID = Picture.P_H_ID
GROUP BY workers.W_ID, workers.W_FIO;`;
  commandInput.value = selectWorkerMaterialResponsibility;
  });

  selectCostPicturesFromHoles.addEventListener('click', function() {
    const selectCostPicturesFromHoles = `SELECT holes.H_ID AS hall_id,
SUM(Picture.P_COST) AS total_value, GROUP_CONCAT(Picture.P_NM) AS
painting_names
FROM holes
JOIN Picture ON holes.H_ID = Picture.P_H_ID
GROUP BY holes.H_ID
ORDER BY total_value DESC;`;
commandInput.value = selectCostPicturesFromHoles;
  });

  simpleSelect.addEventListener('click', function() {
    const simpleSelect = `SELECT W_ID
      const simpleSelect = `SELECT W_ID, W_FIO, W_PM
FROM workers
WHERE W_PM > 50000;`;
commandInput.value = simpleSelect;
    });

    requestAndSubrequest.addEventListener('click', function() {
      const requestAndSubrequest = `SELECT Picture.P_NM AS Picture_Name
FROM Picture
WHERE Picture.P_H_ID = (
  SELECT holes.H_ID
  FROM holes
  JOIN Picture ON holes.H_ID = Picture.P_H_ID
  GROUP BY holes.H_ID
  ORDER BY COUNT(Picture.P_G_ID) DESC
  LIMIT 1 );`;
commandInput.value = requestAndSubrequest;
    });

  </script>
 </body>
 </html>
```

Результат:

Database Console

Введите SQL-запрос:

```
SELECT Ticket.TL_ID, Ticket.T_DATA, Ticket.T_CODE, Visitor.V_ID, Visitor.V_FIO, workers.W_ID, workers.W_FIO
FROM Ticket
INNER JOIN Ticket_lines ON Ticket.TL_ID = Ticket_lines.TL_ID
INNER JOIN Visitor ON Ticket.T_V_ID = Visitor.V_ID AND Ticket.T_V_FIO = Visitor.V_FIO
INNER JOIN workers ON Ticket_lines.TL_W_ID = workers.W_ID AND Ticket.T_W_FIO = workers.W_FIO
ORDER BY Ticket.T_DATA;
```

Выполнить | Сложный JOIN | Сложный запрос | Запрос с подзапросом | Простой запрос | Выполнить UPDATE | Выполнить INSERT Workers | Выполнить INSERT Ticket | Выполнить INSERT Ticket_lines | Выполнить DELETE Ticket_lines | Посмотреть Мат. Отв. работников | Ценность зала

Результаты запроса:

TL_ID	T_DATA	T_CODE	V_ID	V_FIO	W_ID	W_FIO
1	2024-01-01	101	1	Ivanov Ivan Ivanovich	1	Alexsandr Alexsandrov Alexsandrovich
2	2024-01-02	102	2	Petrov Petr Petrovich	2	Dmitriy Alexeevich Popov
3	2024-01-03	502	3	Sidorov Sidor Sidorovich	3	Ekaterina Alexandrovna Volkova
4	2024-01-04	101	4	Alekseev Aleksey Alekseevich	4	Ivan Grigorievich Novikov
5	2024-01-05	102	5	Nikolaev Nikolay Nikolaevich	5	Anna Nikolaevna Smirnova
6	2024-01-06	502	6	Mikhailov Mikhail Mikhailovich	6	Ivan Pavlov Andreevich
1	2024-01-07	101	7	Fedorov Fedor Fedorovich	1	Alexsandr Alexsandrov Alexsandrovich
2	2024-01-08	102	8	Vasilev Vasiliy Vasilievich	2	Dmitriy Alexeevich Popov
3	2024-01-09	502	9	Andreev Andrey Andreevich	3	Ekaterina Alexandrovna Volkova
4	2024-01-10	101	10	Dmitriev Dmitry Dmitrievich	4	Ivan Grigorievich Novikov

Database Console

Введите SQL-запрос:

```
SELECT Picture.P_NM AS Picture_Name
FROM Picture
WHERE Picture.P_H_ID = (
    SELECT holes.H_ID
    FROM holes
    JOIN Picture ON holes.H_ID = Picture.P_H_ID
    GROUP BY holes.H_ID
    ORDER BY COUNT(Picture.P_G_ID) DESC
    LIMIT 1 );
```

Выполнить

Сложный JOIN | Сложный запрос | Запрос с подзапросом | Простой запрос | Выполнить UPDATE | Выполнить INSERT Workers | Выполнить INSERT Ticket | Выполнить INSERT Ticket_lines | Выполнить DELETE Ticket_lines | Посмотреть Мат. Отв. работников | Ценность зала

Результаты запроса:

hall_id	total_value	painting_names
2	200000000	Great Alexsandr,Mona Lisa
1	100000000	Starry Night
10	100000000	Girl with a Pearl Earring
9	90000000	The Night Watch
8	80000000	The Starry Night
7	70000000	The Last Supper
6	60000000	The Birth of Venus
5	50000000	The Persistence of Memory
4	40000000	Guernica
3	30000000	The Scream

Вывод: в результате выполнения лабораторной работы была изучена платформа Node js и фреймворк Express.

Контрольные вопросы:

1. Опишите цикл обработки запроса и формирования ответа.

Цикл обработки запроса и формирования ответа в Express.js состоит из следующих этапов:

1. Получение запроса: Сервер получает HTTP-запрос от клиента.
2. Промежуточные обработчики (middleware): Запрос проходит через все зарегистрированные middleware в порядке их добавления. Каждый middleware может:
   - Изменить объект запроса `req` или объекта ответа `res`.
   - Завершить цикл запроса, отправив ответ клиенту.
   - Вызвать `next()`, передавая управление следующему middleware.
3. Маршрутизация (routing): После прохождения через middleware запрос передается в маршрут, который соответствует пути и методу запроса (GET, POST и т.д.).
4. Формирование ответа: Обработчик маршрута обрабатывает запрос, формирует ответ и отправляет его клиенту.
5. Отправка ответа: Ответ отправляется клиенту, и цикл запроса-ответа завершается.

2. Что такое промежуточный обработчик (middleware)? Какие типы обработчиков вы знаете?

Промежуточный обработчик (middleware) — это функция, которая обрабатывает запрос и ответ в процессе их передачи по цепочке обработки в Express.js. Middleware может выполнять следующие действия:
- Выполнить код.
- Изменить объекты запроса и ответа.
- Завершить цикл запроса.
- Вызвать следующую функцию middleware в цепочке.

Типы middleware:
1. Применяемый ко всем маршрутам: Выполняется для всех запросов (например, логирование, парсинг тела запроса, управление сессиями).
2. Применяемый к определенным маршрутам: Выполняется только для запросов к определенным маршрутам (например, аутентификация, проверка прав доступа).
3. Обработчики ошибок: Специальные middleware для обработки ошибок. Они принимают четыре аргумента: `err`, `req`, `res`, `next`.

3. Как осуществляется конфигурация middleware? Приведите пример конфигурируемого обработчика.

Конфигурация middleware осуществляется через вызов функций app.use() для установки глобального middleware или app.METHOD() для установки middleware на конкретный маршрут.

Пример конфигурируемого middleware:

```
javascript:
const express = require('express');
const app = express();

// Конфигурируемый middleware для логирования запросов
function requestLogger(options) {
  return function(req, res, next) {
    if (options.logRequests) {
      console.log(`${req.method} ${req.url}`);
    }
    next();
  };
}

app.use(requestLogger({ logRequests: true }));

app.get('/', (req, res) => {
  res.send('Hello, World!');
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

4. Какие встроенные обработчики присутствуют в Express?

Встроенные middleware в Express:
- express.static: Обслуживает статические файлы.
- express.json: Обрабатывает JSON-тела запросов.
- express.urlencoded: Обрабатывает URL-кодированные тела запросов.
- express.Router: Обеспечивает функциональность маршрутизации.

5. Что такое шаблонизатор? Для чего он необходим?

Шаблонизатор — это инструмент, который позволяет генерировать HTML на основе шаблонов и данных. Шаблонизатор отделяет логику представления от бизнес-логики и упрощает создание и поддержку веб-страниц. Он позволяет динамически вставлять данные в шаблоны для формирования HTML-страниц, которые затем отправляются клиенту.

6. Какие шаблонизаторы вы знаете?

Некоторые популярные шаблонизаторы:
- Pug (ранее Jade)
- EJS
- Handlebars
- Mustache
- Nunjucks

7. Как подключить шаблонизатор в приложении Express?

Подключение шаблонизатора в Express:

```
javascript
const express = require('express');
const app = express();

// Подключение Pug
app.set('view engine', 'pug');

// Подключение EJS
// app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  res.render('index', { title: 'Home', message: 'Hello, World!' });
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

8. Опишите основные принципы работы шаблонизатора PUG.

Pug (ранее Jade) — это высокоуровневый шаблонизатор для Node.js, который использует отступы для определения структуры документа. Основные принципы работы Pug:
- Читаемый синтаксис с минимальным использованием тегов и атрибутов.
- Поддержка переменных и циклов.
- Вложенность элементов определяется отступами.

Пример Pug-шаблона:

pug
```
doctype html
html
  head
    title= title
  body
    h1= message
    p This is a Pug template.
```

9. Опишите основные принципы работы шаблонизатора EJS.

EJS (Embedded JavaScript) — это шаблонизатор, который позволяет вставлять JavaScript-код в HTML. Основные принципы работы EJS:
- Вставка JavaScript-кода внутри `<%= %>` (для вывода) и `<% %>` (для выполнения кода).
- Поддержка условных операторов и циклов.
- Возможность разделения шаблонов на частичные (partials) для повторного использования.

Пример EJS-шаблона:

html:
```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
</head>
<body>
  <h1><%= message %></h1>
  <p>This is an EJS template.</p>
</body>
</html>
```