Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Инженерно-технологическая академия
Институт компьютерных технологий и информационной безопасности
Кафедра систем автоматизированного проектирования

Отчёт по практической работе №4
по дисциплине «Разработка серверной части веб-приложений»

Работа с SQL БД в Web-приложениях Node.js

Выполнила
студентка группы КТбо3-4 _____ Е. О. Локота

Принял
доцент кафедры МОП ЭВМ _____ А. Н. Шкурко

Таганрог 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Данная практическая работа направлена на изучение и практическое использование базовых методов доступа к БД использующих SQL в web-приложениях Node.js.

Задание

В рамках выполнения практической работы требуется выполнить следующие задачи и продемонстрировать полученные навыки:

– Создать новое node.js приложение;

– Подключить к приложению библиотеки express.js и liquid;

– Разработать HTML-шаблоны для страниц приложения;

– Реализовать функциональность доступа к БД, необходимую для выполнения задания;

– Реализовать логику обработки на сервере согласно варианту задания;

Вариант задания:

Задача 1

Необходимо реализовать простое приложение со списком дел. Приложение должно иметь следующую функциональность:

- отображение дел в виде двух списков - запланированные и выполненные;

- добавление дела в список запланированных при помощи соответствующей формы;

- перенос дела из списка запланированных в список выполненных нажатием кнопки;

- удаление дела из списка выполненных нажатием кнопки.

Списки дел хранятся в БД. У дела не обязательно делать дополнительные атрибуты типа даты или категории. Достаточно будет просто названия. С авторизацией нескольких пользователей.

1 Установка MySQL

Примеры некоторых окошек, с которыми приходилось сталкиваться изображены на рисунках 1–2.

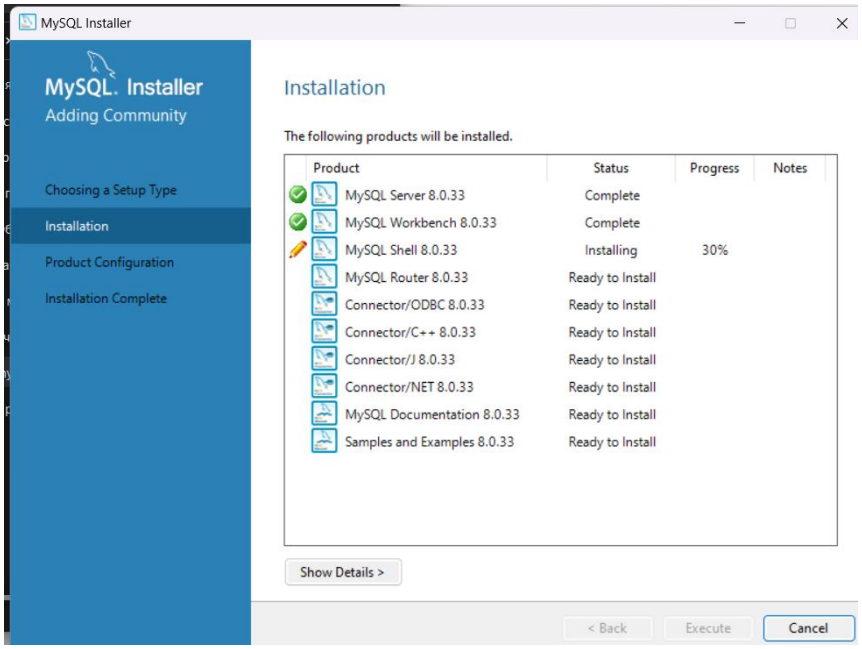

Рисунок 1 – Процесс установки

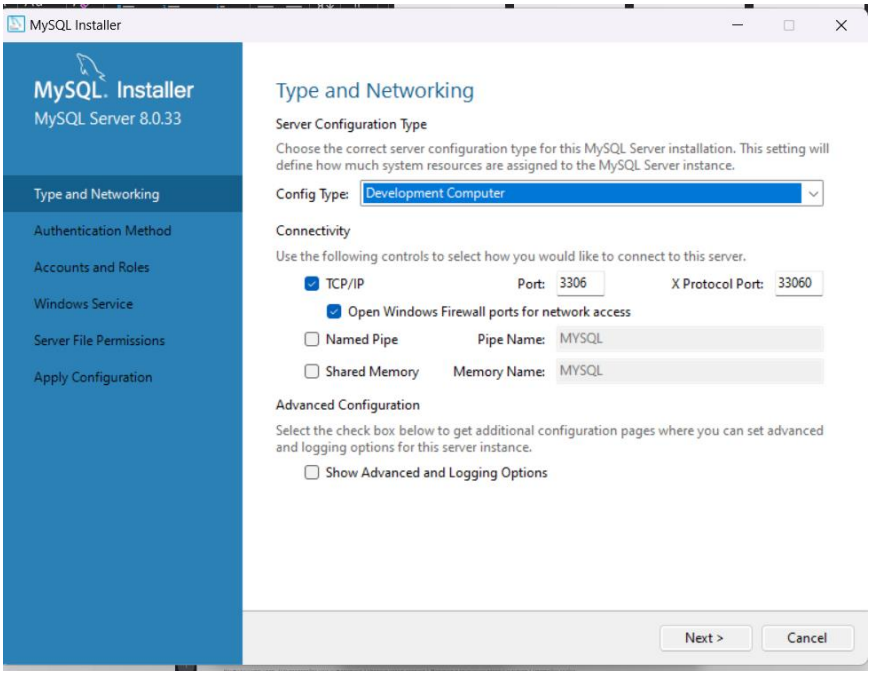

Рисунок 2 – Процесс установки

При запуске MySQL Workbench на главном экране можно увидеть существующие по умолчанию базы данных (далее – БД). Для работы с MySQL необходимо нажать правой кнопкой мыши на БД и выбрать «Start Command Line Client» (рисунок 3).

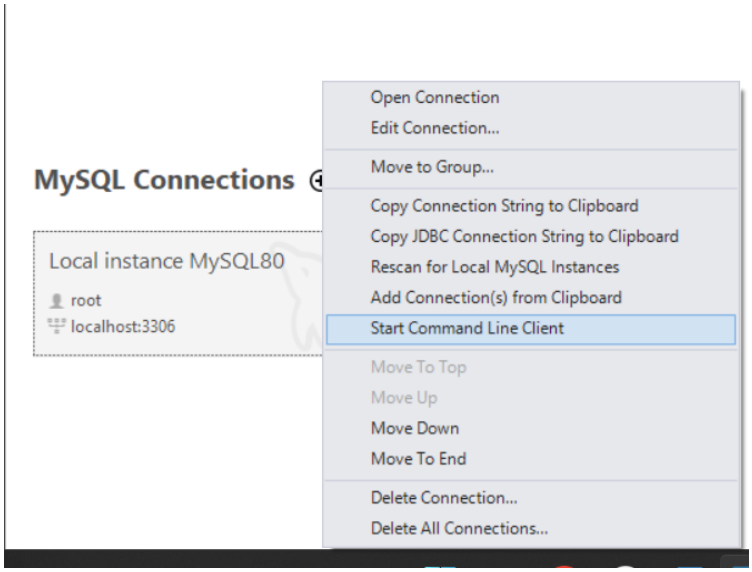

Рисунок 3 – Начало работы

После этого необходимо ввести пароль, который был придуман при установке MySQL. Все корректно запустилось (рисунок 4).

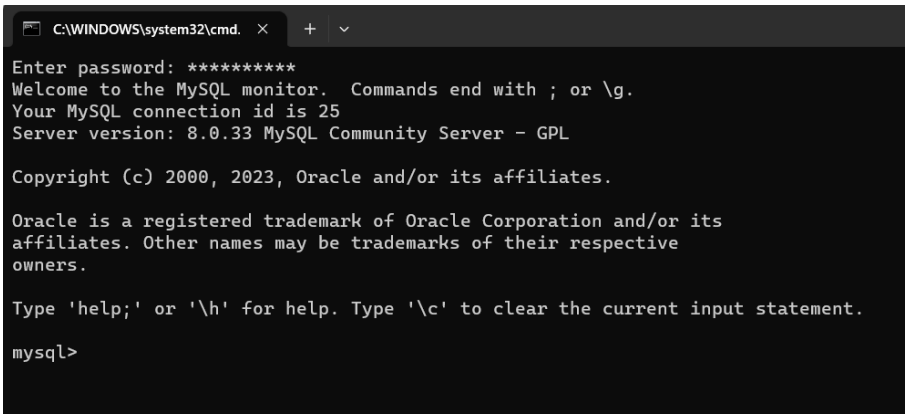

Рисунок 4 – Вход с паролем

Проверяем существующие базы данных с помощью команды «show databases» (рисунок 5).

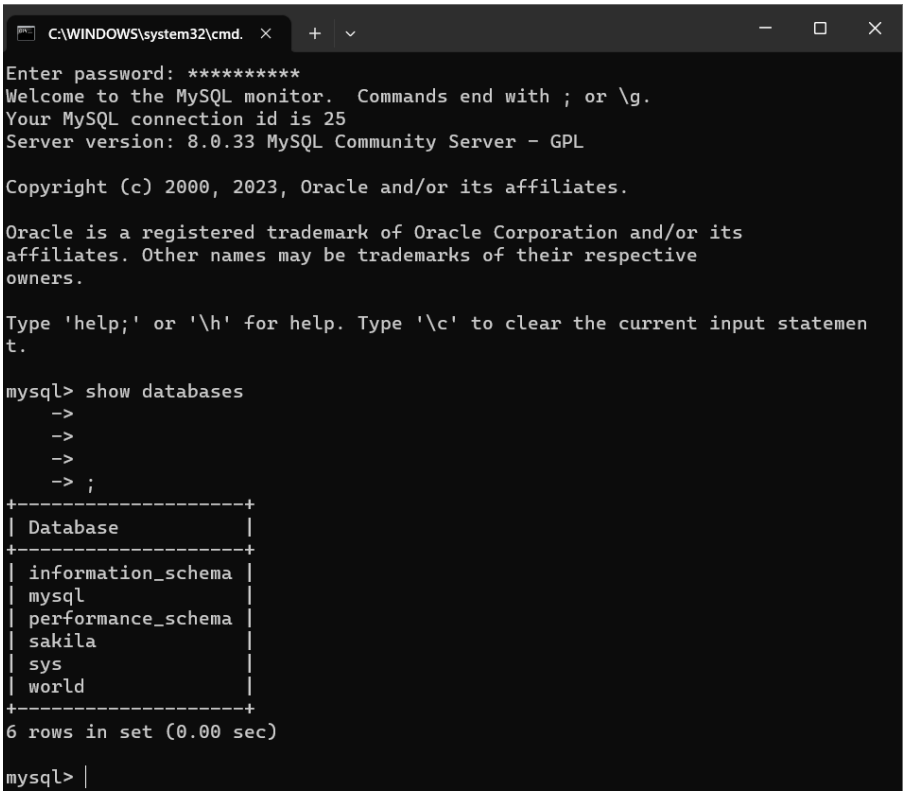

Рисунок 5 – Просмотр существующих баз данных

Создаем базу данных с кодировкой utf8 для того, чтобы можно было корректно работать с кирилицей. Это делается благодаря команде «create database» (рисунок 6).

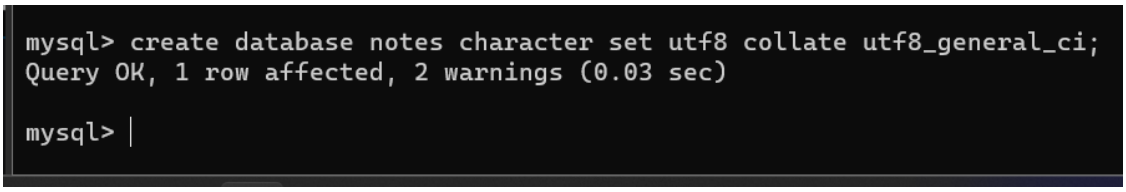

Рисунок 6 – Создание собственной базы данных

БД создана (рисунок 7).

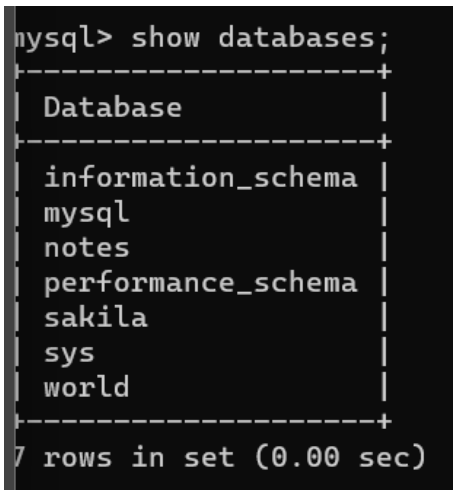

Рисунок 7 – Добавление базы данных

Кодировку созданной БД можно проверить в самой программе MySQL (рисунок 8).

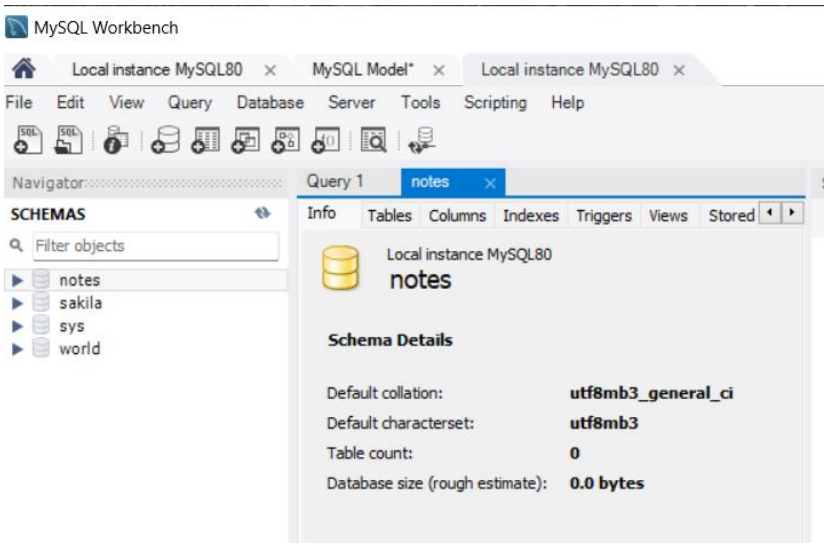

Рисунок 8 – Проверка базы данных

Далее – переходим к созданию таблиц. Первая таблица – пользователи (рисунок 9). Столбцы – id пользователя, логин и пароль. ID и пароль типа int, логин – varchar(45). ID – первичный ключ и автоматически увеличивающийся, логин и пароль – не могут быть пустыми.

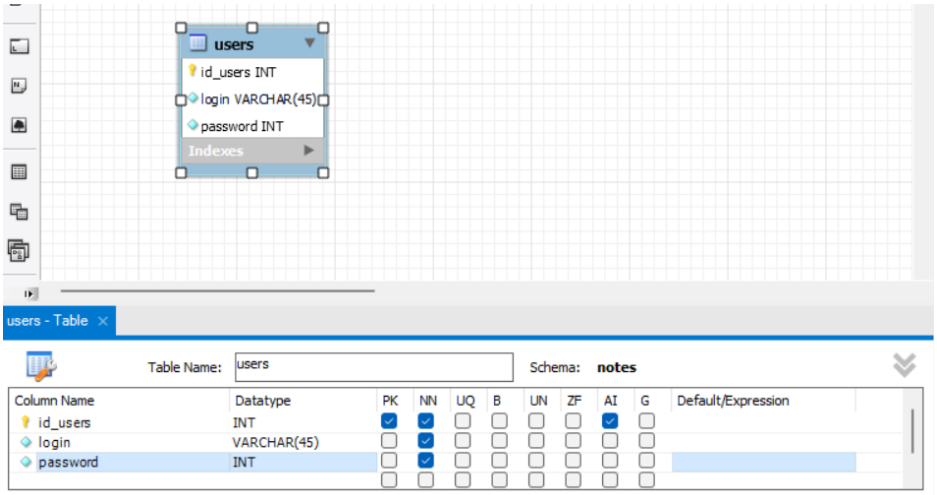

Рисунок 9 – Создание таблицы пользователей

Далее – таблица с делами (рисунок 10). Столбцы – id дела, id пользователя и само дело. Поля с ID типа int, задача – varchar(255). ID задачи – первичный ключ и автоматически увеличивающийся, ID пользователя и задача – не могут быть пустыми. Позже, в данную таблицу был добавлен столбец «выполнено» с типом данных Boolean и с условием обязательной заполненности.

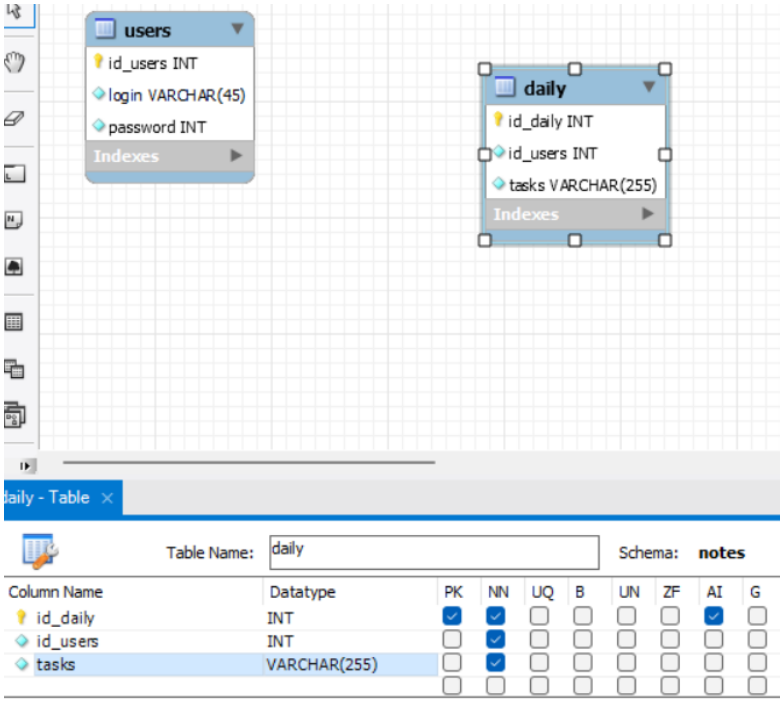

Рисунок 10 – Создание таблицы задач

Далее – настраиваем внешний ключ (id пользователя). При его удалении или изменений будет происходить каскадное удаление (обновление) (рисунок 11).

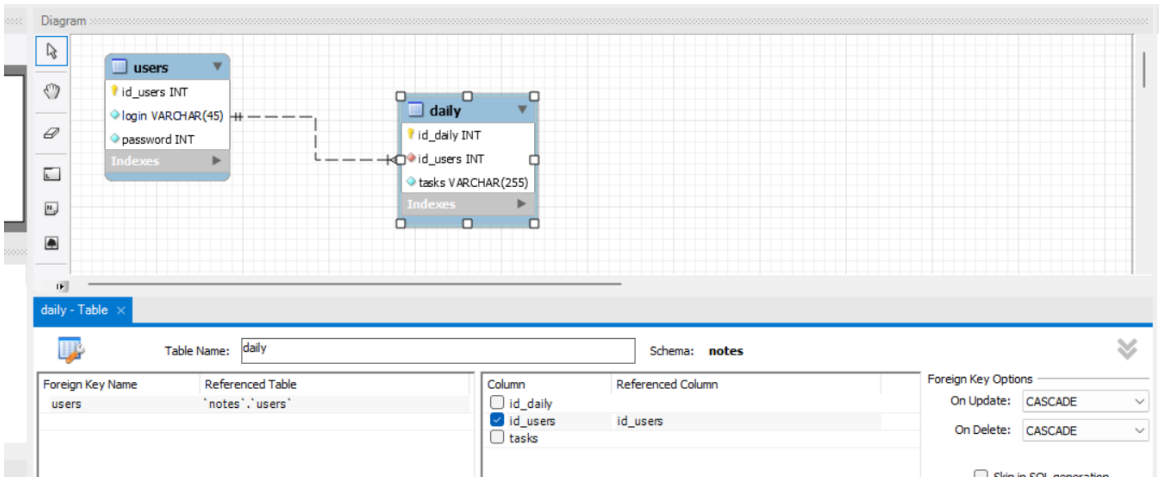

Рисунок 11 – Настройка внешнего ключа

Сохраняем скрипт (рисунок 12).

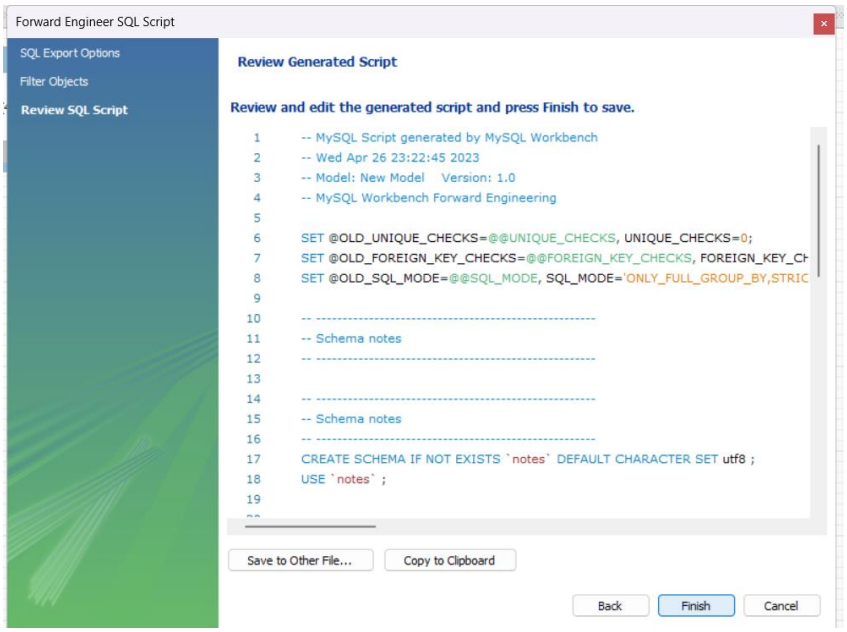

Рисунок 12 – Сохранение скрипта

Далее – устанавливаем соединение с сервером (рисунок 13).

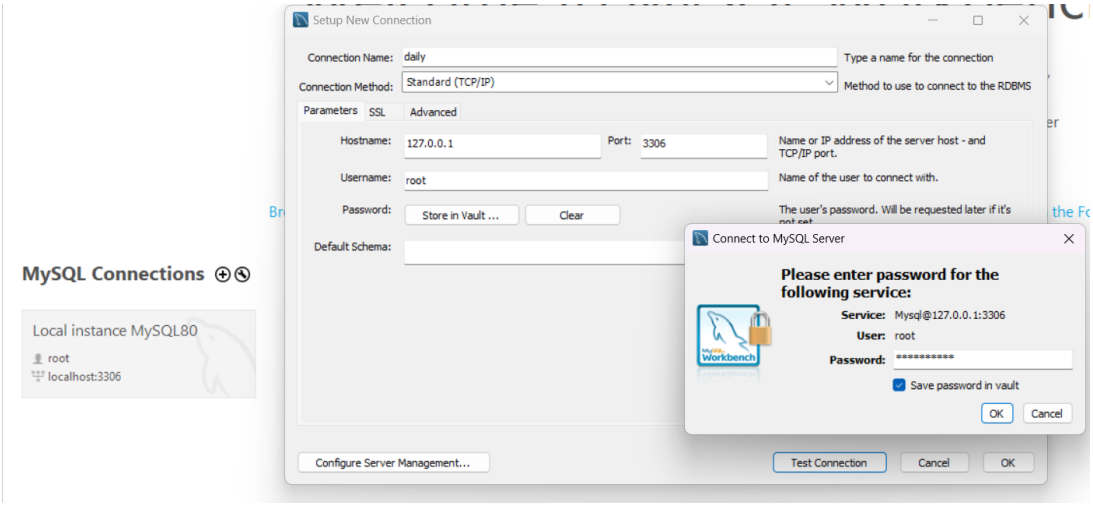

Рисунок 13 – Установление связи

Появилось новое соединение (рисунок 14).

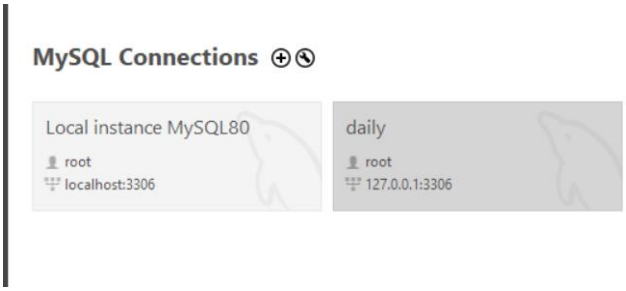

Рисунок 14 – Отображение новой связи

Открываем, загружаем скрипт (рисунок 15).

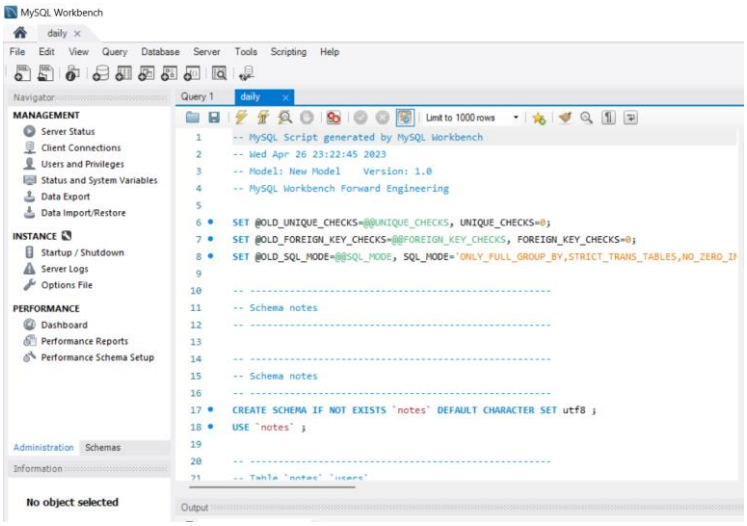

Рисунок 15 – Загруженный скрипт

Далее – запускаем его, тем самым – создаем новую БД (рисунок 16).

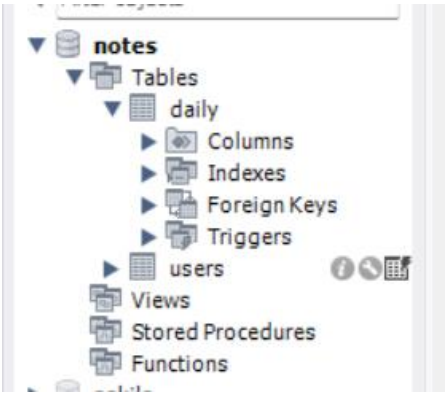

Рисунок 16 – Дерево базы данных

Добавляем данные в таблицы (рисунок 17–18).

Рисунок 17 – Добавление элементов в таблицу пользователей

Рисунок 18 – Добавление элементов в таблицу задач

2 Разработка программы

2.1 Алгоритм решения

- При запуске сайта пользователь попадает на страницу авторизации, которая выполнена на основе предыдущей практической работы
- Далее – попадает на страницу со своими делами. Они разделены на запланированные и выполненные
- Запланированные дела можно выполнить и удалить с помощью

кнопок

- Выполненные задания можно удалить
- Так же, на этой странице можно добавить дело, которое автоматически попадает в запланированные и выйти из аккаунта.

2.2 Программная реализация

Дерево проекта изображено на рисунке 19.

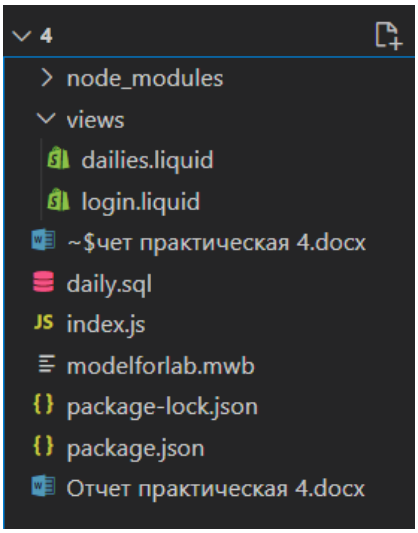

Рисунок 19 – Дерево проекта

2.2.1 Файл index.js

Sequelize — это ORM (Object-Relational Mapping — объектно-реляционное отображение или преобразование) для работы с такими СУБД (системами управления (реляционными) базами данных, Relational Database Management System, RDBMS), как Postgres, MySQL, MariaDB, SQLite и MSSQL.

Подключаем свою БД к проекту (рисунок 20).

```
const sequelize = new Sequelize("notes", "root", "Lokotok17.", {
  dialect: "mysql",
  host: "localhost"
});
```

Рисунок 20 – Подключение базы данных

Далее, определяем модели пользователя (рисунок 21) и дела (рисунок 22).

```
const User = sequelize.define('users', {
  id_users: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  login: {
    type: DataTypes.STRING,
    allowNull: false
  },
  password: {
    type: DataTypes.INTEGER,
    allowNull: false
  }},
  {
    timestamps: false // отключаем создание по
  }
);
```

Рисунок 21 – Модель пользователя

```
const Daily = sequelize.define('dailies', {
  id_daily: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  tasks: {
    type: DataTypes.STRING,
    allowNull: false
  },
  done: {
    type: DataTypes.BOOLEAN,
    allowNull: false,
    defaultValue: false
  }}, {
    timestamps: false // отключаем создание пол
  }
);
```

Рисунок 22 – Модель дела

Следующим шагом были определены связи и внешние ключи для них (рисунок 23).

```
User.hasMany(Daily,  { foreignKey: 'id_users' });
Daily.belongsTo(User,  { foreignKey: 'id_users' });
```

Рисунок 23 – Определение связей

Данные из базы данных извлекались таким образом, как показано на рисунке 24.

```
const user = await User.findOne({ where: { login: login, password: password } });
```

Рисунок 24 – Поиск необходимого пользователя

Для обработки дел была написана функция, которая изображена на рисунке 25. В ней идет проверка авторизации пользователя, поиска его по id, поиска дел у этого пользователя, их распределение на выполненные и запланированные и их выведение на страницу.

```
app.get('/dailies', requiresAuth, async (req, res) => {
  const userId = req.session.user.id_users;
  if (!userId) {
    return res.redirect('/login');
  }
  const user = await User.findOne({ where: { id_users: userId } });
  if (!user) {
    return res.redirect('/login');
  }
  const dailies = await Daily.findAll({ where: { id_users: user.id_users } });
  const plannedTasks = dailies.filter((daily) => !daily.done);
  const completedTasks = dailies.filter((daily) => daily.done);
  console.log(completedTasks);
  res.render('dailies', { plannedTasks: plannedTasks, completedTasks: completedTasks });
});
```

Рисунок 25 – Поиск и вывод дел для определенного пользователя

На рисунке 26 изображена функция, благодаря которой добавляется дело. Создается переменная на основе модели, в нее записывается задание и id пользователя и все сохраняется.

```
app.post('/add', async (req, res) => {
  const daily = new Daily();
  const  task  = req.body.tasks;
  const userId = req.session.user.id_users;
  daily.id_users = userId;
  daily.tasks = task;
  await daily.save();
  res.redirect('/dailies');
});
```

Рисунок 26 – Добавление дела

На рисунке 27 изображена функция смены выполненности задания. В нее передается id задания и далее – в его параметрах меняется параметр выполненности.

```
app.get('/complete/:id_daily', async (req, res) => {
 const itemId = req.params.id_daily;
  console.log(itemId);
  const daily = await Daily.findOne({ where: { id_daily: itemId } });
  if (!daily) {
    return res.status(404).send('Daily not found');
  }
  daily.done = true;
  await daily.save();
  res.redirect('/dailies');
});
```

Рисунок 27 – Определение дела, как выполненного

На рисунке 28 изображена функция удаления задания. В нее передается id задания, оно находится и удаляется.

```
app.get('/delete/:id_daily', async (req, res) => {
  const itemId = req.params.id_daily;
  const daily = await Daily.findOne({ where: { id_daily: itemId } });
  if (!daily) {
    return res.status(404).send('Daily not found');
  }
  await daily.destroy();
  res.redirect('/dailies');
});
```

Рисунок 28 – Удаление дела

2.2.2 Файл dailies.liquid

В данном файле производится вывод списка дел, передача id дела (рисунок 29).

```
{% for daily in plannedTasks %}
  <li>{{ daily.dataValues.tasks }}</li>
   <form action="/complete/{{daily.dataValues.id_daily}}">
    <input id ="itemId" type="hidden" name="itemId" value="{{daily.dataValues.id_daily}}">
    <button type="submit">complete</button>
   </form>
   <form action="/delete/{{daily.dataValues.id_daily}}">
    <input id ="itemId" type="hidden" name="itemId" value="{{daily.dataValues.id_daily}}">
    <button type="submit">delete</button>
   </form>
{% endfor %}
```

Рисунок 29 – Вывод дел на страницу

2.3 Результат работы программы

Страница авторизации выглядит так, как изображено на рисунке 30.

Рисунок 30 – Страница авторизации

После того, как пользователь ввел данные, он нажимает на кнопку «Login» и попадает на страницу со списком дел (рисунок 31).

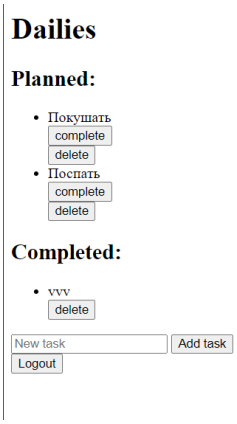

Рисунок 31 – Список дел

ЗАКЛЮЧЕНИЕ

В ходе выполнения практического задания была разобрана работа с базой данных mysql, Sequelize.

Так же, были развиты знания Java Script.

ПРИЛОЖЕНИЕ

Файл index.js

```
const { Sequelize, DataTypes } = require('sequelize');
const express = require('express');
const session = require('express-session');
const app = express();

var { Liquid } = require('liquidjs');
var engine = new Liquid();
const port = 3000;

const bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

app.engine('liquid', engine.express());
app.set('views', './views');
app.set('view engine', 'liquid');
//app.use(express.static('./')); //для применения css

const sequelize = new Sequelize("notes", "root", "Lokotok17.", {
  dialect: "mysql",
  host: "localhost"
});

const authenticate = async () => {
  try {
      await sequelize.authenticate()
      console.log('Соединение с БД было успешно установлено')
    } catch (err) {
      console.log('Невозможно выполнить подключение к БД: ', err)
    }
  };

authenticate();

const User = sequelize.define('users', {
  id_users: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  login: {
    type: DataTypes.STRING,
    allowNull: false
  },
  password: {
    type: DataTypes.INTEGER,
    allowNull: false
```

```
  }},
  {
    timestamps: false // отключаем создание полей createAt и updatedAt
  }
);

const Daily = sequelize.define('dailies', {
  id_daily: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  tasks: {
    type: DataTypes.STRING,
    allowNull: false
  },
  done: {
    type: DataTypes.BOOLEAN,
    allowNull: false,
    defaultValue: false
  }}, {
    timestamps: false // отключаем создание полей createAt и updatedAt
  }
);
ы
User.hasMany(Daily,  { foreignKey: 'id_users' });
Daily.belongsTo(User,  { foreignKey: 'id_users' });

sequelize.sync();


app.use(session({
    secret: 'secret-key',
    resave: false,
    saveUninitialized: true
  }));

  const requiresAuth = (req, res, next) => {
      if (req.session && req.session.user && req.session.user.isAuthenticated) {
          return next();
      } else {
          return res.redirect('/login');
      }
  };

  app.get('/login', (req, res) => {
      res.render('login');
  });

  app.post('/login', async (req, res) => {
```

```
      var login = req.body.username;
      var password = req.body.password;

     const user = await User.findOne({ where: { login: login, password: password
} });
    if (user) {
        req.session.user = { isAuthenticated: true, login: user.login, password:
user.password, id_users: user.id_users };
         return res.redirect('/dailies');
       } else {
        return res.redirect('/login');
       }
  });

  app.get('/dailies', requiresAuth, async (req, res) => {
    const userId = req.session.user.id_users;
    if (!userId) {
      return res.redirect('/login');
    }
    const user = await User.findOne({ where: { id_users: userId } });
    if (!user) {
      return res.redirect('/login');
    }
    const dailies = await Daily.findAll({ where: { id_users: user.id_users } });
    const plannedTasks = dailies.filter((daily) => !daily.done);
    const completedTasks = dailies.filter((daily) => daily.done);
    console.log(completedTasks);
    res.render('dailies', { plannedTasks: plannedTasks, completedTasks:
completedTasks });
  });

  app.get('/logout', (req, res) => {
    req.session.destroy();
    res.redirect('/login');
  });

  app.post('/add', async (req, res) => {
    const daily = new Daily();
    const  task  = req.body.tasks;
    const userId = req.session.user.id_users;
    daily.id_users = userId;
    daily.tasks = task;
    await daily.save();
    res.redirect('/dailies');
  });

  app.get('/complete/:id_daily', async (req, res) => {
   const itemId = req.params.id_daily;
    console.log(itemId);
    const daily = await Daily.findOne({ where: { id_daily: itemId } });
```

```
    if (!daily) {
      return res.status(404).send('Daily not found');
    }
    daily.done = true;
    await daily.save();
    res.redirect('/dailies');
  });

  app.get('/delete/:id_daily', async (req, res) => {
    const itemId = req.params.id_daily;
    const daily = await Daily.findOne({ where: { id_daily: itemId } });
    if (!daily) {
      return res.status(404).send('Daily not found');
    }
    await daily.destroy();
    res.redirect('/dailies');
  });

app.listen(port, () => console.log('started'));
```

Файл login.liquid

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <h1>Login</h1>
  <form action="/login" method="POST">
    Username: <input type="text" name="username"><br><br>
    Password: <input type="password" name="password"><br><br>
    <input type="submit" value="Login">
  </form>
  </body>
</html>
```

Файл dailies.liquid

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <title>Dailies</title>
  </head>
  <body>
    <h1>Dailies</h1>
```

```
    <h2>Planned:</h2>
    <ul>
      {% for daily in plannedTasks %}
        <li>{{ daily.dataValues.tasks }}</li>
         <form action="/complete/{{daily.dataValues.id_daily}}">
          <input id ="itemId" type="hidden" name="itemId"
value="{{daily.dataValues.id_daily}}">
          <button type="submit">complete</button>
         </form>
         <form action="/delete/{{daily.dataValues.id_daily}}">
          <input id ="itemId" type="hidden" name="itemId"
value="{{daily.dataValues.id_daily}}">
          <button type="submit">delete</button>
         </form>
      {% endfor %}
    </ul>

    <h2>Completed:</h2>
    <ul>
      {% for daily in completedTasks %}
        <li>{{ daily.dataValues.tasks }}</li>
        <form action="/delete/{{daily.dataValues.id_daily}}">
        <input id ="itemId" type="hidden" name="itemId"
value="{{daily.dataValues.id_daily}}">
          <button type="submit">delete</button>
         </form>
      {% endfor %}
    </ul>

    <form action="/add" method="post">
      <input type="text" name="tasks" placeholder="New task">
      <button type="submit">Add task</button>
    </form>
    <form  action="/logout">
    <button type="submit">Logout</button>
  </form>
  </body>
</html>
```