Lab Exercise: Building an Address Book Web App with Node.js, Express.js, Pug.js, and MySQL

Objective:

- Create a simple address book application using Node.js, Express.js, Pug.js, and MySQL.

Prerequisites:

- You may continue working on the same Express.js application created in the previous lab session.
- MySQL installed on your system (can use WAMP or XAMPP to manage the database using phpMyAdmin

1. Install Dependencies

In your project folder, run the following commands to install the necessary dependencies: mysql2 and body-parser.

```
npm install mysql2 body-parser
```

mysql2: This is a MySQL client for Node.js with a focus on performance. It supports prepared statements, non-utf8 encodings, binary log protocol, compression, SSL, and much more.

body-parser: This is a Node.js body parsing middleware. It parses incoming request bodies in a middleware before your handlers, available under the req.body property.

These packages are commonly used in web development with Node.js. The mysql package allows your application to interact with a MySQL database, while the body-parser package helps in parsing incoming HTTP request bodies which can be accessed via req.body.

2. Create the MySQL Database

Set up a MySQL database to store the address book entries. You can use a tool like phpMyAdmin or run SQL commands directly. Here's an example SQL schema:

```
CREATE DATABASE addressbook;
USE addressbook;
CREATE TABLE contacts (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255),
  phone VARCHAR(20)
);
```

3. In your current **app.js** file:

3.1 Import the body-parser module and assign it to the constant bodyParser

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

3.2 Import the mysql2 module and assigning it to the constant mysql.

```
const mysql = require('mysql2');
```

3.3 Configure MySQL connection

```
const db = mysql.createConnection({
  host: 'localhost',
  user: 'your_db_user',
  password: 'your_db_password',
  database: 'addressbook',
});

db.connect((err) => {
  if (err) {
    console.error('Database connection failed:', err);
  } else {
    console.log('Connected to the database');
  }
});
```

3.4 Route handler for the route '/addressbook'

Create a new file in the routes folder, call it addressbook.js.
In addressbook.js, you can define your route like this:

```
const express = require('express');
const router = express.Router();

module.exports = (db) => {
   router.get('/addressbook', (req, res) => {
      db.query('SELECT * FROM contacts', (err, results) => {
         if (err) throw err;
         res.render('addressbook', { contacts: results });
      });
   });

   return router;
};
```

`app.get('/addressbook', (req, res) => {...});` This is an Express.js route that handles HTTP GET requests to the /addressbook path. When a GET request is made to this path, the callback function provided as the second argument is invoked.

`db.query('SELECT * FROM contacts', (err, results) => {...});` This line is using a query method from a db object (which is likely a connection to a database) to execute a SQL query. The SQL query 'SELECT * FROM contacts' retrieves all records from the contacts table in the database.

`if (err) throw err;` This line checks if there was an error executing the query. If there was an error (err is not null), it throws the error, effectively stopping execution of the function.

`res.render('addressbook', { contacts: results });` This line uses the render method of the response (res) object to generate a view. It's rendering the view named 'addressbook' and passing in an object with a property contacts, which is set to the results of the database query. The 'addressbook' view will have access to contacts when it's being rendered.

3.5 Route handler for the route '/add':

```
const express = require('express');
const router = express.Router();

module.exports = (db) => {
 router.post('/add', (req, res) => {
  const { name, email, phone } = req.body;
  const query = 'INSERT INTO contacts (name, email, phone) VALUES (?, ?, ?)';
  db.query(query, [name, email, phone], (err, results) => {
   if (err) throw err;
   res.redirect('/addressbook');
  });
 });

 return router;
};
```

`app.post('/add', (req, res) => {...});` This is an Express.js route that handles HTTP POST requests to the /add path. When a POST request is made to this path, the callback function provided as the second argument is invoked.

`const { name, email, phone } = req.body;` This line is using destructuring assignment to extract name, email, and phone properties from req.body. The req.body object contains key-value pairs of data submitted in the request body. This data has been parsed by body-parser middleware.

`const query = 'INSERT INTO contacts (name, email, phone) VALUES (?, ?, ?)';` This line is defining a SQL query to insert a new record into the contacts table. The question marks (?) are placeholders for values that will be inserted in the next step.

`db.query(query, [name, email, phone], (err, results) => {...});` This line is using a query method from a db object (which is likely a connection to a database) to execute the SQL query. The second argument is an array of values that replace the placeholders in the query.

`if (err) throw err;` This line checks if there was an error executing the query. If there was an error (err is not null), it throws the error, effectively stopping execution of the function.

`res.redirect('/addressbook');` If there was no error, this line redirects the client to the /addressbook path.

3.6 In app.js, import and use this route:

```
const addressbookRoutes = require('./routes/addressbook')(db);
app.use('/', addressbookRoutes);

const addRoutes = require('./routes/add')(db);
app.use('/', addRoutes);
```

4. Create Pug Views
In views folder in your project directory and create a addressbook.pug file inside it. This file will display your address book entries. Here's a basic example:

```
extends layout

block content
  h1 Address Book

  form(action="/add", method="post")
    input(type="text", name="name", placeholder="Name" , required)
    input(type="email", name="email", placeholder="Email", required)
    input(type="text", name="phone", placeholder="Phone", required)
    button(type="submit") Add Contact

  each contact in contacts
    div(class="box")
      h4= contact.name
      a(href='mailto:' + contact.email) #{contact.email}
      p= contact.phone
```

5. Add this CSS class in the existing style.css file

```
.box {
 width: 250px;
 height: 150px;
 border-radius: 10px;
 border: 1px #000 solid;
 padding: 10px;
 margin: 10px;
 float: left;
}
```