Week 5 assignment

Database Schema

1. Create Users Table

```
CREATE TABLE IF NOT EXISTS Users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL
);
```

2. Create Expenses Table

```
CREATE TABLE IF NOT EXISTS Expenses (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  amount DECIMAL(10, 2) NOT NULL,
  date DATE NOT NULL,
  category VARCHAR(50) NOT NULL,
  FOREIGN KEY (user_id) REFERENCES Users(id)
);
```

Users	
id	INT, PK, AUTO_INCREMENT
username	VARCHAR(50), UNIQUE, NOT NULL
password	VARCHAR(255), NOT NULL

Expenses	
id	INT, PK, AUTO_INCREMENT
user_id	INT, FK
amount	DECIMAL(10, 2)
date	DATE
category	VARCHAR(50)

User Authentication

1. Implement User Registration and Login

Create routes/auth.js:

```
// routes/auth.js
const express = require('express');
const bcrypt = require('bcryptjs');
const connection = require('../db');
const router = express.Router();

router.post('/register', (req, res) => {
  const { username, password } = req.body;
  const hashedPassword = bcrypt.hashSync(password, 10);

  const query = 'INSERT INTO Users (username, password) VALUES (?, ?)';
  connection.query(query, [username, hashedPassword], (err, results) => {
    if (err) {
      return res.status(500).json({ message: 'Error registering user', err });
```

```
    }
    res.status(201).json({ message: 'User registered successfully' });
  });
});

router.post('/login', (req, res) => {
  const { username, password } = req.body;

  const query = 'SELECT * FROM Users WHERE username = ?';
  connection.query(query, [username], (err, results) => {
    if (err || results.length === 0) {
      return res.status(400).json({ message: 'Invalid username or password' });
    }

    const user = results[0];
    if (!bcrypt.compareSync(password, user.password)) {
      return res.status(400).json({ message: 'Invalid username or password' });
    }

    res.status(200).json({ message: 'User logged in successfully' });
  });
});

module.exports = router;
```

Expense Management

1. Add Expense

Create routes/expenses.js:

```
// routes/expenses.js
const express = require('express');
const connection = require('../db');
const router = express.Router();

router.post('/add', (req, res) => {
```

```
  const { user_id, amount, date, category } = req.body;

  const query = 'INSERT INTO Expenses (user_id, amount, date, category) VALUES
(?, ?, ?, ?)';
  connection.query(query, [user_id, amount, date, category], (err, results) => {
    if (err) {
      return res.status(500).json({ message: 'Error adding expense', err });
    }
    res.status(201).json({ message: 'Expense added successfully' });
  });
});

module.exports = router;
```

2. View Expenses

```
router.get('/view/:user_id', (req, res) => {
  const { user_id } = req.params;

  const query = 'SELECT * FROM Expenses WHERE user_id = ?';
  connection.query(query, [user_id], (err, results) => {
    if (err) {
      return res.status(500).json({ message: 'Error retrieving expenses', err });
    }
    res.status(200).json(results);
  });
});
```

User Authentication and Authorization

Use middleware to protect routes and ensure only authenticated users can add/view expenses.

App.js (Main Server File)

```
// app.js
const express = require('express');
const bodyParser = require('body-parser');
const authRoutes = require('./routes/auth');
const expenseRoutes = require('./routes/expenses');

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

app.use(bodyParser.json());
app.use('/auth', authRoutes);
app.use('/expenses', expenseRoutes);

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

Bonus Challenge: Database Server Comparison

Database Server	Type	Target Audience	Key Features	Ease of Use
MySQL	Open-Source	Small to Large Scale	ACID compliance, High Performance, Sc	High
PostgreSQL	Open-Source	Enterprises, Developers	Advanced SQL features, Extensibility, Data	Moderate
Microsoft SQL Server	Commercial	Enterprises	Integrated with Microsoft products, High Pe	High (for Windows)
Oracle Database	Commercial	Large Enterprises	Robust, Highly Scalable, Advanced Security	Moderate to High
Node.js	Open-Source	Developers, Startups	Non-blocking I/O, Event-driven archite	High

Server creation

Sever.js

```
const express = require('express');
const mysql = require('mysql');
const bodyParser = require('body-parser');
const bcrypt = require('bcryptjs');
const app = express();
const port = 3000;

app.use(bodyParser.json());

// MySQL connection
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '', // Add your MySQL root password here
  database: 'expense_tracker'
});

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

```
// User registration
app.post('/register', (req, res) => {
  const { username, password } = req.body;
  const hashedPassword = bcrypt.hashSync(password, 8);
  const sql = 'INSERT INTO users (username, password) VALUES (?, ?)';
  db.query(sql, [username, hashedPassword], (err, result) => {
    if (err) return res.status(500).send('Server error');
    res.status(201).send('User registered');
  });
});

// User login
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const sql = 'SELECT * FROM users WHERE username = ?';
  db.query(sql, [username], (err, results) => {
    if (err) return res.status(500).send('Server error');
    if (results.length === 0) return res.status(404).send('User not found');
    const user = results[0];
    const passwordIsValid = bcrypt.compareSync(password, user.password);
    if (!passwordIsValid) return res.status(401).send('Invalid password');
    res.status(200).send('Login successful');
```

```
  });
});

// Add expense
app.post('/expenses', (req, res) => {
  const { user_id, category, description, amount, date } = req.body;
  const sql = 'INSERT INTO expenses (user_id, category, description, amount, date)
VALUES (?, ?, ?, ?, ?)';
  db.query(sql, [user_id, category, description, amount, date], (err, result) => {
    if (err) return res.status(500).send('Server error');
    res.status(201).send('Expense added');
  });
});

// View expenses
app.get('/expenses/:user_id', (req, res) => {
  const { user_id } = req.params;
  const sql = 'SELECT * FROM expenses WHERE user_id = ?';
  db.query(sql, [user_id], (err, results) => {
    if (err) return res.status(500).send('Server error');
    res.status(200).json(results);
  });
});
```

```
// Update expense
app.put('/expenses/:id', (req, res) => {
  const { id } = req.params;
  const { category, description, amount, date } = req.body;
  const sql = 'UPDATE expenses SET category = ?, description = ?, amount = ?, date = ? WHERE id = ?';
  db.query(sql, [category, description, amount, date, id], (err, result) => {
    if (err) return res.status(500).send('Server error');
    res.status(200).send('Expense updated');
  });
});

// Delete expense
app.delete('/expenses/:id', (req, res) => {
  const { id } = req.params;
  const sql = 'DELETE FROM expenses WHERE id = ?';
  db.query(sql, [id], (err, result) => {
    if (err) return res.status(500).send('Server error');
    res.status(200).send('Expense deleted');
  });
});
```

```
app.listen(port, () => {
    console.log(`Server running on port ${port}`);
});
```