

Introduction

In the last Node/Express/EJS tutorial, you used dummy data from an array, to represent the meetings. This data is not persisted, if these were real meetings, you would expect them to come from a central source, like a database. In this tutorial, you will

- add support for a relational database, which is the one with tables! We will use SQLite and SQLite3, to build the code base we need for the database connection and query. Remember, a query is how we get data back from a database.
- apply more Bootstrap styling

1) Add dependencies to Support Connecting To a SQLite Database

SQLite and SQLite3 are 2 packages that work together to build the code base for creating and querying a relational database, using the SQLite framework. SQLite is a database engine built in C, but embedded in other languages. We will be installing these 2.

Package, body-parser, is a new addition.

➤ `npm install sqlite sqlite3 body-parser`

As the name indicates, body-parser is a utility package that helps in synthesizing data from objects, and vice versa. It is *middleware* in Node.js and is used to parse incoming request bodies making it easier to handle different types of data sent in HTTP requests. It's commonly used with the Express framework. Here are some key features of body-parser, that we will be using moving forwards with this sample data management project.

- **JSON Parsing:** Parses JSON data sent in the request body.
- **URL-encoded Parsing:** Parses URL-encoded data (like form submissions).
- **Raw Data Parsing:** Parses raw data sent in the request body.
- **Text Parsing:** Parses text data sent in the request body.

We may as well be aware of body-parser now!

2) Modify Styling To Support A List Of Items, Add Bootstrap Classes

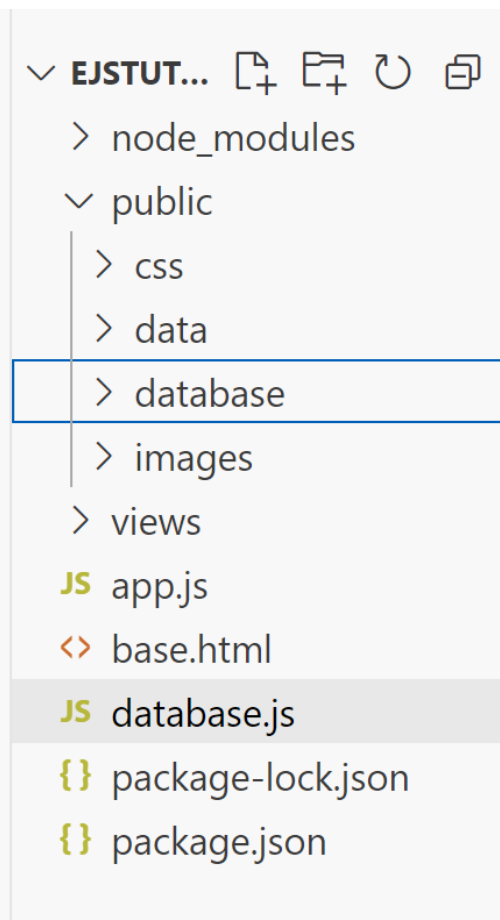
We will be displaying more meetings, so let's alter the container style in index.ejs, to wrap the content, and set some breakpoints.

```
<div class="d-flex flex-wrap flex-column flex-sm-column flex-md-row flex-lg-row">  
<% data.forEach((meeting)=>{ %>
```

3) Build The Database Support Infrastructure

Now let's build the database component to use SQLite.

Create a **file** and a **folder** as follows, **database.js** in the root folder, and **database**, a folder in the public folder, to store the database we will be creating.



4) Build the database call to create the database with the table, and connect the code

Let's take this step by step.

In app.js (or your server file, whatever you titled this)

<p>Import the SQLite/SQLite3 components needed to create and open the database</p>	<pre>import sqlite3 from 'sqlite3'; import { open } from 'sqlite';</pre>
<p>Add 2 functions, both of which need to be seen outside of the database.js file – as they are used to create the database in our public/database folder</p> <p>-setupDatabase -getDbConnection</p> <p>The export directive makes these visible outside the module.</p>	<pre>import sqlite3 from 'sqlite3'; import { open } from 'sqlite'; export const setupDatabase = () => { > return open({ ... > }).then(db => { ... > }); }; export const getDbConnection = () => { > return open({ ... > }); };</pre>
<p>Now add the connection/creation call, using a promise object</p>	

```

export const setupDatabase = () => {
  return open({
    filename: './public/database/meetings.db',
    driver: sqlite3.Database
  }).then(db => {
    return db.exec(`
      CREATE TABLE IF NOT EXISTS meetings (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        topic TEXT,
        mandatory BOOLEAN,
        dateTime TEXT,
        location TEXT,
        parking TEXT
      )
    `).then(() => {
      return db.run(`
        INSERT INTO meetings (topic, mandatory, dateTime, location, parking)
        VALUES
          ('CIT Monthly Meeting', 1, 'September 24th 2024, 1pm-5pm', 'KNOY Hall West L
          Street Garage, 3rd floor. Venue opposite front entrance.'),
          ('Research in Higher Level Ed', 0, 'October 5th 2024, 10am-12pm', 'Beresford
          Lafayette', 'Park in surface lot 300. Venue beside lot.'),
          ('Curriculum Planning', 1, 'October 19th 2024, 4pm-6pm', 'IO240, Indianapoli
          Garage, Michigan St. Venue opposite side of street, 300km North.')
      `);
    }).then(() => {
      console.log('Database setup complete.');
```

Here is the code for the insert, so you can avoid typing:

```

INSERT INTO meetings (topic, mandatory, dateTime, location, parking)
VALUES
  ('CIT Monthly Meeting', 1, 'September 24th 2024, 1pm-5pm', 'KNOY
Hall West Lafayette', 'Park in the West Street Garage, 3rd floor. Venue
opposite front entrance.'),
  ('Research in Higher Level Ed', 0, 'October 5th 2024, 10am-12pm',
'Beresford Building, Room 2, Hall West Lafayette', 'Park in surface lot
300. Venue beside lot.'),
  ('Curriculum Planning', 1, 'October 19th 2024, 4pm-6pm', 'IO240,
Indianapolis', 'Park in North Street Garage, Michigan St. Venue opposite
side of street, 300km North.')

```

<p>The call to open returns a promise object. You have seen that the then method handles a successful completion of a task (like opening the DB), and also that then methods can be chained to add additional consequences of a result.</p> <p>Here, if the DB can be opened successfully, then we create a table, then we add items to the table. We will be replicating items here, but no worries as this makes more meetings! This will be fixed shortly. It serves as a good example of how promises work.</p>	
<p>Now, if a database connection is required, the next method delivers, getDbConnection</p>	<pre>export const getDbConnection = () => { return open({ filename: './public/database/meetings.db', driver: sqlite3.Database }); };</pre>

- 5) Now we use the two methods we just defined in database.js. At the top of app.js, add the import for pulling in the functions.

```
// not doing this any more! import meetings from './public/data/data.js';
import { setupDatabase, getDbConnection } from './database.js';
```

- 6) Change the way we assimilate the meetings

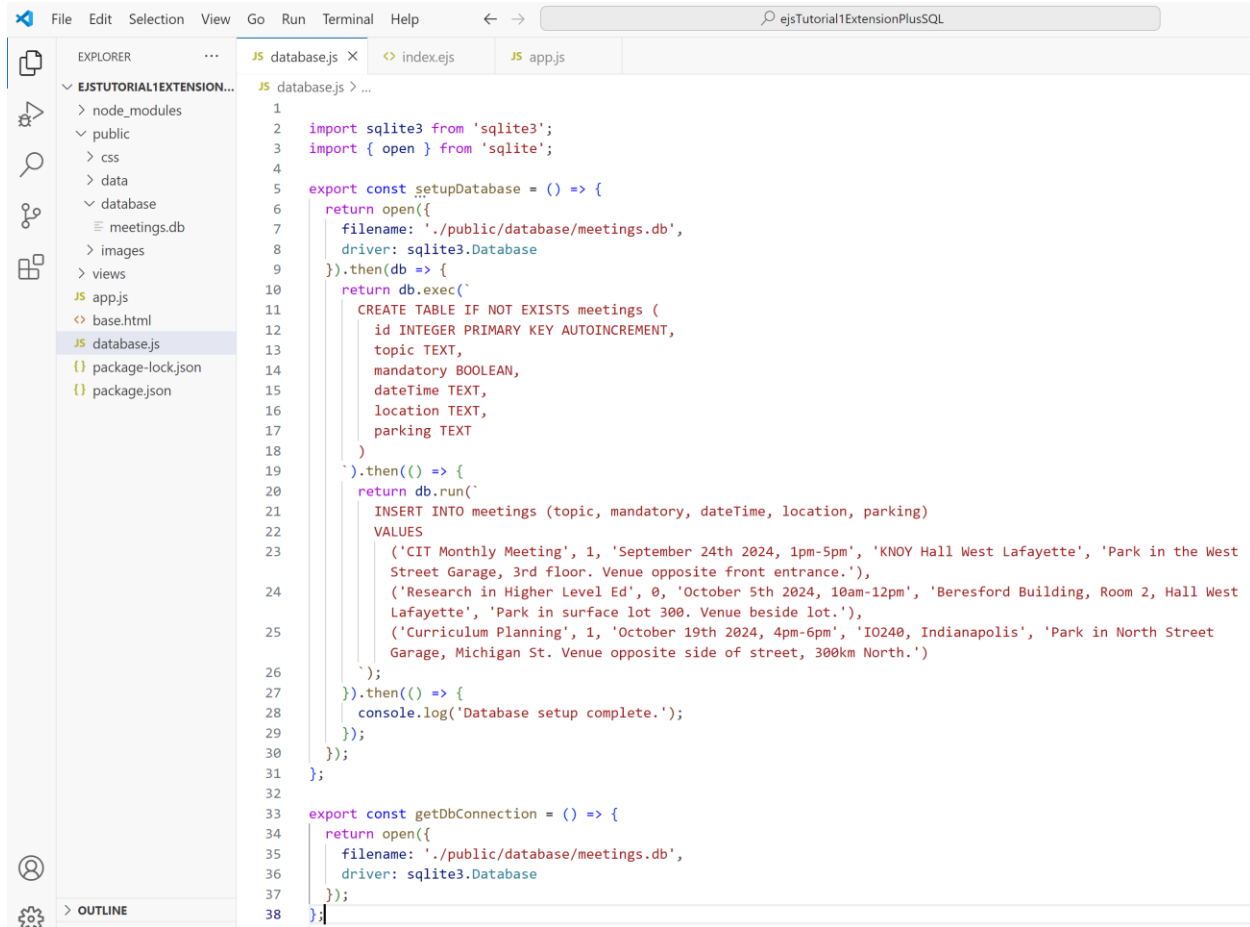
Now we read from the database, and use a promise object.

```
app.get("/", (req, res) => {  
  getDbConnection()  
    .then((db) => {  
      return db.all('SELECT * FROM meetings');  
    })  
    .then((meetings) => {  
      res.render("pages/index", {  
        data: meetings,  
        title: "Scheduled Meetings, Reminders",  
        clickHandler: clickHandler  
      });  
    })  
    .catch((error) => {  
      console.error(error);  
      res.status(500).send('Internal Server Error');  
    });  
});
```

The `getDbConnection` function returns a promise object. We chain a `then` and a `catch` to handle passing that data to `index.ejs`, and potentially handling an error condition.

Finally, now that we have plugged in the call to create and/or access the database, you should see the following.

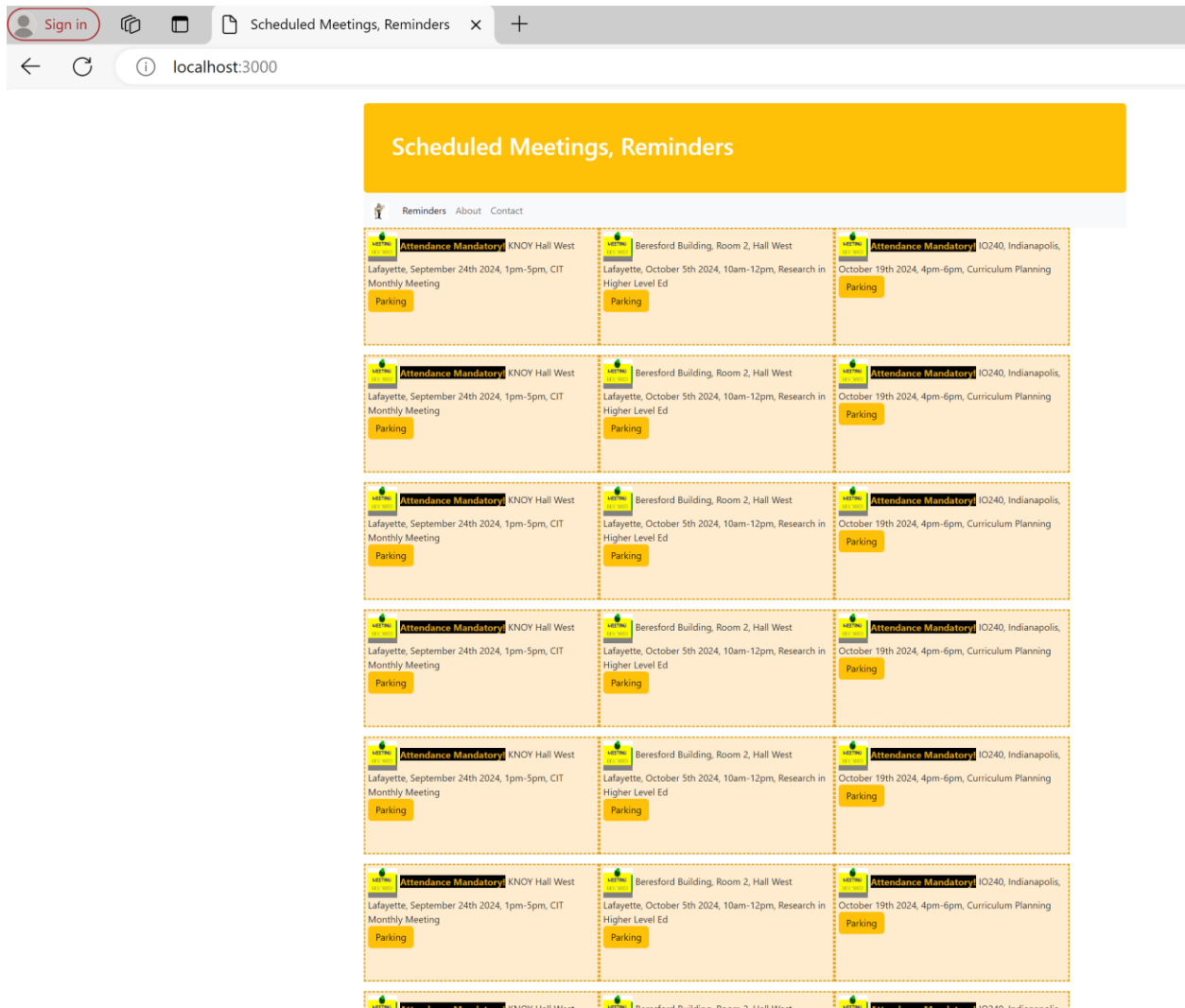
EJS Tutorial 2 --- Adding SQL Support



The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays the project structure for 'EJS TUTORIAL1 EXTENSION...'. The file 'database.js' is selected. The main editor area shows the code for 'database.js', which includes imports for 'sqlite3', a function 'setupDatabase' to create a table and insert data, and a function 'getDbConnection' to return the database connection.

```
1
2 import sqlite3 from 'sqlite3';
3 import { open } from 'sqlite';
4
5 export const setupDatabase = () => {
6   return open({
7     filename: './public/database/meetings.db',
8     driver: sqlite3.Database
9   }).then(db => {
10     return db.exec(`
11       CREATE TABLE IF NOT EXISTS meetings (
12         id INTEGER PRIMARY KEY AUTOINCREMENT,
13         topic TEXT,
14         mandatory BOOLEAN,
15         dateTime TEXT,
16         location TEXT,
17         parking TEXT
18       )
19     `).then(() => {
20       return db.run(`
21         INSERT INTO meetings (topic, mandatory, dateTime, location, parking)
22         VALUES
23         ('CIT Monthly Meeting', 1, 'September 24th 2024, 1pm-5pm', 'KNOY Hall West Lafayette', 'Park in the West
24           Street Garage, 3rd floor. Venue opposite front entrance.'),
25         ('Research in Higher Level Ed', 0, 'October 5th 2024, 10am-12pm', 'Beresford Building, Room 2, Hall West
26           Lafayette', 'Park in surface lot 300. Venue beside lot.'),
27         ('Curriculum Planning', 1, 'October 19th 2024, 4pm-6pm', 'IO240, Indianapolis', 'Park in North Street
28           Garage, Michigan St. Venue opposite side of street, 300km North.')
29       `);
30     });
31   });
32 }
33
34 export const getDbConnection = () => {
35   return open({
36     filename: './public/database/meetings.db',
37     driver: sqlite3.Database
38   });
39 }
```

EJS Tutorial 2 --- Adding SQL Support



Note all the repetition! This is happening because we have chained a *then*, to add the same items each time, to the database. You will be changing this.

Another thing to note for the next tutorial, promise objects and the ensuing *.then* cascades, can be difficult to read. ECMAScript 2017 added support for async functions, through an *await* call. We will be using these to 'tidy up' our database calls moving forwards – as we will need to get items, add items, update items and delete items, from our database table.