Front-End Web Developer Onboarding Handbook

Welcome to our web development team! This handbook will guide you through getting set up. The tech stack we are using for “Java Joes Coffee Shoppe” is Node.js, Express, and EJS. We will be using a SQLite relational database to store all data used for this website. We will be implementing a separation of concerns in order to keep everything organized. Below shows a diagram of how the directories and files will be structured as well as explanations of what each section entails and why. By using the following steps outlined below, you should have all you need to know in order to build “Java Joes Coffee Shoppe” and inventory management system.

1. Project Overview

While creating “Java Joes Coffee Shoppe”, you will be creating a website that displays a list of products stored in a relational database. The product inventory will be presented in a card-based layout. The following will explain the tech stack used, the database we have chosen to use, and the routes necessary to complete this project.

- **Stack**: Node.js, Express, EJS, Bootstrap
- **Database**: SQLite
- **Routes**:
    - `/products`: Displays all products.
    - `/about`: Displays the "About Us" page.
    - `/contact`: Displays the "Contact Us" page.

2. Project Structure

Here’s how the project structure will look:

- .
- ├── public
- │ ├── images
- │ ├── css
- │ └── database
- │ └── products.db
- ├── views
- │ ├── pages
- │ │ ├── about.ejs
- │ │ ├── contact.ejs
- │ │ ├── index.ejs
- │ │ └── products.ejs
- │ └── partials

- │ ├── footer.ejs
- │ ├── head.ejs
- │ └── menu.ejs
- ├── database.js
- ├── app.js
- ├── package.json

public: Static assets (images, stylesheets).
views: EJS templates for rendering the frontend UI.
routes: JavaScript files defining application routes.
database: The SQLite database for storing the product inventory.

3. Step-by-Step Guide

3.1. Install Dependencies

To get started, you'll need to install the following Node.js packages using your IDE terminal:

```
npm init -y
npm install express ejs sqlite3 sqlite
```

3.2. Set Up Express Application

Create the main entry point for the application in `app.js`:

```
import express from 'express';
import path from 'path';
import { fileURLToPath } from 'url';
import { getDbConnection, setupDatabse } from
'./database.js';

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

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

app.use(express.static(path.join(__dirname, 'public')));
app.set('view engine', 'ejs');

// Set up the database
```

```
setupDatabase().then(() => {
     console.log('Database setup complete);
});

// Use routes
app.use('/', (req, res) => {
     res.render('pages/index', {title: "Java Joe's Coffee
Shoppe" });
});

app.get('pages/about', (req, res) =>{
res.render('pages/about', { title: "Gather Round with a Cup
of Joe" });
});

app.get('/products', async(req, res) => {
try {
     const db = await getDbConnection();
     const products = await db.all('SELECT * FROM products
LIMIT 9');
     const productData = products.map(product => ({
     imgSrc: 'images',
     title:product.item,
     availability:product.availability,
     price: product.price
}));
res.render('pages/products', { data: productData, title:
"Settle Your Cravings" });
     } catch(error) {
console.error(error);
res.status(500).send('Internal Server Error');
     }
});


app.listen(port, () => {
    console.log(`App listening at port ${port}`);
});
```

3.3. Database Setup

Create a database in a `database.js` file to handle product inventory storage and retrieval:

```
import sqlite3 from 'sqlite3';
import { open } from 'sqlite';
import path from 'path';
import { fileURLToPath } from 'url';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const dbPath = path.resolve(__dirname, ‘./public/database/
products.db');

// Function to set up the database
export const setupDatabase = async () => {
    const db = await open({
        filename: dbPath,
        driver: sqlite3.Database
    });

    await db.exec(`
        CREATE TABLE IF NOT EXISTS products (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            item TEXT,
            availability BOOLEAN,
            price REAL,
            imgsrc TEXT
        )
    `);

    await db.run(`
        INSERT INTO products (item, availability, price,
imgsrc)
        VALUES
        ('Cappuccino', 1, 8.99, '/images/cappucino.jfif'),
        ('Espresso', 1, 7.99, '/images/espresso.jpg'),
        ('Coffee', 0, 4.99, '/images/coffee.jpg'),
        ('Bagel', 1, 4.49, '/images/bagel.jpg'),
        ('Muffin', 1, 6.99, '/images/muffin.jpg'),
```

```
        ('Bacon Egg Cheese Sandwich', 0, 8.99, '/images/
bacon_egg_cheese.jpg'),
        ('Sausage Egg Cheese Biscuit', 1, 8.99, '/images/
sausage_egg_cheese.jpg'),
        ('Hash Brown', 0, 1.99, '/images/hash_brown.jpg'),
        ('French Toast', 1, 7.99, '/images/
french_toast.jpg');
    `);
};
```

3.5. Create Views

In the `views` folder, create the following EJS files:

`products.ejs`:

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="https://
stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/
bootstrap.min.css">
</head>
<body>
    <div class="container">
        <h1>Our Products</h1>
        <div class="row">
            <% products.forEach(product => { %>
                <div class="col-md-4">
                    <div class="card">
                        <img src="<%= product.imgsrc %>"
class="card-img-top" alt="<%= product.item %>">
                        <div class="card-body">
                            <h5 class="card-title"><%=
product.item %></h5>
                            <button class="btn btn-primary"
onclick="alert('Price: $<%= product.price %>, Available:
<%= product.availability ? 'Yes' : 'No' %>')">View
Details</button>
                        </div>
```

```
                    </div>
                </div>
            <% }) %>
        </div>
    </div>
</body>
</html>
```

`about.ejs` and **`contact.ejs`** can be simple static pages describing the company and contact information.

4. Final Steps

Once everything is in place:

- **Run the server**: `node app.js`

- **Access the application** in your browser at `http://localhost:3000/`

5. Conclusion

This handbook provides a solid foundation for building a Node.js/Express/EJS-based prototype for managing a product inventory. As part of your onboarding, feel free to experiment with more advanced features such as product filtering, search functionality, or integrating with a cloud-based database.