Below is an **example project structure** and code for a simple Food Delivery System in Node.js using **Express.js**, **Promises**, **Async/Await**, and **EventEmitter**. This demo shows how to:

1. **Accept a new order** (using a callback for the final “accepted” step).
2. **Process the order asynchronously** (using a Promise).
3. **Emit an event** when the order is ready (using EventEmitter).
4. **Retrieve the order status** (using an async/await call).

Feel free to modify or extend this structure as needed.

## Project Structure

food-delivery/

├── package.json

├── README.md

└── src

├── app.js

├── server.js

├── controllers

│ └── orderController.js

├── events

│ └── orderEvents.js

├── models

│ └── orderModel.js

└── routes

└── orderRoutes.js

## 1. **package.json**

Create a package.json file to manage dependencies and scripts:

{

"name": "food-delivery-system",

"version": "1.0.0",

"description": "An Online Order Processing System using Node.js, Express, Promises, Async/Await, and EventEmitter",

"main": "src/server.js",

"scripts": {

"start": "node src/server.js"

},

"dependencies": {

"express": "^4.18.0"

}

}

Install the dependencies:

npm install

## 2. **src/app.js**

This file sets up the Express application and attaches our routes:

// src/app.js

const express = require('express');

const app = express();

const orderRoutes = require('./routes/orderRoutes');

// Middleware

app.use(express.json());

// Routes

app.use('/order', orderRoutes);

module.exports = app;

## 3. **src/server.js**

This file starts the server on a specified port:

// src/server.js

const app = require('./app');

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

## 4. **src/routes/orderRoutes.js**

Defines the endpoints for creating and retrieving orders:

// src/routes/orderRoutes.js

const express = require('express');

const router = express.Router();

const orderController = require('../controllers/orderController');

// 1. Accept a new order

router.post('/', orderController.acceptNewOrder);

// 2. Retrieve the order status

router.get('/:id', orderController.getOrderStatus);

module.exports = router;

## 5. **src/controllers/orderController.js**

Implements the logic for accepting and retrieving orders.

* Uses a **callback** for notifying when an order is “accepted.”
* Uses a **Promise** to process the order asynchronously.
* Uses **async/await** to fetch the order status.
* Emits an event when the order is ready.

// src/controllers/orderController.js

const Order = require('../models/orderModel');

const orderEvents = require('../events/orderEvents');

exports.acceptNewOrder = (req, res) => {

const orderData = req.body;

// Callback function to confirm the order was accepted

function orderAcceptedCallback(order) {

// Respond to the client once the order is accepted

res.status(201).json({

message: 'Order accepted successfully',

order: order

});

}

// 1. Accept the order (using callback)

Order.createOrder(orderData, orderAcceptedCallback);

// 2. Process the order asynchronously (using Promise)

Order.processOrder(orderData.id)

.then(() => {

// 3. Emit an event when order is processed/ready

orderEvents.emit('orderReady', orderData.id);

})

.catch((err) => {

console.error('Error processing order:', err);

});

};

exports.getOrderStatus = async (req, res) => {

try {

const orderId = parseInt(req.params.id, 10);

// 4. Retrieve order status using async/await

const status = await Order.getOrderStatus(orderId);

if (!status) {

return res.status(404).json({ message: 'Order not found' });

}

res.json({ orderId, status });

} catch (error) {

res.status(500).json({ message: 'Internal server error', error });

}

};

## 6. **src/models/orderModel.js**

Stores orders in memory (for demo purposes) and provides methods to create, process, and get status.

// src/models/orderModel.js

let orders = []; // In-memory data store

module.exports = {

createOrder(orderData, callback) {

// Simulate saving the order with status 'accepted'

const newOrder = {

id: orders.length + 1,

...orderData,

status: 'accepted'

};

orders.push(newOrder);

// Simulate asynchronous acceptance with setTimeout

setTimeout(() => {

// Once "accepted", call the callback

callback(newOrder);

}, 500);

},

processOrder(orderId) {

// Return a Promise to process the order

return new Promise((resolve, reject) => {

const order = orders.find(o => o.id === orderId);

if (!order) {

return reject('Order not found');

}

// Simulate async processing (e.g., cooking, packaging)

setTimeout(() => {

order.status = 'processed';

resolve();

}, 2000);

});

},

getOrderStatus(orderId) {

// Return a Promise to fetch order status asynchronously

return new Promise((resolve) => {

setTimeout(() => {

const order = orders.find(o => o.id === orderId);

if (!order) {

return resolve(null);

}

resolve(order.status);

}, 500);

});

}

};

## 7. **src/events/orderEvents.js**

Defines and exports an **EventEmitter** instance. We also set up a listener to log when an order is ready.

// src/events/orderEvents.js

const EventEmitter = require('events');

const eventEmitter = new EventEmitter();

// Listen for the 'orderReady' event

eventEmitter.on('orderReady', (orderId) => {

console.log(`Order with ID ${orderId} is ready!`);

});

module.exports = eventEmitter;

## 8. **README.md**

A simple README.md could look like this:

# Food Delivery System

A demo Node.js + Express app that accepts orders, processes them asynchronously, emits an event when ready, and retrieves the order status.

## Features

- \*\*POST /order\*\* - Accept a new order

- \*\*GET /order/:id\*\* - Retrieve order status

## Technologies

- Node.js

- Express.js

- Promises, Async/Await

- EventEmitter

## Getting Started

1. \*\*Install dependencies:\*\*

```bash

npm install

1. **Run the server:**
2. npm start
3. **Test Endpoints** (e.g., using curl or Postman):
   * **Create an Order:**
   * curl -X POST -H "Content-Type: application/json" \
   * -d '{"item":"Pizza","quantity":2}' \
   * http://localhost:3000/order
   * **Get Order Status:**
   * curl http://localhost:3000/order/1

## License

MIT

---

# How to Run

1. \*\*Download/clone\*\* this project.

2. Run `npm install` in the project folder to install dependencies.

3. Run `npm start` to start the server.

4. Use any HTTP client (like Postman or curl) to test:

- `POST /order` with a JSON body, for example:

```json

{

"item": "Burger",

"quantity": 2

}

```

- `GET /order/1` (or any other ID you created) to check the status.

This setup demonstrates the \*\*callback\*\* approach (for acceptance), \*\*Promise\*\* approach (for processing), \*\*EventEmitter\*\* (for readiness), and \*\*async/await\*\* (for retrieving status). Adjust as needed for a production-ready system (database integration, authentication, etc.).