**Task: Order Billing Form with Database Integration**

**Task Description**

You are required to develop a backend for an **Order Billing Form** that allows users to submit their billing details. You will integrate this form with two different databases: **MongoDB** and **SQL**. The backend will validate the input data, store it, and provide a confirmation response.

**Form Fields**

Here is a list of fields you need to handle from the form:

**Basic Information**

* Email (required)
* First Name (required)
* Last Name (required)
* Full Name (auto-generated by combining First Name and Last Name)
* Tracking Number (auto-generated unique value)

**Address Information**

* Address (required)
* Address 2 (optional)
* City (required)
* State (required)
* Zip Code (required)
* County (required)

**Steps to Complete the Task**

**1. Set Up the Project**

* Initialize a Node.js project with npm init.
* Install necessary dependencies:
* npm install express body-parser mongoose mysql dotenv

**2. Create the MongoDB Database**

* Set up a MongoDB collection called orders.
* Use **Mongoose** to define a schema for the form data:

const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({

email: { type: String, required: true },

firstName: { type: String, required: true },

lastName: { type: String, required: true },

fullName: { type: String },

status: { type: String, default: 'Pending' },

tracking: { type: String, unique: true },

address: { type: String, required: true },

address2: { type: String },

city: { type: String, required: true },

state: { type: String, required: true },

zip: { type: String, required: true },

county: { type: String, required: true }

});

module.exports = mongoose.model('Order', orderSchema);

**3. Create the SQL Database**

* Set up a MySQL table called orders.
* Define columns matching the form fields.
* Use the following SQL to create the table:

CREATE TABLE orders (

id INT AUTO\_INCREMENT PRIMARY KEY,

email VARCHAR(255) NOT NULL,

first\_name VARCHAR(255) NOT NULL,

last\_name VARCHAR(255) NOT NULL,

full\_name VARCHAR(510),

status VARCHAR(50) DEFAULT 'Pending',

tracking VARCHAR(255) UNIQUE,

address VARCHAR(255) NOT NULL,

address2 VARCHAR(255),

city VARCHAR(255) NOT NULL,

state VARCHAR(255) NOT NULL,

zip VARCHAR(50) NOT NULL,

county VARCHAR(255) NOT NULL

);

**4. Develop the Backend API**

* Create an **Express.js** server to handle form submissions.
* Add routes for:
  + Submitting form data (POST /submit):
    - Validate the input.
    - Save the data to **both MongoDB and SQL databases**.
    - Return a response confirming successful submission.

Example:

const express = require('express');

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

const mongoose = require('mongoose');

const mysql = require('mysql2');

const Order = require('./models/order'); // MongoDB model

const app = express();

app.use(bodyParser.json());

// MongoDB connection

mongoose.connect('mongodb://localhost:27017/orderBilling', { useNewUrlParser: true, useUnifiedTopology: true });

// MySQL connection

const sqlConnection = mysql.createConnection({

host: 'localhost',

user: 'root',

password: '',

database: 'orderBilling'

});

app.post('/submit', async (req, res) => {

const data = req.body;

// Auto-generate fullName and tracking number

data.fullName = `${data.firstName} ${data.lastName}`;

data.tracking = `TRACK-${Date.now()}`;

try {

// Save to MongoDB

const mongoOrder = new Order(data);

await mongoOrder.save();

// Save to MySQL

sqlConnection.query(

'INSERT INTO orders (email, first\_name, last\_name, full\_name, status, tracking, address, address2, city, state, zip, county) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',

[

data.email,

data.firstName,

data.lastName,

data.fullName,

data.status,

data.tracking,

data.address,

data.address2,

data.city,

data.state,

data.zip,

data.county

]

);

res.status(200).send({ message: 'Order submitted successfully!' });

} catch (error) {

res.status(500).send({ error: 'Error submitting the order.' });

}

});

app.listen(3000, () => console.log('Server running on port 3000'));

**5. Test the API (optional)**

* Use **Postman** or a similar tool to test the POST /submit endpoint.
* Confirm that data is correctly stored in both MongoDB and SQL.

**Bonus Challenge**

* Add an endpoint to retrieve all orders from both databases and compare the data.
* Implement additional validation for the form fields (e.g., email format, zip code length).
* Create a simple frontend (using HTML/CSS) to connect to your API.