Full Stack Feedback App (Node.js + Express + React + MongoDB)

UNIT-4

Backend — server.js — Part 1 of 3

```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");

const app = express();
app.use(bodyParser.json());

// 1) Connect to MongoDB (replace with your connection string)
mongoose.connect("mongodb://127.0.0.1:27017/userdb", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log("✅ MongoDB connected"))
.catch(err => console.error("❌ MongoDB connection error:", err));

// 2) Define schema (structure of data)
const userSchema = new mongoose.Schema({
  name: String,
```

Backend — server.js — Part 2 of 3

```
  email: String,
  feedback: String,
  createdAt: { type: Date, default: Date.now }
});

// 3) Create model
const User = mongoose.model("User", userSchema);

// 4) Route to gather data
app.post("/submit", async (req, res) => {
  try {
    const { name, email, feedback } = req.body;

    const newUser = new User({ name, email, feedback });
    await newUser.save();

    res.json({ message: " Data saved successfully!", user: newUser });
  } catch (err) {
```

Backend — server.js — Part 3 of 3

```
    console.error(err);
    res.status(500).json({ error: "Failed to save data" });
  }
});

// 5) Start server
const PORT = 3000;
app.listen(PORT, () => {
  console.log(` Server running on http://localhost:${PORT}`);
});
```

Backend Overview

This backend is built using Node.js, Express, and Mongoose (ODM for MongoDB).

It exposes a single POST endpoint (/submit) that accepts JSON payloads and writes them to the MongoDB 'userdb' database in the 'users' collection.

Frontend — App.js (React + Bootstrap) — Part 1 of 4

```
import React, { useState } from "react";
import 'bootstrap/dist/css/bootstrap.min.css';

function App() {
  const [formData, setFormData] = useState({ name: "", email: "", feedback: "" });
  const [message, setMessage] = useState("");

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await fetch("http://localhost:3000/submit", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(formData),
```

Frontend — App.js (React + Bootstrap) — Part 2 of 4

```
    });
    const result = await res.json();
    setMessage(result.message || "Submitted!");
    setFormData({ name: "", email: "", feedback: "" });
  } catch (err) {
    console.error(err);
    setMessage("Error submitting form.");
  }
};

return (
  <div className="d-flex justify-content-center align-items-center vh-100 bg-light">
    <div className="card shadow-lg p-4" style={{ width: "400px" }}>
      <h3 className="text-center mb-3">Feedback Form</h3>
      <form onSubmit={handleSubmit}>
        <div className="mb-3">
          <input type="text" className="form-control" name="name"
            placeholder="Your Name" value={formData.name}
```

Frontend — App.js (React + Bootstrap) — Part 3 of 4

```
            onChange={handleChange} required />
          </div>
          <div className="mb-3">
            <input type="email" className="form-control" name="email"
              placeholder="Your Email" value={formData.email}
              onChange={handleChange} required />
          </div>
          <div className="mb-3">
            <textarea className="form-control" name="feedback"
              placeholder="Your Feedback" rows="4"
              value={formData.feedback} onChange={handleChange} required />
          </div>
          <button type="submit" className="btn btn-success w-100">Submit</button>
        </form>
        {message && <div className="alert alert-info mt-3 text-center">{message}</div>}
      </div>
    </div>
  );
```

Frontend — App.js (React + Bootstrap) — Part 4 of 4

```
}
export default App;
```

package.json (important scripts)

```
{
  "name": "feedback-app",
  "version": "1.0.0",
  "scripts": {
    "start": "node server.js",
    "client": "react-scripts start",
    "dev": "concurrently "nodemon server.js" "npm run client""
  },
  "dependencies": {
    "express": "^4.x",
    "mongoose": "^6.x",
    "body-parser": "^1.x",
    "react": "^18.x",
    "bootstrap": "^5.x"
  }
}
```

How to run (local development)

1) Start MongoDB (local or use Atlas)
   - Local (e.g. mac/linux): mongod --dbpath /path/to/db
   - Atlas: get connection URI and set as MONGODB_URI

2) Backend:
   - npm install
   - node server.js (or use nodemon)

3) Frontend (in feedback-app/):
   - npm install
   - npm start

4) Open the React app in browser (usually http://localhost:3000 or 3001 depending on setup).
5) Submit form — data will be saved into MongoDB 'userdb' -> 'users' collection.

Student Registration System using MERN

Overview

- A simple student registration app:
- - Frontend: React (marks-client)
- - Backend: Node.js + Express + MongoDB (marks-backend)
- - Features: save student info (name, roll, gender, department, section, skills)
- - Exports: Excel (previous project) and scalable structure

Backend - Key Points

- server.js: Express server, Mongoose schema, routes for saving and listing students.
- Uses environment variable MONGO_URI for Atlas connection.
- Save endpoint: POST /students | List endpoint: GET /students
- Ensure CORS enabled for frontend communication.

Backend: server.js (part 1 of 3)

```
// server.js - Node.js + Express + MongoDB
require('dotenv').config();
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');

const app = express();
app.use(cors());
app.use(express.json());

// MongoDB connection
const MONGODB_URI = process.env.MONGO_URI || 'mongodb://127.0.0.1:27017/studentsdb';
mongoose.connect(MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log(' MongoDB connected'))
  .catch(err => console.error(' DB error:', err));

// Schema
const studentSchema = new mongoose.Schema({
  name: { type: String, required: true },
  rollNo: { type: String, required: true },
  gender: { type: String, enum: ['Male','Female'], required: true },
  department: { type: String, enum: ['IT','CSE','AIDS','CET'], required: true },
  section: { type: Number, enum: [1,2,3], required: true },
  skills: [{ type: String }],
  createdAt: { type: Date, default: Date.now }
});

const Student = mongoose.model('Student', studentSchema);
```

Defines server, DB connection, and Student schema.

Backend: server.js (part 2 of 3)

```
// Routes: save and list
// Save student
app.post('/students', async (req, res) => {
  try {
    const student = new Student(req.body);
    await student.save();
    res.json({ message: ' Student saved', student });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: ' Save failed' });
  }
});

// List students
app.get('/students', async (req, res) => {
  try {
    const list = await Student.find().sort({ createdAt: -1 });
    res.json(list);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: ' Fetch failed' });
  }
});
```

Routes for saving and listing students.

Backend: server.js (part 3 of 3)

```
// Start server
const PORT = process.env.PORT || 4000;
app.listen(PORT, () => console.log(` Server running on http://localhost:${PORT}`));
```

Start command and environment variable usage.

Frontend - Key Points

- React app (marks-client) with a simple form to collect student info.
- Uses fetch to POST to backend /students endpoint.
- Skills are handled as checkbox multi-select.
- Using environment variable REACT_APP_API_URL for backend URL.

Frontend: App.js (part 1 of 4)

```
// App.js - React Form
import React, { useState } from 'react';
import './App.css';

function App() {
  const [form, setForm] = useState({
    name: '', rollNo: '', gender: '', department: '', section: '', skills: []
  });
  const [message, setMessage] = useState(null);
  const API = process.env.REACT_APP_API_URL || 'http://localhost:4000';
```

State setup and API base URL (uses REACT_APP_API_URL).

Frontend: App.js (part 2 of 4)

```
// Handlers for inputs and skills
const handleChange = (e) => {
  setForm({ ...form, [e.target.name]: e.target.value });
};

const handleSkills = (e) => {
  const { value, checked } = e.target;
  setForm(prev => ({
    ...prev,
    skills: checked ? [...prev.skills, value] : prev.skills.filter(s => s !== value)
  }));
};
```

Input handlers and checkbox logic for skills.

Frontend: App.js (part 3 of 4)

```
// Submit form
const handleSubmit = async (e) => {
  e.preventDefault();
  setMessage(null);
  try {
    const res = await fetch(`${API}/students`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(form)
    });
    const data = await res.json();
    if (!res.ok) throw new Error(data.error || 'Save failed');
    setMessage(' Student saved');
    setForm({ name:'', rollNo:'', gender:'', department:'', section:'', skills:[] });
  } catch (err) {
    setMessage(' ' + err.message);
  }
};
```

Form submit logic with POST request to backend.

Frontend: App.js (part 4 of 4)

```
// JSX (form)
return (
  <div className="form-container">
    <h2>Student Registration</h2>
    {message && <p className="msg">{message}</p>}
    <form onSubmit={handleSubmit}>
      <label>Name</label>
      <input name="name" value={form.name} onChange={handleChange} required />

      <label>Roll No</label>
      <input name="rollNo" value={form.rollNo} onChange={handleChange} required />

      <label>Gender</label>
      <div className="radio-group">
        <label><input type="radio" name="gender" value="Male" checked={form.gender==='Male'} onChange={handleChange} />
Male</label>
        <label><input type="radio" name="gender" value="Female" checked={form.gender==='Female'} onChange={handleChange}
/> Female</label>
      </div>
```

JSX form for student data input.

```
 <label>Department</label>
      <select name="department" value={form.department} onChange={handleChange} required>
        <option
value="">--Select--
</option><option>IT</option><option>CSE</option><option>AIDS</option><option>CET</option>
      </select>

      <label>Section</label>
      <select name="section" value={form.section} onChange={handleChange} required>
        <option value="">--Select--</option><option value="1">1</option><option
value="2">2</option><option
value="3">3</option>
      </select>

      <label>Skills</label>
      <div className="checkbox-group">
        {['C','C++','Java','JS','Ruby'].map(s => (
          <label key={s}><input type="checkbox" value={s} checked={form.skills.includes(s)}
onChange={handleSkills} />
{s}</label>
        ))}
      </div>

      <button type="submit">Save</button>
    </form>
  </div>
);
```

Styling: App.css

```
/* App.css */
body { background: #f4f7fc; font-family: Arial, sans-serif; }
.form-container { max-width: 500px; background: #fff; margin: 40px auto; padding: 20px 30px; border-radius: 12px; box-
shadow: 0 4px 10px rgba(0,0,0,0.1); }
label { display:block; margin-top:10px; font-weight:600; color:#333; }
input[type='text'], select { width:100%; padding:8px; margin-top:5px; border-radius:6px; border:1px solid #ccc; }
.radio-group, .checkbox-group { display:flex; gap:12px; margin-top:8px; }
button { width:100%; padding:10px; background:#007bff; color:#fff; border:none; border-radius:8px; margin-top:20px;
cursor:pointer; }
.msg { text-align:center; font-weight:600; margin-bottom:12px; }
```

Simple clean styling for the form.

Significant Code Parts

- 1) MongoDB connection: uses MONGO_URI from environment for security.
- 2) Student schema: enforces required fields and enums to keep data clean.
- 3) POST /students: validates and saves incoming JSON payloads.
- 4) Frontend uses controlled components and checkbox logic for skills.
- 5) REACT_APP_API_URL: set this in Render or local env to point frontend to backend.

Deploying Frontend on Render (Static Site)

- 1. Create new Static Site and connect GitHub repo.
- 2. Root Directory: marks-client
- 3. Build Command: npm install && npm run build
- 4. Publish Directory: build
- 5. Add Environment Variable: REACT_APP_API_URL = https://your-backend-url
- 6. Create and deploy.

Deploying Backend on Render (Web Service)

- 1. Create new Web Service and connect GitHub repo.
- 2. Root Directory: marks-backend
- 3. Build Command: npm install
- 4. Start Command: npm start
- 5. Add environment variables: MONGO_URI, PORT
- 6. Deploy and link frontend API URL.

Next Steps / Enhancements

- - Add authentication (JWT) and per-teacher data separation.
- - Add subjects collection and subject dropdown in frontend.
- - Add Excel multi-sheet export per subject/exam (as previously discussed).
- - Add student list view and edit/delete functions.