ChatGPT

Shall's Construction Smart Tools Suite Codebase

Below is the complete monorepo codebase for the **Shall's Construction Smart Tools Suite**, including the **Project Bid Estimator** and **Permit & Inspection Scheduler** tools. The structure of the repository is as follows:

- **Root Directory (`shalls-tools/`)**
- `README.md` – Project overview and instructions.
- `package.json` – Project dependencies and scripts.
- **Backend (`shalls-tools/backend/`)**
- `.env` – Environment variables for API keys and credentials.
- `app.js` – Express server setup.
- `uploads/` – Directory to store uploaded files (e.g., PDFs).
- **Routes (`routes/`)**
  - `estimate.js` – API endpoint for bid estimation.
  - `scheduler.js` – API endpoint for permit/inspection scheduling.
- **Utilities (`utils/`)**
  - `email.js` – Stubbed email sending utility.
  - `costLogic.json` – Data for bid estimation logic.
  - `scheduleTemplate.json` – Template data for scheduling.
- **Frontend (`shalls-tools/frontend/`)**
- **Public (`public/`)**
  - `index.html` – Main HTML file for the React app.
- `main.jsx` – Entry point for the Vite + React app.
- **Pages (`pages/`)**
  - `Dashboard.jsx` – Main dashboard with links to tools.
  - `EstimatorForm.jsx` – Form page for project bid estimation.
  - `SchedulerForm.jsx` – Form page for permit/inspection scheduling.
- **Components (`components/`)**
  - `FormInput.jsx` – Reusable form input component (text/select).
  - `PDFViewer.jsx` – Component to preview uploaded PDF files.

Each file below includes inline comments explaining its purpose and logic.

README.md

```
# Shall's Construction Smart Tools Suite

This repository contains the source code for *Shall's Construction Smart Tools
Suite*, which includes:
```

```
- **Project Bid Estimator**: Estimate project costs based on project type, area, and material quality.
- **Permit & Inspection Scheduler**: Schedule permits and inspections with date, time, and location validation.

## Project Structure
```

shalls-tools/ ├── backend/ │ ├── routes/ │ │ ├── estimate.js │ │ ├── scheduler.js │ ├── utils/ │ │ ├── email.js │ │ ├── costLogic.json │ │ ├── scheduleTemplate.json │ ├── uploads/ │ ├── app.js │ └── .env ├── frontend/ │ ├── pages/ │ │ ├── EstimatorForm.jsx │ │ ├── SchedulerForm.jsx │ │ ├── Dashboard.jsx │ ├── components/ │ │ ├── FormInput.jsx │ │ ├── PDFViewer.jsx │ └── main.jsx ├── public/ │ └── index.html ├── README.md └── package.json

````
## Installation

1. **Clone the repository**:
   ```bash
   git clone https://github.com/yourusername/shalls-tools.git
   cd shalls-tools
   ```

2. **Install dependencies** (in the root directory, which contains `package.json`):
   ```bash
   npm install
   ```

3. **Environment Variables**:
   Create a `.env` file in `backend/` with your configuration. See `backend/.env.example` for reference.
   ```
   PORT=3000
   EMAIL_API_KEY=your_email_api_key
   TWILIO_SID=your_twilio_sid
   TWILIO_AUTH_TOKEN=your_twilio_auth_token
   GOOGLE_MAPS_API_KEY=your_google_maps_api_key
   ```

4. **Run the project**:
   - **Backend**: Start the Express server.
     ```bash
     npm run backend
     ```
   - **Frontend**: Start the Vite development server.
     ```bash
     npm run frontend
     ```
````

```
   (You can also run both concurrently with `npm run dev` if `concurrently` is
set up.)

5. **Access the application**:
   Open your browser and go to `http://localhost:3000` (backend serves APIs) and
`http://localhost:5173` (Vite dev server) as needed.

## Features

- **Responsive UI** with Tailwind CSS (blue/gray palette).
- **Form Inputs** with dropdowns, date/time pickers, and conditional logic.
- **PDF Upload and Preview** for blueprints or documents.
- **Express API** endpoints with LowDB (JSON file) for persistence.
- **Stubbed Integrations**:
  - Email (Nodemailer placeholder).
  - SMS (Twilio placeholder).
  - PDF generation (placeholder).
  - Google Maps ZIP code validation (stubbed).
  - Zapier webhook for leads (stubbed).
```

---

package.json

```
{
  "name": "shalls-tools",
  "version": "1.0.0",
  "description":
"Smart Tools for construction: Project Bid Estimator and Permit Scheduler",
  "scripts": {
    "start": "node backend/app.js",
    "dev": "concurrently \"npm run backend\" \"npm run frontend\"",
    "backend": "nodemon backend/app.js",
    "frontend": "vite"
  },
  "dependencies": {
    "express": "^4.18.2",
    "lowdb": "^3.0.0",
    "nodemailer": "^6.9.1",
    "twilio": "^3.80.0",
    "cors": "^2.8.5",
    "react": "^17.0.0",
    "react-dom": "^17.0.0",
    "react-router-dom": "^5.2.0"
  },
  "devDependencies": {
```

```
    "concurrently": "^7.2.2",
    "vite": "^3.0.0",
    "tailwindcss": "^3.0.0",
    "postcss": "^8.0.0",
    "autoprefixer": "^10.0.0"
  }
}
```

- **Scripts**:
- `npm run backend` – Starts the Express backend with `nodemon`.
- `npm run frontend` – Starts the Vite development server for React.
- `npm run dev` – Runs both backend and frontend concurrently.
- `npm start` – Runs the backend (useful for production).

Backend Code

backend/.env

```
# Server port
PORT=3000

# Email (Nodemailer) API key or credentials
EMAIL_API_KEY=your_email_api_key

# Twilio SMS credentials
TWILIO_SID=your_twilio_sid
TWILIO_AUTH_TOKEN=your_twilio_auth_token

# Google Maps API Key for ZIP code validation (stubbed)
GOOGLE_MAPS_API_KEY=your_google_maps_api_key
```

Note: Replace placeholder values with your actual API keys and credentials.

backend/app.js

```
// Backend server entry point (Node.js + Express)

const express = require('express');
const app = express();
const cors = require('cors');
const path = require('path');
require('dotenv').config(); // Load environment variables

// Import route handlers
```

```
const estimateRoute = require('./routes/estimate');
const schedulerRoute = require('./routes/scheduler');

// Enable CORS for requests from frontend
app.use(cors());

// Parse JSON request bodies
app.use(express.json());

// Serve static files from the 'uploads' directory (for file previews, etc.)
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));

// Mount API routes
app.use('/api/estimate', estimateRoute);
app.use('/api/scheduler', schedulerRoute);

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Shall's tools backend is running on port ${PORT}`);
});
```

- The server listens on the port defined in `.env` (default 3000).
- Requests to `/api/estimate` and `/api/scheduler` are handled by their respective route modules.
- The `uploads` folder is exposed as a static directory.

backend/routes/estimate.js

```
// Route for handling project bid estimation requests

const express = require('express');
const router = express.Router();
const path = require('path');
const { Low, JSONFile } = require('lowdb');

// Load cost logic data for estimates
const costLogic = require('../utils/costLogic.json');

// Initialize LowDB (stores data in backend/db.json)
const dbFile = path.join(__dirname, '../db.json');
const adapter = new JSONFile(dbFile);
const db = new Low(adapter);

// Initialize database with default structure
async function initDB() {
    await db.read();
```

```
    db.data = db.data || { estimates: [] };
    await db.write();
}
initDB();

// POST /api/estimate
router.post('/', async (req, res) => {
    // Extract form data from request body
    const { projectType, area, materialQuality } = req.body;

    // Basic estimation calculation using costLogic
    // (In a real app, this logic could be more complex)
    const baseRate = costLogic.baseRates[projectType] || costLogic.defaultRate;
    const materialMultiplier = costLogic.materials[materialQuality] || 1;
    const estimatedCost = area * baseRate * materialMultiplier;

    // Save the estimate in the database (with timestamp)
    db.data.estimates.push({
        projectType,
        area,
        materialQuality,
        estimatedCost,
        timestamp: Date.now()
    });
    await db.write();

    // TODO: Integrate email/SMS notifications (stubbed)
    // e.g., use email.sendEmail(...) or Twilio API here.

    // TODO: Zapier webhook stub for leads
    // Example (stubbed): send estimate data to Zapier webhook via axios/fetch.

    // Respond with the calculated estimate
    res.json({ estimatedCost, message: "Estimation complete." });
});

module.exports = router;
```

- **Endpoint**: `POST /api/estimate`
- **Request Body**: `{ projectType, area, materialQuality }`
- **Response**: JSON with the calculated `estimatedCost` and a message.
- The route stores each estimation in `db.json` under `estimates`.

backend/routes/scheduler.js

```
// Route for handling permit/inspection scheduling requests
```

```
const express = require('express');
const router = express.Router();
const path = require('path');
const { Low, JSONFile } = require('lowdb');

// Load scheduling template (not used directly in logic, placeholder data)
const scheduleTemplate = require('../utils/scheduleTemplate.json');

// Initialize LowDB (uses same db.json as estimates for simplicity)
const dbFile = path.join(__dirname, '../db.json');
const adapter = new JSONFile(dbFile);
const db = new Low(adapter);

// Initialize database structure for schedules
async function initDB() {
    await db.read();
    db.data = db.data || { schedules: [] };
    await db.write();
}
initDB();

// Stubbed function to validate ZIP code using Google Maps API
function validateZip(zip) {
    // Placeholder: In a real implementation, call Google Maps Geocoding API
    console.log(`Validating zip code ${zip} (stubbed)`);
    // For now, assume all zip codes are valid
    return true;
}

// POST /api/scheduler
router.post('/', async (req, res) => {
    const { name, date, time, zipCode } = req.body;

    // Validate ZIP code
    if (!validateZip(zipCode)) {
        return res.status(400).json({ error: "Invalid zip code." });
    }

    // Create a schedule entry
    const scheduleEntry = {
        name,
        date,
        time,
        zipCode,
        status: 'Scheduled',
        createdAt: new Date().toISOString()
    };
```

```
    // Save the schedule entry in the database
    db.data.schedules.push(scheduleEntry);
    await db.write();

    // TODO: Send notification (email/SMS) to the user (stubbed)
    // e.g., email.sendEmail(...), Twilio API, etc.

    res.json({
        message: "Inspection scheduled successfully.",
        schedule: scheduleEntry
    });
});

module.exports = router;
```

- **Endpoint**: `POST /api/scheduler`
- **Request Body**: `{ name, date, time, zipCode }`
- **Response**: JSON with a success message and the saved schedule entry.
- ZIP code validation is stubbed for demonstration.

backend/utils/email.js

```
/**
 * Stubbed email utility (placeholder for nodemailer).
 * In a production app, configure nodemailer with real SMTP/API credentials.
 */
module.exports.sendEmail = async function(to, subject, text) {
    console.log(`Sending email to ${to}: [${subject}] ${text}`);
    // Example of real usage:
    // const transporter = nodemailer.createTransport({ /* SMTP config */ });
    // await transporter.sendMail({ to, subject, text });
};
```

- This module exports a stubbed `sendEmail` function that logs the email instead of sending.

backend/utils/costLogic.json

```
{
  "baseRates": {
    "residential": 100,
    "commercial": 150,
    "industrial": 200
  },
  "materials": {
    "basic": 1,
    "standard": 1.25,
```

```
    "premium": 1.5
  },
  "defaultRate": 120
}
```

- Contains base rates and material multipliers for the estimator logic.

backend/utils/scheduleTemplate.json

```
{
  "inspectionTypes": ["Electrical", "Plumbing", "Structural", "General"],
  "defaultSlots": ["09:00", "11:00", "13:00", "15:00"]
}
```

- Template data (for reference) for scheduling. Not directly used in this stubbed example.

Frontend Code

public/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Shall's Tools Suite</title>
  <!-- Tailwind CSS (add your generated CSS if applicable) -->
</head>
<body class="bg-gray-100 font-sans">
  <div id="root"></div>
  <!-- The React app is mounted into #root by main.jsx -->
  <script type="module" src="/frontend/main.jsx"></script>
</body>
</html>
```

- Includes a `<div id="root">` where the React app renders.
- Uses Tailwind CSS classes for basic styling.

frontend/main.jsx

```
// Entry point for the React application (Vite)

import React from 'react';
```

```
import ReactDOM from 'react-dom';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

// Import pages
import Dashboard from './pages/Dashboard';
import EstimatorForm from './pages/EstimatorForm';
import SchedulerForm from './pages/SchedulerForm';

// Import global CSS (Tailwind base) - ensure this file includes Tailwind
directives if set up
import './index.css';

// Main App component with routing
function App() {
  return (
    <Router>
      <div className="min-h-screen bg-gray-100 p-4">
        <nav className="bg-blue-500 text-white p-4 rounded mb-4">
          <h1 className="text-2xl">Shall's Construction Tools</h1>
        </nav>
        <Switch>
          <Route exact path="/" component={Dashboard} />
          <Route path="/estimate" component={EstimatorForm} />
          <Route path="/schedule" component={SchedulerForm} />
        </Switch>
      </div>
    </Router>
  );
}

// Mount the App into the DOM
ReactDOM.render(<App />, document.getElementById('root'));
```

- Uses React Router for navigation between pages.
- A simple navigation header is included.
- Tailwind classes apply a clean blue/gray palette.

frontend/pages/Dashboard.jsx

```
// Dashboard page with links to each tool

import React from 'react';
import { Link } from 'react-router-dom';

export default function Dashboard() {
  return (
    <div className="p-4">
```

```
      <h2 className="text-xl font-semibold mb-4">Dashboard</h2>
      <ul className="space-y-2">
        <li>
          <Link to="/estimate" className="text-blue-600 hover:underline">
            Project Bid Estimator
          </Link>
        </li>
        <li>
          <Link to="/schedule" className="text-blue-600 hover:underline">
            Permit & Inspection Scheduler
          </Link>
        </li>
      </ul>
    </div>
  );
}
```

- Provides navigation links to the estimator and scheduler tools.

frontend/pages/EstimatorForm.jsx

```
// Project Bid Estimator form component

import React, { useState } from 'react';
import FormInput from '../components/FormInput';
import PDFViewer from '../components/PDFViewer';

export default function EstimatorForm() {
  // Form state variables
  const [projectType, setProjectType] = useState('residential');
  const [area, setArea] = useState('');
  const [materialQuality, setMaterialQuality] = useState('basic');
  const [numFloors, setNumFloors] = useState('');
  const [uploadedPdf, setUploadedPdf] = useState(null);
  const [estimate, setEstimate] = useState(null);

  // Handle form submission
  const handleSubmit = async (e) => {
    e.preventDefault();
    const data = { projectType, area, materialQuality, numFloors };
    try {
      const res = await fetch('http://localhost:3000/api/estimate', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
      });
      const result = await res.json();
```

```
      setEstimate(result.estimatedCost);
    } catch (error) {
      console.error('Error:', error);
    }
  };

  // Handle PDF file selection
  const handleFileChange = (e) => {
    const file = e.target.files[0];
    setUploadedPdf(file);
  };

  return (
    <div className="max-w-md mx-auto bg-white p-4 rounded shadow">
      <h2 className="text-xl font-semibold mb-4">Project Bid Estimator</h2>
      <form onSubmit={handleSubmit}>
        <FormInput
          label="Project Type"
          type="select"
          name="projectType"
          options={['residential', 'commercial', 'industrial']}
          value={projectType}
          onChange={(e) => setProjectType(e.target.value)}
        />
        {projectType === 'commercial' && (
          <FormInput
            label="Number of Floors"
            type="number"
            name="numFloors"
            value={numFloors}
            onChange={(e) => setNumFloors(e.target.value)}
          />
        )}
        <FormInput
          label="Area (sq ft)"
          type="number"
          name="area"
          value={area}
          onChange={(e) => setArea(e.target.value)}
        />
        <FormInput
          label="Material Quality"
          type="select"
          name="materialQuality"
          options={['basic', 'standard', 'premium']}
          value={materialQuality}
          onChange={(e) => setMaterialQuality(e.target.value)}
        />
```

```
        <div className="mb-4">
          <label className="block text-gray-700 mb-1" htmlFor="file">Blueprint
PDF Upload</label>
          <input
            type="file"
            id="file"
            accept="application/pdf"
            onChange={handleFileChange}
            className="w-full"
          />
        </div>
        {/* Display PDF preview if a file is selected */}
        {uploadedPdf && <PDFViewer file={uploadedPdf} />}
        <button type="submit" className="bg-blue-500 text-white py-2 px-4
rounded">
          Calculate Estimate
        </button>
      </form>
      {/* Show the estimated cost if available */}
      {estimate && (
        <div className="mt-4 p-2 bg-green-100 text-green-800 rounded">
          Estimated Cost: ${estimate}
        </div>
      )}
    </div>
  );
}
```

- Includes dropdowns and number inputs for project details.
- Conditional logic: if **Commercial** is selected, an extra "Number of Floors" field appears.
- Allows uploading a PDF (e.g., a blueprint) and previews it with `PDFViewer`.
- On submit, sends data to `POST /api/estimate` and displays the returned cost.

frontend/pages/SchedulerForm.jsx

```
// Permit & Inspection Scheduler form component

import React, { useState } from 'react';
import FormInput from '../components/FormInput';

export default function SchedulerForm() {
  // Form state variables
  const [name, setName] = useState('');
  const [date, setDate] = useState('');
  const [time, setTime] = useState('');
  const [zipCode, setZipCode] = useState('');
  const [message, setMessage] = useState('');
```

```
  // Handle form submission
  const handleSubmit = async (e) => {
    e.preventDefault();
    const data = { name, date, time, zipCode };
    try {
      const res = await fetch('http://localhost:3000/api/scheduler', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data)
      });
      const result = await res.json();
      setMessage(result.message);
    } catch (error) {
      console.error('Error:', error);
    }
  };

  return (
    <div className="max-w-md mx-auto bg-white p-4 rounded shadow">
      <h2 className="text-xl font-semibold mb-4">Permit & Inspection Scheduler</
h2>
      <form onSubmit={handleSubmit}>
        <FormInput
          label="Your Name"
          type="text"
          name="name"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
        <FormInput
          label="Preferred Date"
          type="date"
          name="date"
          value={date}
          onChange={(e) => setDate(e.target.value)}
        />
        <FormInput
          label="Preferred Time"
          type="time"
          name="time"
          value={time}
          onChange={(e) => setTime(e.target.value)}
        />
        <FormInput
          label="Zip Code"
          type="text"
          name="zipCode"
```

```
          value={zipCode}
          onChange={(e) => setZipCode(e.target.value)}
        />
        <button type="submit" className="bg-blue-500 text-white py-2 px-4
rounded">
          Submit
        </button>
      </form>
      {/* Show a confirmation message if available */}
      {message && (
        <div className="mt-4 p-2 bg-green-100 text-green-800 rounded">
          {message}
        </div>
      )}
    </div>
  );
}
```

- Collects the user's name, preferred date/time, and ZIP code.
- On submit, sends data to `POST /api/scheduler` and displays a success message from the response.

frontend/components/FormInput.jsx

```
// Reusable form input component supporting text, number, date, select, etc.

import React from 'react';

export default function FormInput({ label, type = "text", name, options, value,
onChange }) {
  return (
    <div className="mb-4">
      <label className="block text-gray-700 mb-1" htmlFor={name}>{label}</label>
      {type === 'select' ? (
        <select
          id={name}
          name={name}
          value={value}
          onChange={onChange}
          className="w-full p-2 border rounded"
        >
          {options && options.map((opt) => (
            <option key={opt} value={opt}>{opt}</option>
          ))}
        </select>
      ) : (
        <input
```

```
          id={name}
          name={name}
          type={type}
          value={value}
          onChange={onChange}
          className="w-full p-2 border rounded"
        />
      )}
    </div>
  );
}
```

- A versatile input component:
- Renders a `<select>` when `type="select"` (using provided `options`).
- Otherwise renders an `<input>` of the given `type`.
- Used by both forms to reduce repetition.

frontend/components/PDFViewer.jsx

```
// PDF viewing component: displays a preview of an uploaded PDF file.

import React from 'react';

export default function PDFViewer({ file }) {
  const [url, setUrl] = React.useState(null);

  React.useEffect(() => {
    if (file) {
      // Create a temporary URL for the file object
      const fileURL = URL.createObjectURL(file);
      setUrl(fileURL);
      // Revoke the object URL on cleanup
      return () => URL.revokeObjectURL(fileURL);
    }
  }, [file]);

  if (!url) return null;
  return (
    <div className="mb-4">
      <h3 className="text-lg font-semibold mb-2">Preview:</h3>
      {/* Iframe to display PDF */}
      <iframe src={url} title="PDF Preview" className="w-full h-64"></iframe>
    </div>
```

```
  );
}
```

- Takes a `file` object (from an `<input type="file">`).
- Creates an object URL to display the PDF inside an `<iframe>` as a preview.

Each part of this codebase is modular and commented for clarity. Together, the backend and frontend work seamlessly:

- **Frontend**: Provides responsive forms and UI for users to submit data and view results/previews.
- **Backend**: Receives form data via REST APIs, processes/stores it with LowDB, and returns responses.

This setup can be expanded with real integrations (email, SMS, mapping APIs) and further validation as needed.
