React + Node (Express) Invoicing System — Fullstack (Scaffolded)

This canvas contains the full scaffold of files for a React-based invoicing system with a Node/Express backend using Azure Cosmos DB (Core/SQL API). You can copy files directly from this document into your project.

File tree

```
invoicing-app/
├─ backend/
│  ├─ package.json
│  ├─ server.js
│  ├─ routes/invoices.js
│  ├─ services/cosmosClient.js
│  └─ .env.example
├─ frontend/
│  ├─ package.json
│  ├─ vite.config.js
│  ├─ index.html
│  └─ src/
│     ├─ main.jsx
│     ├─ App.jsx
│     ├─ api.js
│     └─ components/
│        ├─ InvoiceForm.jsx
│        └─ InvoiceList.jsx
└─ README.md
```

---

backend/package.json

```
{
  "name": "invoicing-backend",
  "version": "1.0.0",
  "main": "server.js",
  "license": "MIT",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "@azure/cosmos": "^3.17.0",
```

```
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "uuid": "^9.0.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

---

backend/.env.example

```
# Copy to .env and fill values (do NOT commit .env to repo)
PORT=4000
COSMOS_ENDPOINT=https://<your-cosmos-account>.documents.azure.com:443/
COSMOS_KEY=<your-cosmos-key>
COSMOS_DATABASE=InvoicesDB
COSMOS_CONTAINER=Invoices
```

---

backend/services/cosmosClient.js

```
const { CosmosClient } = require('@azure/cosmos');
const dotenv = require('dotenv');
dotenv.config();

const endpoint = process.env.COSMOS_ENDPOINT;
const key = process.env.COSMOS_KEY;
const databaseId = process.env.COSMOS_DATABASE || 'InvoicesDB';
const containerId = process.env.COSMOS_CONTAINER || 'Invoices';

if (!endpoint || !key) {
  console.warn('COSMOS_ENDPOINT or COSMOS_KEY not set. Make sure to
configure .env');
}

const client = new CosmosClient({ endpoint, key });

async function getContainer() {
  const { database } = await client.databases.createIfNotExists({ id:
databaseId });
  const { container } = await database.containers.createIfNotExists({ id:
containerId, partitionKey: { kind: 'Hash', paths: ['/id'] } });
  return container;
}
```

```
module.exports = { client, getContainer };
```

---

backend/routes/invoices.js

```
const express = require('express');
const router = express.Router();
const { getContainer } = require('../services/cosmosClient');
const { v4: uuidv4 } = require('uuid');

// CREATE
router.post('/', async (req, res) => {
  try {
    const container = await getContainer();
    const invoice = req.body;
    invoice.id = uuidv4();
    invoice.createdAt = new Date().toISOString();
    const { resource } = await container.items.create(invoice);
    res.status(201).json(resource);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: err.message });
  }
});

// READ all
router.get('/', async (req, res) => {
  try {
    const container = await getContainer();
    const querySpec = { query: 'SELECT * FROM c ORDER BY c.createdAt DESC' };
    const { resources } = await container.items.query(querySpec).fetchAll();
    res.json(resources);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: err.message });
  }
});

// READ one
router.get('/:id', async (req, res) => {
  try {
    const container = await getContainer();
    const { resource } = await container.item(req.params.id,
req.params.id).read();
    if (!resource) return res.status(404).json({ error: 'Not found' });
    res.json(resource);
  } catch (err) {
    console.error(err);
```

```
    res.status(500).json({ error: err.message });
  }
});

// UPDATE
router.put('/:id', async (req, res) => {
  try {
    const container = await getContainer();
    const id = req.params.id;
    const updated = req.body;
    updated.id = id; // ensure same id
    const { resource } = await container.items.upsert(updated);
    res.json(resource);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: err.message });
  }
});

// DELETE
router.delete('/:id', async (req, res) => {
  try {
    const container = await getContainer();
    const id = req.params.id;
    await container.item(id, id).delete();
    res.status(204).end();
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: err.message });
  }
});

module.exports = router;
```

---

backend/server.js

```
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const dotenv = require('dotenv');
dotenv.config();

const invoiceRoutes = require('./routes/invoices');

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

```
app.use('/api/invoices', invoiceRoutes);

const port = process.env.PORT || 4000;
app.listen(port, () => console.log(`Server listening on port ${port}`));
```

---

frontend/package.json

```
{
  "name": "invoicing-frontend",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "18.2.0",
    "react-dom": "18.2.0"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "3.1.0",
    "vite": "5.0.0"
  }
}
```

---

frontend/vite.config.js

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173
  }
});
```

---

frontend/index.html

```
<!doctype html>
<html>
```

```
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Invoicing App</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

---

frontend/src/main.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
import './styles.css'

createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
)
```

---

frontend/src/api.js

```
const base = import.meta.env.VITE_API_BASE || 'http://localhost:4000/api';

export async function fetchInvoices() {
  const r = await fetch(`${base}/invoices`);
  return r.json();
}
export async function createInvoice(data) {
  const r = await fetch(`${base}/invoices`, { method: 'POST', headers:
{'Content-Type':'application/json'}, body: JSON.stringify(data) });
  return r.json();
}
export async function updateInvoice(id, data) {
  const r = await fetch(`${base}/invoices/${id}`, { method: 'PUT', headers:
{'Content-Type':'application/json'}, body: JSON.stringify(data) });
  return r.json();
}
export async function deleteInvoice(id) {
  const r = await fetch(`${base}/invoices/${id}`, { method: 'DELETE' });
```

```
  return r;
}
```

---

frontend/src/App.jsx

```
import React, { useState, useEffect } from 'react';
import InvoiceForm from './components/InvoiceForm';
import InvoiceList from './components/InvoiceList';
import * as api from './api';

export default function App() {
  const [invoices, setInvoices] = useState([]);
  const [editing, setEditing] = useState(null);

  async function load() {
    try {
      const data = await api.fetchInvoices();
      setInvoices(data || []);
    } catch (err) {
      console.error('Failed to load invoices', err);
    }
  }

  useEffect(() => { load(); }, []);

  async function handleSaved(data) {
    try {
      if (editing) {
        await api.updateInvoice(editing.id, { ...editing, ...data });
        setEditing(null);
      } else {
        await api.createInvoice(data);
      }
      await load();
    } catch (err) {
      console.error(err);
    }
  }

  async function handleEdit(inv) { setEditing(inv); }
  async function handleDelete(id) { await api.deleteInvoice(id); await
load(); }

  return (
    <div style={{display:'grid',gridTemplateColumns:'360px 1fr',gap:
24,padding:24}}>
      <div>
        <h3>{editing ? 'Edit Invoice' : 'New Invoice'}</h3>
```

```
        <InvoiceForm onSaved={handleSaved} editing={editing} />
      </div>
      <div>
        <h3>Invoices</h3>
        <InvoiceList invoices={invoices} onEdit={handleEdit}
onDelete={handleDelete} />
      </div>
    </div>
  );
}
```

---

frontend/src/components/InvoiceForm.jsx

```
import React, { useState, useEffect } from 'react';

export default function InvoiceForm({ onSaved, editing }) {
  const [form, setForm] = useState({ customer: '', items: [{ description:
'', qty: 1, price: 0 }], dueDate: '' });

  useEffect(() => { if (editing) setForm(editing); }, [editing]);

  function updateItem(index, field, value) {
    const copy = { ...form };
    copy.items = copy.items.map((it, i) => i === index ? { ...it, [field]:
value } : it);
    setForm(copy);
  }

  function addItem() { setForm({ ...form, items: [...form.items, {
description: '', qty: 1, price: 0 }] }); }
  function removeItem(i) { setForm({ ...form, items: form.items.filter((_,
idx) => idx !== i) }); }

  function handleSubmit(e) {
    e.preventDefault();
    const total = form.items.reduce((s, it) => s + (Number(it.qty) *
Number(it.price || 0)), 0);
    onSaved({ ...form, total });
    setForm({ customer: '', items: [{ description: '', qty: 1, price: 0 }],
dueDate: '' });
  }

  return (
    <form onSubmit={handleSubmit} className="p-4 border rounded">
      <div>
        <label>Customer</label>
        <input value={form.customer} onChange={e => setForm({ ...form,
customer: e.target.value })} required />
```

```
      </div>
      <div>
        <label>Due date</label>
        <input type="date" value={form.dueDate} onChange={e =>
setForm({ ...form, dueDate: e.target.value })} />
      </div>

      <h4>Items</h4>
      {form.items.map((it, idx) => (
        <div key={idx} style={{display:'grid',gridTemplateColumns:'1fr 80px
100px 40px',gap:'8px',alignItems:'center'}}>
          <input placeholder="description" value={it.description}
onChange={e => updateItem(idx, 'description', e.target.value)} required />
          <input type="number" min="1" value={it.qty} onChange={e =>
updateItem(idx, 'qty', Number(e.target.value))} />
          <input type="number" min="0" step="0.01" value={it.price}
onChange={e => updateItem(idx, 'price', Number(e.target.value))} />
          <button type="button" onClick={() => removeItem(idx)}>X</button>
        </div>
      ))}
      <div>
        <button type="button" onClick={addItem}>Add item</button>
      </div>

      <div style={{marginTop:12}}>
        <button type="submit">Save Invoice</button>
      </div>
    </form>
  );
}
```

---

frontend/src/components/InvoiceList.jsx

```
import React from 'react';

export default function InvoiceList({ invoices, onEdit, onDelete }) {
  return (
    <div>
      <table style={{width:'100%', borderCollapse:'collapse'}}>
        <thead>
          <tr>
            <th>Customer</th>
            <th>Due</th>
            <th>Total</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
```

```
          {invoices.map(inv => (
            <tr key={inv.id}>
              <td>{inv.customer}</td>
              <td>{inv.dueDate ? new
Date(inv.dueDate).toLocaleDateString() : '-'}</td>
              <td>{inv.total}</td>
              <td>
                <button onClick={() => onEdit(inv)}>Edit</button>
                <button onClick={() => onDelete(inv.id)}>Delete</button>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}
```

README.md

```
# Invoicing App (React + Node + Azure Cosmos DB)

## Setup

1. Backend
   - Copy `backend/.env.example` to `backend/.env` and fill in your Cosmos DB
connection info.
   - `cd backend && npm install`
   - `npm run dev` (needs nodemon) or `npm start`

2. Frontend
   - `cd frontend && npm install`
   - `npm run dev` (Vite)

Open the Vite URL (usually http://localhost:5173). The frontend expects the
backend at `http://localhost:4000/api` by default. Use `VITE_API_BASE` env in
frontend to change base URL.

## Next steps / production notes
- Do not commit secrets. Use Azure Key Vault / Azure App Service settings.
- Replace partition key with a meaningful value for scale (e.g., customerId
or companyId).
- Add authentication & validation before production.
```

If you want any of these next, tell me which and I will scaffold them into the canvas as additional files:

- Dockerfiles (backend + frontend) and docker-compose
- Azure deployment files (ARM/Bicep or GitHub Actions + Azure Web Apps / Cosmos provisioning)
- Convert backend to Azure Functions (serverless) and a Static Web Apps config
- Add authentication (Azure AD / Microsoft identity) and protected API