You're working with a Node.js backend (mainApp.js) and a static HTML/CSS/JS frontend (RetailApp.html) for a retail application, and using MSSQL for your database. You want to deploy this so others can access it online, with the option of keeping the MSSQL database on-premise or within the LAN.

**🔧 Here's a full deployment strategy:**

**✅ 1. Backend & Frontend Deployment**

**Option A: Deploy to a Cloud Provider (Recommended)**

This lets anyone with the URL access your app.

**Use providers like:**

* [Render](https://render.com/) – Free tier, easy Node.js deployment.
* [Railway](https://railway.app/) – Easy backend and frontend deploy.
* [Vercel](https://vercel.com/) or [Netlify](https://netlify.com/) – Good for frontend, but backend support is limited.
* [Heroku](https://www.heroku.com/) – Still works for small apps, though limited.

**Steps to deploy with Render (as example):**

1. Push your project to GitHub.
2. Go to [Render.com](https://render.com/), sign in.
3. Create a **Web Service**, connect your GitHub repo.
4. Set the **start command** as:
5. node BackEnd/mainApp.js
6. Add environment variables from your .env file to Render.
7. It will assign a public URL to your backend API.

**Frontend Deployment:**  
You can serve the RetailApp.html, css, images, and src folder via:

* A static site host (Vercel, Netlify, GitHub Pages)
* OR serve it from the Node backend (mainApp.js) using express.static.

**Example Express setup:**

const express = require('express');

const app = express();

const path = require('path');

app.use(express.static(path.join(\_\_dirname, '../'))); // serve css, images, etc.

app.get('/', (req, res) => {

res.sendFile(path.join(\_\_dirname, '../RetailApp.html'));

});

app.listen(process.env.PORT || 3000, () => {

console.log("Server running");

});

**✅ 2. Database (MSSQL) Deployment**

**Option A: On-Premise with Static Public IP**

* Host your MSSQL Server on a machine with a **static public IP**.
* **Open TCP port 1433** (default for MSSQL) in your firewall.
* Set up **SQL Server authentication**, not Windows auth.
* Whitelist your backend server IP (on cloud) to connect securely.

🔐 Use a strong username/password. Consider using a VPN or SSH tunnel if security is a concern.

**Option B: LAN-Only Access**

* Backend app must also be deployed **inside the LAN** (e.g., on a local Windows/Linux server).
* Use a reverse proxy (like Nginx) to expose the backend via public IP or domain, if needed.

**✅ 3. Other Considerations**

**SSL/TLS (HTTPS)**

* If using a public URL, ensure HTTPS is enabled (Render/Netlify do this automatically).

**Environment Variables**

* Do **not** expose your .env or DB credentials in the frontend.
* Use a .env file and dotenv in Node.js.

**Security**

* Sanitize all inputs to protect against SQL injection.
* Use helmet, cors, and express-rate-limit in Express for basic API security.

**✅ Suggested Deployment Setup:**

| **Component** | **Hosting Option** | **Notes** |
| --- | --- | --- |
| Frontend | Netlify/Vercel/Render | Deploy RetailApp.html and static files |
| Backend API | Render/Heroku/Railway | Deploy mainApp.js with MSSQL DB connection logic |
| MSSQL Database | On-Premise with Public IP | Use IP whitelisting and firewall for access control |

Would you like help setting up deployment with Render or another provider step-by-step?