GenealogyApp Web Deployment Guide

This guide walks you through deploying the Genealogy app to the web using a practical, low-friction setup:

- Database: MongoDB Atlas (managed, free tier)
- Backend: Render (Node/Express Web Service)
- Frontend: Vercel (React)

It also includes an optional single-host approach (serving the React build from Express) if you prefer one service.

---

0) Prerequisites

- GitHub repository: `https://github.com/perelgutTrios/GenealogyApp`
- Accounts:
  - MongoDB Atlas: https://www.mongodb.com/atlas
  - Render: https://render.com/
  - Vercel: https://vercel.com/
- Optional: A custom domain (you can add this later)

---

1) Create a MongoDB Atlas Cluster

1. Sign up and create a new project + free cluster.
2. Create a database user with username/password.
3. Network access:
   - Quick start: allow access from anywhere (0.0.0.0/0).
   - For production, restrict to your hosting provider's egress IP(s).
4. Get the connection string (SRV URI), e.g.: `mongodb+srv://<user>:<password>@<cluster>.mongodb.net/genealogy-db?retryWrites=true&w=majority`

Keep this URI for the backend `MONGODB_URI`.

---

2) Deploy the Backend on Render (Express API)

1. Create a new "Web Service" in Render.
2. Connect your GitHub repo and select the `backend` subfolder as the root.
3. Settings:
   - Runtime: Node
   - Build Command: `npm install`
   - Start Command: `node server.js` (or `npm start` if defined)
4. Environment variables (add these in Render):
   - `NODE_ENV=production`
   - `MONGODB_URI=<your Atlas URI>`
   - `JWT_SECRET=<long-random-secret>`
   - `ENCRYPTION_KEY=<32-char-random-key>`
   - `CLIENT_URL=https://<your-frontend-domain>` (set later when Vercel URL is known)
   - `TRUST_PROXY=true`
   - `ENABLE_MOCK_SOURCES=false`
   - Optional AI:
     - `GEMINI_API_KEY=<your key>`

        - `GEMINI_ENABLE_WEB_GROUNDING=false`
        - `OPENAI_API_KEY=<your key>`
    - Optional providers:
        - `FAMILYSEARCH_CLIENT_ID=...`
        - `FAMILYSEARCH_CLIENT_SECRET=...`
        - `CHRONICLING_AMERICA_API_BASE=https://chroniclingamerica.loc.gov`
        - `WIKITREE_API_BASE=https://api.wikitree.com/api.php`
        - `WIKITREE_APP_ID=GenealogyApp`

5. Deploy and wait for the service to boot.
6. Verify health:
    - Open `https://your-backend.onrender.com/api/health` and confirm a 200 JSON response.

Notes:

- `server.js` already honors `PORT` injected by Render and sets CORS based on `CLIENT_URL`.
- `TRUST_PROXY=true` ensures correct IP/logging/HTTPS behavior behind Render's proxy.

---

3) Deploy the Frontend on Vercel (React)

1. Create a new project in Vercel and import the same GitHub repo.
2. Choose the `frontend` folder as the project root.
3. Environment variables (Vercel → Project Settings → Environment Variables):
    - `REACT_APP_API_URL=https://your-backend.onrender.com/api`
4. Deploy. You'll get a domain like `https://your-frontend.vercel.app/`.
5. Test the app: register/login, open the AI Research Assistant, run a search, etc.

Notes:

- Chronicling America and WikiTree can return results without API keys.
- FamilySearch requires credentials to return live data.
- Gemini is recommended for structured JSON (free-tier friendly), with OpenAI as fallback.

---

4) Wire CORS and Client URL Precisely

- Update `CLIENT_URL` in Render (backend) to your Vercel domain:
    - `CLIENT_URL=https://your-frontend.vercel.app`
- Redeploy/restart the backend to apply.

This ensures the backend's CORS allows your frontend origin.

---

5) Optional: Custom Domains (HTTPS)

- Vercel (Frontend): add your domain (e.g., `app.yourdomain.com`). Vercel will guide DNS + SSL.
- Render (Backend): add a domain (e.g., `api.yourdomain.com`).
- Update envs:
    - Vercel: `REACT_APP_API_URL=https://api.yourdomain.com/api`
    - Render: `CLIENT_URL=https://app.yourdomain.com`
- Redeploy both.

---

6) Optional: Single-Host Deployment (Serve React from Express)

If you prefer one service hosting both frontend and backend:

1. In the frontend, set `REACT_APP_API_URL=/api` (relative path) for production.
2. Build the frontend locally (or via CI):

```
cd frontend
npm install
npm run build
```

3. Copy the build output into the backend (e.g., `backend/public`).
4. In `backend/server.js`, add static hosting:

```
const path = require('path');
app.use(express.static(path.join(__dirname, 'public')));

// API routes above ...

// Fallback to index.html for client-side routing
app.get('*', (req, res) => {
  if (req.path.startsWith('/api')) {
    return res.status(404).json({ message: 'Route not found' });
  }
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
```

5. Deploy only the backend service to Render (no separate Vercel app).

Pros: single URL, no CORS. Cons: any frontend change requires a new build + backend redeploy.

---

7) Security & Hardening Checklist

- Secrets:
  - Use strong values for `JWT_SECRET` and `ENCRYPTION_KEY`.
  - Never commit real secrets; use environment variables.
- CORS:
  - Set `CLIENT_URL` to your actual frontend domain (avoid localhost in production).
- Rate limiting:
  - `express-rate-limit` is enabled; adjust thresholds if needed.
- Proxies:
  - `TRUST_PROXY=true` on Render for accurate IPs and secure cookies.
- Mock sources:
  - Keep `ENABLE_MOCK_SOURCES=false` to avoid fabricated records.

---

8) Observability

- Render: tail logs for API errors and performance.

- MongoDB Atlas: monitor connections, slow queries, and storage.
- Optional: add error tracking (e.g., Sentry) later.

---

9) Troubleshooting

- CORS error in browser console:
  - Ensure backend `CLIENT_URL` matches your frontend domain exactly (protocol + host + port).
- 401 and redirect loops:
  - Token expired or missing; re-login. Confirm `REACT_APP_API_URL` points to the correct backend.
- MongoDB connection errors:
  - Check Atlas network access and `MONGODB_URI` correctness.
- "ECONNREFUSED" in local dev (Create React App proxy):
  - Ensure backend is running on port 5000 locally. In production, use `REACT_APP_API_URL` instead of a CRA proxy.
- 0 records found:
  - Enable more sources (WikiTree, Chronicling America), widen time ranges, or increase name/location variations.

---

10) Quick Validation

- Backend health:
  - `GET /api/health` should return 200 OK with JSON.
- Frontend smoke test:
  - Load the site, register/login, view GEDCOM stats, open AI Research Assistant, search external records.

---

11) Optional: Governance & CI

- Branch protection (master): require PRs and prevent force pushes.
- CODEOWNERS: auto-request reviewers.
- CI: add a lightweight GitHub Actions workflow (lint/build/smoke). After stable, enable "required status checks" on the protected branch.

Environment Variables Reference

Backend (Render):

- Core
  - `NODE_ENV=production`
  - `PORT` (injected by Render)
  - `MONGODB_URI`
  - `JWT_SECRET`
  - `ENCRYPTION_KEY` (32 chars)
  - `CLIENT_URL` (frontend origin)
  - `TRUST_PROXY=true`
  - `ENABLE_MOCK_SOURCES=false`
- AI & Providers (optional)

- `GEMINI_API_KEY`
- `GEMINI_ENABLE_WEB_GROUNDING=false`
- `OPENAI_API_KEY`
- `FAMILYSEARCH_CLIENT_ID`
- `FAMILYSEARCH_CLIENT_SECRET`
- `CHRONICLING_AMERICA_API_BASE`
- `WIKITREE_API_BASE`
- `WIKITREE_APP_ID`

Frontend (Vercel):

- `REACT_APP_API_URL` (e.g., `https://api.yourdomain.com/api` or Render URL)

That's it!

You now have a clear, step-by-step path to get GenealogyApp live on the web. If you'd like, we can automate parts of this with a CI/CD workflow, or switch to the single-host deployment to simplify hosting.