# End-to-end Flow: Frontend → Backend → Database → Deploy

A single document that walks you through a complete and secure workflow from local development to production deployment for a typical MERN-style app (React frontend, Express/Node backend, MongoDB database), including environment handling, build & deploy steps (Netlify for frontend, Render for backend), monitoring, rollbacks, and security best practices.

## Table of contents

1. Overview
2. Local development setup (FE & BE)
3. Environment variables & secrets handling
4. Backend (Express) — structure & important patterns
5. Frontend (React) — structure & API usage
6. Database (MongoDB Atlas) — connection & security
7. CI/CD & deployment (Netlify + Render)
8. Health checks, monitoring & logging
9. Common problems & diagnostics
10. Security checklist
11. Quick commands reference
12. Appendix: sample files & snippets

## 1. Overview

A standard production flow:

1. Developer works locally on frontend (React) and backend (Express/Node). Frontend calls backend using a configurable REACT\_APP\_API\_URL.
2. Backend talks to DB (MongoDB Atlas) using a MONGO\_URI stored in server-side environment variables (Render).
3. Frontend is built and deployed on Netlify (static assets). Backend is deployed on Render (Node service) and exposes API endpoints.
4. CI/CD pipelines or hosted deployers (Netlify & Render) build and publish the app. Secrets are stored in provider environment variable UI.
5. Monitoring, logs, health endpoints, and alerts ensure the system is healthy.

## 2. Local development setup (FE & BE)

### Folder layout (example)

repo/  
├─ frontend/ # React app  
│ ├─ src/  
│ └─ package.json  
├─ backend/ # Express app  
│ ├─ src/  
│ ├─ routes/  
│ └─ package.json  
└─ .gitignore

### Start backend locally

* Add a .env for local dev only (never commit it). Example .env.development (gitignored):

PORT=5000  
MONGO\_URI=mongodb+srv://dev\_user:dev\_pass@cluster.mongodb.net/devdb  
JWT\_SECRET=devsecret123

* Run the backend locally:

cd backend  
npm install  
npm run dev # or npm start  
# ensure server listens on process.env.PORT or fallback

### Start frontend locally

* In frontend, use REACT\_APP\_API\_URL for the API base:

// src/utils/api.js  
import axios from 'axios';  
const api = axios.create({  
 baseURL: process.env.REACT\_APP\_API\_URL || 'http://localhost:5000',  
 withCredentials: true,  
});  
export default api;

* Local env file .env.development (gitignored):

REACT\_APP\_API\_URL=http://localhost:5000

* Run:

cd frontend  
npm install  
npm run start

## 3. Environment variables & secrets handling

**Principles** - Never commit secrets (DB URIs, JWT secrets, passwords) into source control. - Keep secrets only in the server-side environment (Render/Atlas) and CI secret stores. - Only public, non-sensitive values (API base URL) can live in Netlify as REACT\_APP\_ vars.

**Where to store** - MongoDB URI, JWT\_SECRET, SMTP credentials, admin secrets: Render service environment variables (or secret manager). - Frontend config (public): REACT\_APP\_API\_URL on Netlify.

**Netlify vs Render** - Netlify: for building & hosting static frontend. Any REACT\_APP\_ var gets embedded in the client bundle — therefore *public*. - Render: backend service env vars are not exposed to the client; safe for secrets.

**Emergency (if leaked)** 1. Rotate the secret at the provider (DB, JWT signing secret — reissue tokens if needed). 2. Remove secret from repo (git history purge if committed). 3. Replace with new secret in Render env variables and redeploy.

## 4. Backend (Express) — structure & important patterns

### Minimal entry (server.js)

* Use process.env.PORT.
* Connect to DB before starting.
* Add /health and /\_\_routes debug endpoints (remove latter in prod).
* Add global error handler.

**Example pattern** (pseudocode):

import express from 'express';  
import connectDB from './config/db.js';  
const app = express();  
app.use(express.json());  
app.get('/health', (req,res)=>res.json({status:'ok'}));  
// mount routes  
connectDB().then(()=>{  
 const PORT = process.env.PORT || 5000;  
 app.listen(PORT, ()=>console.log('listening',PORT));  
});

### Error handling & async routes

* Wrap async route handlers or use a small wrapper to catch and forward errors to next(err).
* Use a centralized error middleware to log and return safe error messages.

### CORS

* Configure CORS origin list through env (e.g. FRONTEND\_URL), allow credentials if you use cookies.
* Example CORS config:

app.use(cors({  
 origin: (origin, cb) => { if(!origin) return cb(null,true); if(allowed.has(origin)) cb(null,true); else cb(new Error('Not allowed')); },  
 credentials: true  
}));

## 5. Frontend (React) — API usage & builds

### API client

* Centralize API base in src/utils/api.js using process.env.REACT\_APP\_API\_URL.
* Use withCredentials: true if your backend uses cookies for auth.

### Build-time & envs

* Local dev: .env.development (ignored) with REACT\_APP\_API\_URL=http://localhost:5000.
* Netlify: set REACT\_APP\_API\_URL=https://travelstrem-test.onrender.com in Netlify site settings.

### Build command

* In Netlify build settings or netlify.toml, the build command is typically:

cd frontend && npm ci && npm run build

* Netlify will run a production build and publish frontend/build.

## 6. Database (MongoDB Atlas)

### Best practices

* Use least-privilege DB user for the app. Create separate users for dev/prod.
* Whitelist connection IPs if possible (or use VPC peering for strict setups).
* Rotate credentials if leaked.

### Connection string

* Store MONGO\_URI in Render env variables.
* Use options like retryWrites=true&w=majority as needed.

## 7. CI/CD & deployment (Netlify + Render)

### Frontend (Netlify)

1. Connect Git repo to Netlify (select build directory frontend).
2. Set environment variable: REACT\_APP\_API\_URL=https://travelstrem-test.onrender.com.
3. Netlify runs npm run build and publishes static files.
4. If you need to omit secret scanning warnings for false positives, use SECRETS\_SCAN\_OMIT\_KEYS=PORT,SMTP\_PORT or SECRETS\_SCAN\_ENABLED=false via Netlify envs.

### Backend (Render)

1. Create a Web Service on Render and link to your Git repo (backend folder or root with start command).
2. In Render service settings, set env vars: MONGO\_URI, JWT\_SECRET, ADMIN\_CREATION\_SECRET, FRONTEND\_URL.
3. Render will build and run npm start (or your start command). Ensure process.env.PORT is used.

### Deployment flow

* Push to main/master triggers Netlify & Render builds (or you can configure branch deploys).
* Both providers show logs — check startup logs and test /health and API endpoints after deploy.

### Rollbacks

* Netlify: revert to previous deploy from deploy list.
* Render: choose previous deploy or redeploy a previous commit.

## 8. Health checks, monitoring & logging

* Add /health endpoint for uptime checks.
* Use a logging library and ensure logs go to stdout so Render captures them.
* Use monitoring/alerts: Render health checks, Datadog / Sentry for errors, Netlify deploy alerts.
* Set up uptime monitor (UptimeRobot or similar) calling /health and alerting on failures.

## 9. Common problems & diagnostics

* **Case-sensitive paths**: Mac can be case-insensitive; Linux (Netlify) is not. Ensure import paths match exact filename casing.
* **ESLint errors in CI**: CI=true treats warnings as errors. Fix lint or set CI=false in netlify env to relax (not recommended long-term).
* **Secrets leakage**: Don’t put secrets in Netlify; store them on Render. If leaked, rotate immediately.
* **CORS errors**: Ensure FRONTEND\_URL is allowed origin in backend.
* **Wrong API base**: Ensure REACT\_APP\_API\_URL points to Render URL in Netlify envs.

## 10. Security checklist (must-dos)

* Never commit .env to repo.
* Use different DB users for dev/staging/prod.
* Rotate keys when leaked.
* Use HTTPS for all endpoints.
* Validate and sanitize inputs server-side.
* Use rate-limiting for public endpoints (contact forms, login).
* Enable security headers (CSP, X-Content-Type-Options, X-Frame-Options, HSTS).
* Limit CORS origins to known frontends.
* Encrypt backups and restrict access.

## 11. Quick commands reference

* Start backend locally:

cd backend  
npm install  
npm run dev

* Start frontend locally:

cd frontend  
npm install  
npm start

* Local production build:

cd frontend  
CI=true npm run build

* Remove .env from git and ignore:

git rm --cached .env  
echo ".env" >> .gitignore  
git add .gitignore  
git commit -m "chore: ignore .env"  
git push

* Rotate a secret (example generate new secret):

node -e "console.log(require('crypto').randomBytes(48).toString('hex'))"

## 12. Appendix: sample snippets

### Example axios client (frontend)

// src/utils/api.js  
import axios from 'axios';  
const api = axios.create({  
 baseURL: process.env.REACT\_APP\_API\_URL || 'http://localhost:5000',  
 withCredentials: true,  
});  
export default api;

### Example health endpoint (backend)

app.get('/health', (req,res)=>{  
 res.json({status:'ok', uptime:process.uptime()});  
});

## Final notes

* Keep secrets on the server (Render) and public config on Netlify only.
* Use environment variables to switch API base URLs between local/stage/prod.
* Automate small checks (grep for secrets) in your pre-commit or CI to avoid accidental leaks.

If you’d like, I can: - Export this document as a downloadable markdown file or PDF. - Create a checklist PR that applies .gitignore changes and patches api.js/server.js in your repo. - Walk you step-by-step through rotating the exposed secrets (without you pasting them here).

Which next step would you like?