Crownshiftlogistics — Vercel conversion, Google GenAI, admin features, and asset optimization

This document contains ready-to-drop code snippets, GitHub Actions workflow, and instructions to:

- Convert Firebase Functions into Vercel Serverless endpoints
- Wire the AI proxy to Google GenAI (with provider-fallback logic)
- Add admin dashboard features: live revenue aggregation (Firestore) and invoice PDF generator using Puppeteer
- Replace and optimize public/logo.jpg into responsive WebP/PNG assets

> **Note:** You still need to add secrets / environment variables in Vercel or GitHub Secrets (examples shown). Replace placeholders with your real values.

---

1) Project layout (suggested)

```
/ (repo root)
  /api
    genai-proxy.js          # Vercel serverless endpoint for AI proxy
    invoice-pdf.js          # serverless endpoint that returns invoice PDF
    revenue-aggregate.js    # serverless endpoint that aggregates revenue
from Firestore (admin-only)
  /admin
    dashboard.js            # React page for admin dashboard
    invoice-template.html   # HTML template for Puppeteer invoice generator
  /public
    logo.jpg                # original (will be replaced by optimized
variants)
  /scripts
    optimize-logo.js        # sharp-based image optimizer script
  package.json
  .github/workflows/deploy-vercel.yml
  README.md
```

---

2) Dependencies (package.json snippets)

Add these dependencies (install with `npm i`):

```
{
  "dependencies": {
    "firebase-admin": "latest",
    "firebase": "latest",
    "puppeteer-core": "latest",
```

```
    "chrome-aws-lambda": "latest",
    "sharp": "latest",
    "form-data": "latest"
  }
}
```

You may also add any Google client you'd like; the serverless code below uses `fetch` to call Google GenAI REST endpoints so no extra SDK is required.

---

3) Environment variables (Vercel / GitHub Secrets)

- `GENAI_PROVIDER` — `google` or `openai` (default fallback `openai`)
- `GOOGLE_API_KEY` — Google GenAI API key (if using Google)
- `GOOGLE_MODEL` — e.g. `models/text-bison-001` (replace with your model name)
- `OPENAI_API_KEY` — OpenAI API key (fallback)
- `FIREBASE_PROJECT_ID`, `FIREBASE_CLIENT_EMAIL`, `FIREBASE_PRIVATE_KEY` — for `firebase-admin` (service account)
- `VERCEL_TOKEN` — for GitHub Action deploy (if using GH Actions to deploy)

Important: On Vercel set `FIREBASE_PRIVATE_KEY` with newlines escaped (or use Vercel secret management).

---

4) `api/genai-proxy.js` — Vercel serverless function with provider fallback

```
// api/genai-proxy.js
export default async function handler(req, res) {
  const provider = process.env.GENAI_PROVIDER || 'openai';
  const body = req.body || {};

  try {
    if (provider === 'google') {
      // Google GenAI REST request
      const GOOGLE_API_KEY = process.env.GOOGLE_API_KEY;
      const GOOGLE_MODEL = process.env.GOOGLE_MODEL || 'models/text-
bison-001';
      if (!GOOGLE_API_KEY) return res.status(500).json({ error: 'Missing
GOOGLE_API_KEY' });

      const resp = await fetch(`https://generativelanguage.googleapis.com/v1/
${GOOGLE_MODEL}:generateText?key=${GOOGLE_API_KEY}`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          // adapt fields as required by the model
          "prompt": body.prompt || "",
```

```
          "temperature": body.temperature ?? 0.2,
          "maxOutputTokens": body.maxOutputTokens ?? 512
        })
      });

      const data = await resp.json();
      return res.status(resp.status).json(data);
    }

    // fallback to OpenAI
    const OPENAI_API_KEY = process.env.OPENAI_API_KEY;
    if (!OPENAI_API_KEY) return res.status(500).json({ error: 'Missing
OPENAI_API_KEY' });

    const openaiResp = await fetch('https://api.openai.com/v1/chat/
completions', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${OPENAI_API_KEY}`
      },
      body: JSON.stringify({
        model: body.model || 'gpt-4o-mini',
        messages: body.messages || [{ role: 'user', content: body.prompt ||
'' }],
        max_tokens: body.max_tokens || 512
      })
    });

    const openaiData = await openaiResp.json();
    return res.status(openaiResp.status).json(openaiData);
  } catch (err) {
    console.error('genai-proxy error', err);
    return res.status(500).json({ error: String(err) });
  }
}
```

Notes: - The Google GenAI endpoint and request body may vary depending on API version — the code above uses a flexible URL and simple body that should be adapted to match the exact Google GenAI API version you use. - Vercel Node has `fetch` available. If you prefer axios or node-fetch, add it to dependencies and import it.

---

5) `api/revenue-aggregate.js` — serverless endpoint for live revenue aggregation

```
// api/revenue-aggregate.js
import admin from 'firebase-admin';
```

```
if (!admin.apps.length) {
  admin.initializeApp({
    credential: admin.credential.cert({
      projectId: process.env.FIREBASE_PROJECT_ID,
      clientEmail: process.env.FIREBASE_CLIENT_EMAIL,
      // private key needs newlines fixed: replace '\\n' with '\n'
      privateKey: process.env.FIREBASE_PRIVATE_KEY?.replace(/\\n/g, '\n')
    }),
    databaseURL: `https://${process.env.FIREBASE_PROJECT_ID}.firebaseio.com`
  });
}

const db = admin.firestore();

export default async function handler(req, res) {
  // Authentication check here (admin-only) — e.g. a simple API key header
  if (req.headers['x-admin-key'] !== process.env.ADMIN_API_KEY) {
    return res.status(401).json({ error: 'Unauthorized' });
  }

  try {
    // Example: invoices collection with fields: amount, currency, paidAt
(timestamp)
    const snapshot = await db.collection('invoices').where('paid', '==',
true).get();
    let total = 0;
    const byMonth = {};

    snapshot.forEach(doc => {
      const data = doc.data();
      const amount = Number(data.amount) || 0;
      total += amount;
      const m = data.paidAt ? new Date(data.paidAt._seconds * 1000) : null;
      const monthKey = m ? `${m.getUTCFullYear()}-${String(m.getUTCMonth()
+1).padStart(2,'0')}` : 'unknown';
      byMonth[monthKey] = (byMonth[monthKey] || 0) + amount;
    });

    return res.json({ total, byMonth });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: String(err) });
  }
}
```

Client-side you can poll or use a websocket / server-sent events to display live numbers. A simple `setInterval` polling every 10s is often fine.

---

6) `api/invoice-pdf.js` — generate invoice PDF (Puppeteer in serverless)

```
// api/invoice-pdf.js
import chromium from 'chrome-aws-lambda';
import puppeteer from 'puppeteer-core';

export default async function handler(req, res) {
  // Expect invoice data in req.body or fetch invoice by ID
  const invoice = req.body.invoice;
  if (!invoice) return res.status(400).send('Missing invoice payload');

  try {
    const browser = await puppeteer.launch({
      args: chromium.args,
      defaultViewport: chromium.defaultViewport,
      executablePath: await chromium.executablePath,
      headless: chromium.headless,
    });

    const page = await browser.newPage();

    // Option A: render an HTML string
    const html = `<!doctype html><html><head><meta
charset="utf-8"><title>Invoice</title></head><body>${invoice.html}</body></
html>`;
    await page.setContent(html, { waitUntil: 'networkidle0' });

    const pdfBuffer = await page.pdf({ format: 'A4', printBackground:
true });
    await browser.close();

    res.setHeader('Content-Type', 'application/pdf');
    res.setHeader('Content-Disposition', `attachment; filename=invoice-$
{invoice.id || Date.now()}.pdf`);
    return res.send(pdfBuffer);
  } catch (err) {
    console.error('invoice-pdf error', err);
    return res.status(500).json({ error: String(err) });
  }
}
```

Important notes: - Use `puppeteer-core` + `chrome-aws-lambda` on Vercel serverless. This combination works well on many serverless platforms but occasionally needs tuning. Alternatively, consider generating PDFs on a separate server/container if you run into cold-start or size limits.

---

7) Admin dashboard client-side: live revenue aggregation (example React fetch)

```
// admin/dashboard.js (React, simplified)
import { useEffect, useState } from 'react';

export default function Dashboard() {
  const [data, setData] = useState(null);

  async function fetchAgg() {
    const resp = await fetch('/api/revenue-aggregate', {
      headers: { 'x-admin-key': process.env.NEXT_PUBLIC_ADMIN_API_KEY }
    });
    const json = await resp.json();
    setData(json);
  }

  useEffect(() => {
    fetchAgg();
    const id = setInterval(fetchAgg, 10000);
    return () => clearInterval(id);
  }, []);

  return (
    <div>
      <h1>Revenue</h1>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
}
```

Security: Keep admin routes gated — do not publish `NEXT_PUBLIC_ADMIN_API_KEY` in public client builds; instead use a secure session on the server that proxies requests to `api/revenue-aggregate`.

---

8) Invoice HTML template (example)

Create `admin/invoice-template.html` and load it into the Puppeteer page when generating PDFs. Keep styling inline (serverless friendly).

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <style>
      body { font-family: Arial, sans-serif; padding: 24px }
      .header { display:flex; justify-content:space-between }
    </style>
```

```
  </head>
  <body>
    <div class="header">
      <div><img src="CID_OR_PUBLIC_URL_TO_LOGO" style="height:60px"/></div>
      <div><h2>Invoice #{{id}}</h2></div>
    </div>
    <hr />
    <div>{{line_items_html}}</div>
    <hr />
    <div>Total: {{total}}</div>
  </body>
</html>
```

When using a template string server-side, replace placeholders with actual invoice values.

---

9) Optimize `public/logo.jpg` using `sharp` (script)

Create `scripts/optimize-logo.js`:

```
// scripts/optimize-logo.js
const sharp = require('sharp');
const fs = require('fs');

const input = 'public/logo.jpg';
const sizes = [48, 96, 128, 256, 512];
(async ()=>{
  for (const size of sizes) {
    await sharp(input)
      .resize(size)
      .webp({ quality: 80 })
      .toFile(`public/logo-${size}.webp`);

    await sharp(input)
      .resize(size)
      .png({ quality: 90 })
      .toFile(`public/logo-${size}.png`);
  }
  console.log('Optimized logos written to public/');
})();
```

Run with: `node scripts/optimize-logo.js` (requires `sharp`).

Then update your HTML `<link rel="icon">` and image `srcset` to use the generated assets.

---

10) GitHub Actions workflow to build and deploy to Vercel (optional)

Create `.github/workflows/deploy-vercel.yml`:

```
name: Deploy to Vercel
on:
  push:
    branches: [ main ]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Use Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18
      - name: Install
        run: npm ci
      - name: Build
        run: npm run build
      - name: Deploy to Vercel
        env:
          VERCEL_TOKEN: ${{ secrets.VERCEL_TOKEN }}
        run: npx vercel --prod --confirm --token $VERCEL_TOKEN
```

Alternative: Use the official Vercel GitHub App for automatic deployments instead of an action.

---

11) Notes about Vercel serverless specifics

- Vercel serverless functions have cold starts and limited execution time. Keep heavy processing (large Puppeteer jobs) small or consider a separate microservice if you need many PDF generations in parallel or complex rendering.
- Keep `firebase-admin` initialization outside the handler (as shown) to reuse warm instances.

---

12) What I implemented here (summary)

- Vercel serverless endpoint examples for AI proxy, revenue aggregation, invoice PDF generation
- Google GenAI provider fallback sample code (read `GENAI_PROVIDER`) — you can switch with `GENAI_PROVIDER=google`
- `sharp` script to generate multi-size WebP/PNG logo assets
- GitHub Actions workflow that runs `npm run build` and uses `npx vercel` to deploy

---

13) Next steps you need to do (quick checklist)

- Add required secrets in Vercel or GitHub: `GOOGLE_API_KEY`, `OPENAI_API_KEY`, `FIREBASE_*`, `VERCEL_TOKEN`, `ADMIN_API_KEY`.
- Replace placeholder model name and request structure for Google GenAI if your account uses a different API path/version.
- Put your invoice template HTML into `admin/invoice-template.html` and ensure images referenced are accessible (public URL or embedded data URI).
- Run `node scripts/optimize-logo.js` locally and commit generated assets to `/public` (or upload optimized images to your CDN and update references).
- Test PDF generation on Vercel (or a staging environment) to confirm chrome/puppeteer binary availability.

If you want, I can also: - Produce exact `package.json` with scripts (build, optimize, start) - Create the actual `api/*` files in the repository (I prepared the full contents here) - Provide a ready-to-paste `invoice-template.html` filled with your branding and line items format

Tell me which of the three you'd like me to add to the repo canvas as separate files and I'll add them now.

Additional files added by assistant

The following files have been prepared and are included in the downloadable ZIP (see chat for link):

- `package.json` with exact scripts for build, dev, optimize, and deploy
- `api/genai-proxy.js` — Vercel serverless AI proxy (Google GenAI + OpenAI fallback)
- `api/revenue-aggregate.js` — serverless revenue aggregator (Firestore)
- `api/invoice-pdf.js` — Puppeteer-based invoice PDF generator
- `admin/invoice-template.html` — finished invoice template branded for Crownshift Logistics (placeholder logo path `/public/logo-256.webp`, brand colors: `#0B5FFF` and `#00B37E`)
- `scripts/optimize-logo.js` — sharp script to produce WebP/PNG variants of the logo
- `.github/workflows/deploy-vercel.yml` — GitHub Action to build and deploy to Vercel
- `README.md` — usage notes and environment variables list

  Note: the ZIP includes these files arranged in the suggested project layout. You still need to add secrets (GOOGLE_API_KEY, OPENAI_API_KEY, FIREBASE_* credentials, ADMIN_API_KEY, VERCEL_TOKEN) in Vercel or GitHub Secrets before deploying.

If you'd like any text or colors changed in the invoice template (I used Crownshift Logistics and brand colors `#0B5FFF` and `#00B37E`), tell me the exact hex codes and I'll update the canvas and ZIP immediately.