### Cum să Adaptezi Aplicația pentru Cloud Run (Pași Recomandați)

Pentru a implementa corect această aplicație pe Cloud Run, trebuie să o transformați într-o aplicație care include un server backend.

#### **Pasul 1: Introduceți un Server Node.js (cu Express)**

Creați un fișier server.js în rădăcina proiectului. Acesta va servi fișierele statice și va acționa ca un proxy pentru API-ul Gemini.

Generated javascript

// server.js

const express = require('express');

const { GoogleGenAI } = require('@google/genai'); // Asigură-te că e în package.json

const path = require('path');

const app = express();

const port = process.env.PORT || 8080;

// Validează existența cheii API

if (!process.env.API\_KEY) {

throw new Error("API\_KEY environment variable not set.");

}

const ai = new GoogleGenAI({ apiKey: process.env.API\_KEY });

app.use(express.json());

// 1. Endpoint proxy pentru API-ul Gemini

app.post('/api/generate', async (req, res) => {

try {

const { model, contents, config } = req.body;

if (!model || !contents) {

return res.status(400).json({ error: 'Model and contents are required.' });

}

// Apel securizat de pe server către Gemini

const response = await ai.models.generateContent({ model, contents, config });

// Returnează doar textul sau răspunsul necesar, nu întregul obiect

res.json({ text: response.text });

} catch (error) {

console.error('Error proxying to Gemini API:', error);

res.status(500).json({ error: 'Failed to generate content from AI.' });

}

});

// Aici adaugi endpoint-uri proxy și pentru generateImages, etc.

// 2. Servește fișierele statice ale aplicației React

// Asumând că un pas de build generează totul în folderul 'dist'

app.use(express.static(path.join(\_\_dirname, 'dist')));

// Pentru orice altă rută, servește index.html (pentru rutare client-side)

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

res.sendFile(path.join(\_\_dirname, 'dist', 'index.html'));

});

app.listen(port, () => {

console.log(`Server listening on port ${port}`);

});

#### **Pasul 2: Modificați Codul Frontend**

În index.tsx, în loc să apelați direct API-ul Gemini, trebuie să apelați noul endpoint de pe serverul dvs.

**Exemplu de modificare în**

Generated typescript

// Înainte (în index.tsx)

// const response = await aiRef.current.models.generateContent({ model: '...', contents: '...' });

// const blueprint = parseJsonFromAiResponse(response.text);

// După (în index.tsx)

const serverResponse = await fetch('/api/generate', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({

model: 'gemini-2.5-flash-preview-04-17',

contents: prompt, // prompt-ul existent

config: { responseMimeType: "application/json" }

})

});

if (!serverResponse.ok) {

const errorBody = await serverResponse.json();

throw new Error(errorBody.error || 'Server request failed');

}

const serverData = await serverResponse.json();

const blueprint = parseJsonFromAiResponse(serverData.text);

#### **Pasul 3: Creați un**

Acest fișier îi spune lui Cloud Run cum să construiască imaginea containerului.

Generated dockerfile

# Dockerfile

# ---- Etapa 1: Build ----

# Folosește o imagine Node pentru a construi aplicația React

FROM node:18-alpine AS builder

WORKDIR /app

# Copiază package.json și instalează dependențele de build

COPY package\*.json ./

RUN npm install

# Copiază tot codul sursă

COPY . .

# Rulează comanda de build (asumând că ai un script "build" în package.json)

# Acest script ar trebui să ruleze `vite build` sau `tsc && vite build`

RUN npm run build

# ---- Etapa 2: Producție ----

# Folosește o imagine Node mai mică pentru rulare

FROM node:18-slim

WORKDIR /app

# Copiază dependențele de producție din etapa de build

COPY package\*.json ./

RUN npm install --omit=dev

# Copiază serverul

COPY server.js .

# Copiază fișierele statice compilate din etapa de build

COPY --from=builder /app/dist ./dist

# Expune portul pe care ascultă serverul

EXPOSE 8080

# Comanda de pornire a serverului

**Pasul 4: Implementarea pe Cloud Run**

1. **Construiți și încărcați imaginea Docker** în Google Artifact Registry sau Container Registry.
2. **Creați un serviciu Cloud Run** folosind imaginea încărcată.
3. **Configurați Secretele:** Setați variabila de mediu API\_KEY în Cloud Run folosind **Secret Manager**. Aceasta este metoda sigură de a oferi cheia API serverului dvs.
4. **Implementați Funcțiile Firebase:** Rulați firebase deploy --only functions din folderul functions pentru a implementa funcțiile cloud separat.

**În concluzie:** Aplicația dvs. este o fundație excelentă, dar necesită adăugarea unui strat de server (BFF - Backend-for-Frontend) și un proces de containerizare (Dockerfile) pentru a fi compatibilă cu Google Cloud Run și pentru a funcționa într-un mod securizat și optimizat pentru producție.