Module 5: Firebase, AI & Deploy

Firebase Auth/Firestore, Gemini API & Deploy

Adrián Catalán

adriancatalan@galileo.edu

Agenda

1. **Project: EventPass Pro**
2. **Firebase Auth, Firestore & Storage**
3. **Using Gemini API**
4. **Deploy to Vercel**
5. **Deep Dive**
6. **Challenge Lab**

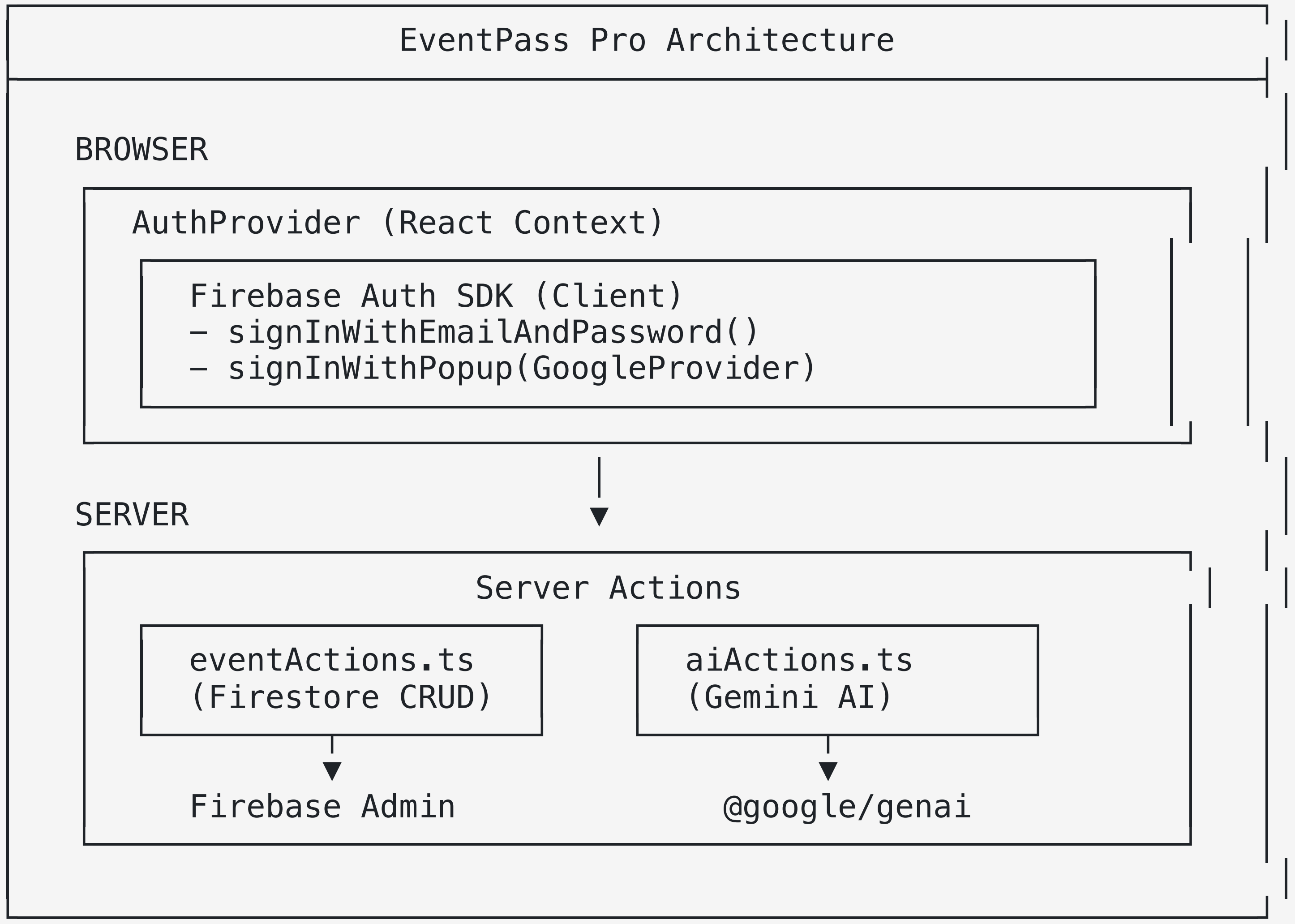
EventPass Pro App

Evolving EventPass with cloud services and AI.

New Features:

1. **Authentication**: Login with email/password or Google.
2. **Cloud Database**: Real-time Firestore database.
3. **AI Generation**: Gemini generates event descriptions.
4. **User Events**: Events associated with authenticated users.

Architecture Overview


EventPass Pro Architecture
BROWSER
AuthProvider (React Context)
Firebase Auth SDK (Client)
- signInWithEmailAndPassword()
- signInWithPopup(GoogleProvider)
SERVER
Server Actions
eventActions.ts
(Firestore CRUD)
aiActions.ts
(Gemini AI)
Firebase Admin
@google/genai

2. Firebase Authentication

Why Firebase Auth?

The Problem (DIY Auth):

- Password hashing, token management, session handling
- Email verification, password reset flows
- Security vulnerabilities
- OAuth integration complexity

The Solution (Firebase Auth):

- Battle-tested authentication
- Multiple providers (Email, Google, Apple, GitHub)
- Automatic token refresh
- Security rules integration

Firebase in Next.js

Two SDKs for different contexts.

SDK	Context	Use For
`firebase`	Client (Browser)	Interactive auth, listeners
`firebase-admin`	Server (Actions/Routes)	Privileged access, verify tokens

```
// Client-side (visible in browser)
NEXT_PUBLIC_FIREBASE_API_KEY=xxx
NEXT_PUBLIC_FIREBASE_PROJECT_ID=xxx

// Server-side (secrets, never exposed)
FIREBASE_ADMIN_PROJECT_ID=xxx
FIREBASE_ADMIN_PRIVATE_KEY=xxx
```

Client SDK Configuration

```
// lib/firebase/config.ts
import { initializeApp, getApps, getApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';

const firebaseConfig = {
    apiKey: process.env.NEXT_PUBLIC_FIREBASE_API_KEY,
    authDomain: process.env.NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN,
    projectId: process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID,
};

// Singleton pattern - don't initialize twice
const app = getApps().length === 0
    ? initializeApp(firebaseConfig)
    : getApp();

export const auth = getAuth(app);
```

Auth Context Pattern

Share auth state across the app with React Context.

```
// contexts/AuthContext.tsx
'use client';

import { createContext, useContext, useEffect, useState } from 'react';
import { onAuthStateChanged, User } from 'firebase/auth';
import { auth } from '@/lib/firebase/config';

interface AuthContextType {
    user: User | null;
    loading: boolean;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export function useAuth() {
    const context = useContext(AuthContext);
    if (!context) throw new Error('useAuth must be within AuthProvider');
    return context;
```

Auth Provider Implementation

```
// contexts/AuthContext.tsx (continued)
export function AuthProvider({ children }: { children: React.ReactNode }) {
    const [user, setUser] = useState<User | null>(null);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        // Firebase listener for auth state changes
        const unsubscribe = onAuthStateChanged(auth, (firebaseUser) => {
            setUser(firebaseUser);
            setLoading(false);
        });

        // Cleanup on unmount
        return () => unsubscribe();
    }, []);

    return (
        <AuthContext.Provider value={{ user, loading }}>
            {children}
        </AuthContext.Provider>
    );
}
```

Sign In Methods

```
// contexts/AuthContext.tsx
import {
    signInWithEmailAndPassword,
    signInWithPopup,
    GoogleAuthProvider,
    createUserWithEmailAndPassword,
    signOut
} from 'firebase/auth';

// Email/Password
const signIn = async (email: string, password: string) => {
    await signInWithEmailAndPassword(auth, email, password);
};

// Google OAuth
const signInWithGoogle = async () => {
    const provider = new GoogleAuthProvider();
    await signInWithPopup(auth, provider);
};

// Sign Out
const signOut = async () => {
    await firebaseSignOut(auth);
};
```

Login Form Component

```tsx
// components/auth/LoginForm.tsx
'use client';

import { useState } from 'react';
import { useAuth } from '@/contexts/AuthContext';

export function LoginForm() {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const { signIn, signInWithGoogle, error } = useAuth();

    const handleSubmit = async (e: React.FormEvent) => {
        e.preventDefault();
        await signIn(email, password);
    };

    return (
        <form onSubmit={handleSubmit}>
            <input type="email" value={email} onChange={...} />
            <input type="password" value={password} onChange={...} />
            {error && <p className="text-red-500">{error}</p>}
            <button type="submit">Sign In</button>
            <button type="button" onClick={signInWithGoogle}>
                Continue with Google
            </button>
        </form>
    );
}
```

User Menu Component

```tsx
// components/auth/UserMenu.tsx
'use client';

import { useAuth } from '@/contexts/AuthContext';
import { Avatar } from '@/components/ui/avatar';
import {
    DropdownMenu,
    DropdownMenuContent,
    DropdownMenuItem,
    DropdownMenuTrigger
} from '@/components/ui/dropdown-menu';

export function UserMenu() {
    const { user, signOut } = useAuth();

    if (!user) return <Link href="/auth">Sign In</Link>;

    return (
        <DropdownMenu>
            <DropdownMenuTrigger>
                <Avatar src={user.photoURL} alt={user.displayName} />
            </DropdownMenuTrigger>
            <DropdownMenuContent>
                <DropdownMenuItem>{user.email}</DropdownMenuItem>
                <DropdownMenuItem onClick={signOut}>Sign Out</DropdownMenuItem>
            </DropdownMenuContent>
        </DropdownMenu>
    );
}
```

3. Cloud Firestore

What is Firestore?

A NoSQL document database with real-time sync.

```
┌─────────────────────────────────────────────────────────────────┐
│                      Firestore Data Model                       │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  events (Collection)                                            │
│    │                                                            │
│    ├── abc123 (Document)                                        │
│    │     ├── title: "Web Conference"                            │
│    │     ├── description: "..."                                 │
│    │     ├── organizerId: "user_xyz"                            │
│    │     ├── date: Timestamp                                    │
│    │     └── createdAt: Timestamp                               │
│    │                                                            │
│    └── def456 (Document)                                        │
│          └── ...                                                │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

Firebase Admin SDK Setup

```
// lib/firebase/admin.ts
import { initializeApp, cert, getApps, getApp } from 'firebase-admin/app';
import { getFirestore } from 'firebase-admin/firestore';

const serviceAccount = {
    projectId: process.env.FIREBASE_ADMIN_PROJECT_ID,
    clientEmail: process.env.FIREBASE_ADMIN_CLIENT_EMAIL,
    privateKey: process.env.FIREBASE_ADMIN_PRIVATE_KEY?.replace(/\\n/g, '\n'),
};

const app = getApps().length === 0
    ? initializeApp({ credential: cert(serviceAccount) })
    : getApp();

export const adminDb = getFirestore(app);
```

Firestore Data Layer

```
// lib/firebase/firestore.ts
import { adminDb } from './admin';
import { Timestamp } from 'firebase-admin/firestore';

const EVENTS_COLLECTION = 'events';

export async function getEvents(filters?: EventFilters): Promise<Event[]> {
    let query = adminDb.collection(EVENTS_COLLECTION)
        .orderBy('date', 'asc');

    if (filters?.status) {
        query = query.where('status', '==', filters.status);
    }

    if (filters?.category) {
        query = query.where('category', '==', filters.category);
    }

    const snapshot = await query.get();
    return snapshot.docs.map(doc => docToEvent(doc.id, doc.data()));
}
```

CRUD Operations

```
// lib/firebase/firestore.ts

// CREATE
export async function createEvent(data: CreateEventInput): Promise<Event> {
    const eventData = {
        ...data,
        registeredCount: 0,
        createdAt: Timestamp.now(),
        updatedAt: Timestamp.now(),
        date: Timestamp.fromDate(new Date(data.date)),
    };

    const docRef = await adminDb.collection(EVENTS_COLLECTION).add(eventData);
    const newDoc = await docRef.get();
    return docToEvent(docRef.id, newDoc.data()!);
}

// READ BY ID
export async function getEventById(id: string): Promise<Event | null> {
    const docSnap = await adminDb.collection(EVENTS_COLLECTION).doc(id).get();
    return docSnap.exists ? docToEvent(docSnap.id, docSnap.data()!) : null;
}
```

Update & Delete

```
// lib/firebase/firestore.ts

// UPDATE
export async function updateEvent(
    id: string,
    data: Partial<CreateEventInput>
): Promise<Event | null> {
    const docRef = adminDb.collection(EVENTS_COLLECTION).doc(id);

    await docRef.update({
        ...data,
        updatedAt: Timestamp.now(),
    });

    const updatedDoc = await docRef.get();
    return docToEvent(id, updatedDoc.data()!);
}

// DELETE
export async function deleteEvent(id: string): Promise<boolean> {
    await adminDb.collection(EVENTS_COLLECTION).doc(id).delete();
    return true;
}
```

Transactions for Consistency

```
// lib/firebase/firestore.ts
export async function registerForEvent(eventId: string): Promise<Event | null> {
    const docRef = adminDb.collection(EVENTS_COLLECTION).doc(eventId);

    const result = await adminDb.runTransaction(async (transaction) => {
        const docSnap = await transaction.get(docRef);
        if (!docSnap.exists) return null;

        const data = docSnap.data()!;
        if (data.registeredCount >= data.capacity) return null;
        if (data.status !== 'published') return null;

        transaction.update(docRef, {
            registeredCount: data.registeredCount + 1,
            updatedAt: Timestamp.now(),
        });

        return { ...data, registeredCount: data.registeredCount + 1 };
    });

    return result ? docToEvent(eventId, result) : null;
}
```

Firestore Security Rules

```
// firestore.rules
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /events/{eventId} {
      // Anyone can read published events
      allow read: if resource.data.status == 'published';

      // Only owner can create (must set own organizerId)
      allow create: if request.auth != null
                    && request.auth.uid == request.resource.data.organizerId;

      // Only owner can update/delete
      allow update, delete: if request.auth != null
                            && request.auth.uid == resource.data.organizerId;
    }
  }
}
```

4. Gemini AI Integration

Gemini for Content Generation

Use AI to generate event descriptions automatically via Server Actions.

```
// lib/gemini.ts
import { GoogleGenAI } from '@google/genai';

const genAI = new GoogleGenAI({ apiKey: process.env.GEMINI_API_KEY || '' });

export const GEMINI_MODELS = {
    TEXT: 'gemini-3-flash-preview',
    IMAGE: 'gemini-3-pro-image-preview',
} as const;

export const getGeminiClient = () => genAI;
```

```
// actions/aiActions.ts - Server Action
'use server';

export async function generateEventDetailsAction(title: string) {
    const client = getGeminiClient();
```

Server Actions for AI

Server Actions keep API keys secure while providing a simple interface.

```typescript
// actions/aiActions.ts
'use server';

import { getGeminiClient, GEMINI_MODELS } from '@/lib/gemini';

export async function generateEventDetailsAction(title: string) {
    if (!process.env.GEMINI_API_KEY) {
        return { success: false, error: 'GEMINI_API_KEY not configured' };
    }

    const client = getGeminiClient();
    const prompt = `Generate event details for: "${title}"...`;

    const result = await client.models.generateContent({
        model: GEMINI_MODELS.TEXT,
        contents: [{ role: 'user', parts: [{ text: prompt }] }],
        config: { responseMimeType: 'application/json' }
    });

    const data = JSON.parse(result.text);
    return { success: true, data };
```

Using AI in EventForm

Call Server Actions directly from Client Components.

```
// components/EventForm.tsx
'use client';

import { useState } from 'react';
import { generateEventDetailsAction } from '@/actions/aiActions';

export function EventForm() {
    const [isGenerating, setIsGenerating] = useState(false);

    const handleGenerateWithAI = async () => {
        const title = watch('title');
        if (!title || title.length < 3) return;

        setIsGenerating(true);
        try {
            const result = await generateEventDetailsAction(title);
            if (result.success && result.data) {
                setValue('description', result.data.description);
                setValue('category', result.data.category);
                setValue('tags', result.data.tags.join(', '));
            }
        } finally {
            setIsGenerating(false);
        }
    };

    return (
        <Button onClick={handleGenerateWithAI} disabled={isGenerating}>
            {isGenerating ? 'Generating...' : '✨ Generate with AI'}
        </Button>
```

AI Image Generation

Generate event posters with Gemini 3 Pro Image.

```
// actions/aiActions.ts
'use server';

export async function generateEventPosterAction(prompt: string, eventId?: string) {
    const client = getGeminiClient();

    const result = await client.models.generateContent({
        model: GEMINI_MODELS.IMAGE, // 'gemini-3-pro-image-preview'
        contents: [{
            role: 'user',
            parts: [{ text: `Create event poster: ${prompt}. 16:9, professional.` }]
        }]
    });

    // Extract base64 image from response
    const part = result.candidates?.[0]?.content?.parts?.[0];
    const base64Image = part?.inlineData?.data;

    // Upload to Firebase Storage
    const imageUrl = await uploadPosterToStorage(eventId, base64Image);
    return { success: true, imageUrl };
```

4. Deploy to Vercel

4.1 Option A: Vercel CLI

Deploy directly from your terminal.

```
# 1. Install CLI
npm i -g vercel

# 2. Login
vercel login

# 3. Deploy (Follow the prompts)
vercel
```

Common Prompts:

- Set up and deploy? [Y/n] **y**
- Which scope? **(Select your account)**
- Link to existing project? [y/N] **n**
- Project name? **event-pass-pro**

Environment Variables with CLI

You must set environment variables for the production build.

```
# Set secrets via CLI
vercel env add NEXT_PUBLIC_FIREBASE_API_KEY
vercel env add FIREBASE_ADMIN_PRIVATE_KEY

# Push local env vars (Development)
vercel env pull .env.development.local
```

> **Pro Tip:** Vercel automatically detects Next.js projects and configures build settings (`next build`).

4.2 Option B: GitHub Integration

Continuous Deployment (CD) - The recommended way.

1. **Push your code** to a GitHub repository.
2. **Go to Vercel Dashboard** > "Add New..." > "Project".
3. **Import** your `event-pass-pro` repository.
4. **Configure Environment Variables**:
   - Copy/Paste all values from `.env.local` into the "Environment Variables" section.
5. **Click Deploy**.

Benefits:

- **Automatic Deploys:** Pushing to `main` deploys to production.
- **Preview URLs:** Pull Requests get their own live URL.

5. Deep Dive

1. Client vs Admin SDK

Understanding when to use each SDK.

```
┌─────────────────────────────────────────────────────────────────┐
│              Firebase SDK Decision Tree                         │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  Where is the code running?                                     │
│                                                                 │
│  ├── BROWSER (Client Components)                                │
│  │   └── Use `firebase` (client SDK)                            │
│  │       - Auth UI (login forms, OAuth popups)                  │
│  │       - Real-time listeners                                  │
│  │       - User-initiated actions                               │
│  │                                                              │
│  └── SERVER (Server Actions, API Routes)                        │
│      └── Use `firebase-admin`                                   │
│          - Database operations                                  │
│          - Verify tokens                                        │
│          - Privileged operations                                │
│                                                                 │
```

2. Build Time Safety

Firebase can fail during Next.js build if not configured.

```
// lib/firebase/config.ts

function hasFirebaseCredentials(): boolean {
    return Boolean(
        process.env.NEXT_PUBLIC_FIREBASE_API_KEY &&
        process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID
    );
}

// Lazy initialization
export function getFirebaseAuth(): Auth | null {
    if (!hasFirebaseCredentials()) {
        if (typeof window !== 'undefined') {
            console.warn('Firebase not configured');
        }
        return null;
    }

    return getAuth(getFirebaseApp());
```

3. onAuthStateChanged Lifecycle

How Firebase auth state flows through the app.

```
┌─────────────────────────────────────────────────────────────────┐
│                    Auth State Flow                               │
├─────────────────────────────────────────────────────────────────┤
│                                                                  │
│  1. App Loads                                                    │
│     └── AuthProvider mounts                                      │
│         └── onAuthStateChanged registers listener                │
│                                                                  │
│  2. User Signs In                                                │
│     └── signInWithEmailAndPassword() succeeds                    │
│         └── Firebase triggers listener                           │
│             └── setUser(firebaseUser)                            │
│                 └── Context updates → Components re-render       │
│                                                                  │
│  3. Token Refresh (automatic, ~1 hour)                           │
│     └── Firebase handles silently                                │
│                                                                  │
│  4. User Signs Out                                               │
│     └── signOut() called                                         │
│         └── Listener fires with null                             │
│             └── setUser(null)                                    │
```

4. Firestore Timestamps

Converting between Firestore and JavaScript dates.

```
import { Timestamp } from 'firebase-admin/firestore';

// Writing to Firestore
const eventData = {
    title: 'Conference',
    date: Timestamp.fromDate(new Date('2024-12-15')),
    createdAt: Timestamp.now(),
};

// Reading from Firestore
function docToEvent(id: string, data: DocumentData): Event {
    return {
        id,
        title: data.title,
        // Convert Timestamp to ISO string
        date: data.date instanceof Timestamp
            ? data.date.toDate().toISOString()
            : data.date,
        createdAt: data.createdAt.toDate().toISOString(),
```

5. Error Translation

Map Firebase error codes to user-friendly messages.

```
// contexts/AuthContext.tsx
function translateFirebaseError(message: string): string {
    const translations: Record<string, string> = {
        'auth/email-already-in-use': 'This email is already registered',
        'auth/invalid-email': 'Invalid email address',
        'auth/weak-password': 'Password must be at least 6 characters',
        'auth/user-not-found': 'No account exists with this email',
        'auth/wrong-password': 'Incorrect password',
        'auth/invalid-credential': 'Invalid credentials',
        'auth/popup-closed-by-user': 'Authentication window was closed',
    };

    for (const [code, translation] of Object.entries(translations)) {
        if (message.includes(code)) return translation;
    }
    return message;
}
```

6. Challenge Lab

Practice & Application

Part 1: My Events Dashboard

Context:

Authenticated users need to see only their events and manage them (edit/delete).

Your Task:

Create a "My Events" page that:

- Only accessible to authenticated users
- Shows events where organizerId matches current user
- Allows editing and deleting own events
- Redirects to login if not authenticated

Files to Create/Modify:

- `app/my-events/page.tsx`
- `actions/eventActions.ts` (add delete action)

Part 1: Definition of Done

Criteria	Description
Protected route	Redirects to /auth if not logged in
User filter	Only shows events where organizerId === user.uid
Edit button	Navigates to edit form with pre-filled data
Delete button	Confirms and deletes event
Authorization	Server validates user owns event before delete
Empty state	"No events yet" message with CTA
Loading state	Shows skeleton while fetching

Part 2: Enhanced AI Generation

Context:

The AI description generator could be more powerful with structured output and multiple suggestions.

Your Task:

Enhance the AI feature to:

- Generate 3 description variants
- Let user pick their favorite
- Support different tones (formal, casual, exciting)
- Show loading progress

Files to Modify:

- `lib/gemini.ts`

Part 2: Definition of Done

Criteria	Description
Multiple variants	Returns array of 3 descriptions
Tone selector	Dropdown to choose formal/casual/exciting
Selection UI	Cards to preview and select variant
Apply button	Selected description fills textarea
Progress indicator	Shows "Generating..." with spinner
Error handling	Shows error message if generation fails
Regenerate	Button to generate new variants

Resources & Wrap-up

Resources

Firebase Auth

- Firebase Auth Web Docs
- Auth State Persistence
- Google Sign-In

Cloud Firestore

- Firestore Web Docs
- Security Rules
- Data Modeling
- Transactions

Gemini AI

- Gemini API Docs

Recommended Articles

Firebase in Next.js

- Firebase with Next.js 13+ - Firebase Docs
- Server-Side Auth with Firebase - Firebase Blog
- Firestore Security Rules Guide - Firebase Docs

Generative AI

- Gemini API Quickstart - Google AI
- Prompt Design Strategies - Google AI
- Building AI-Powered Apps - Google Developers