perplexity

You will act as a development team, which consists of a frontend developer, UI/UX designer and a backend developer to build this software engineering firm website using the specifications attached. Replace the Supabase components with the Google Firestore.

I'll provide comprehensive documentation for building the TETRIX software engineering firm website using Google Firestore instead of Supabase, structured for your development team of frontend developer, UI/UX designer, and backend developer.

Project Architecture Overview

TETRIX will be built using Astro (Islands architecture) with Tailwind CSS, Lucide React icons, and Google Firestore for authentication and data management. The site serves as a SaaS marketing platform with user authentication, CRM dashboard, and ticket management system.

1. UI/UX Design Documentation

Brand & Visual Identity

- **Logo**: Centralized on static pages, top-left in navigation; geometric and modern design reflecting "TETRIX"
- **Color Palette**: Gradient background from #FFF8E7 to #E8F5E9; text #1E1E1E; CTA buttons #2A2A2A; navigation background rgba(42,42,42,0.8)[1]
- **Typography**: Sans-serif, 2.5rem hero titles, 1.25rem body text, consistent 12px base spacing[1]
- **Imagery**: Stock images from `picsum.photos` with configured remotePatterns[1]

Layout Structure

- **Header**: Full-width 60px height with logo (left), main navigation (center), contact info (top), CTA button (right)[1]
- **Navigation**: Home, About, Solutions, Services, Contact; sticky desktop, collapsible mobile[1]
- **Grid System**: 12-column layout, max-width 1200px, responsive padding (24px mobile, 48px desktop)[1]
- **Hero Section**: 2-column layout on tablet/desktop with partner logos (Framer, AWS, NVIDIA)[1]

Component Library

- **Buttons**: Primary (dark bg, light text), Secondary (light bg, dark text) with Lucide icons[1]
- **Forms**: Inline validation, floating error messages, progressive multi-step flows[1]
- **Cards**: Service grids, solution showcases, testimonials, partner displays[1]
- **Interactive Elements**: Click-to-call, email forms, live chat, support ticket submission[1]

2. Frontend Development Guide

Project Structure

```
src/
├── components/
│   ├── layout/ (Header, Navigation, Footer)
│   ├── features/ (Hero, Partners, CTASection)
│   ├── auth/ (AuthForm, ProtectedRoute)
│   └── shared/ (Buttons, FormControls)
├── pages/
│   ├── index.astro (Home)
│   ├── about.astro
│   ├── solutions.astro
│   ├── services.astro
│   ├── contact.astro
│   ├── contact-information-phone.astro
│   ├── email.astro
│   ├── address.astro
│   ├── onboarding.astro (SignUp/SignIn)
│   └── dashboard/ (CRM dashboard, protected)
├── lib/ (firebase.ts, firestore.ts)
├── styles/
└── utils/
```

Key Components Implementation

- **Layout**: Root layout wraps all pages with navigation and global styles[1]
- **Navigation**: Responsive with hover states and Lucide icons, collapsible on mobile[1]
- **Hero Section**: Large heading, subheading, partner logos, dual CTA buttons[1]
- **Protected Routes**: Middleware guards `/dashboard` routes, redirects unauthenticated users[1]

Styling Guidelines

- Tailwind CSS utility classes exclusively (no other UI libraries except Lucide React)[1]
- Responsive breakpoints: mobile (320px), tablet (768px), desktop (1024px), wide (1440px)[1]
- Image optimization using Astro's `<Image>` component with picsum.photos remotePatterns[1]

3. Backend & Authentication (Google Firestore)

Firebase/Firestore Setup

Create a Firebase project and enable Firestore database in production mode[2]. Configure authentication providers (Email/Password, Google OAuth) in the Firebase console[3].

```
// src/lib/firebase.ts
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: import.meta.env.PUBLIC_FIREBASE_API_KEY,
  authDomain: import.meta.env.PUBLIC_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.PUBLIC_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.PUBLIC_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.PUBLIC_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.PUBLIC_FIREBASE_APP_ID
};

const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export const db = getFirestore(app);
```

Environment Variables

```
PUBLIC_FIREBASE_API_KEY=your-api-key
PUBLIC_FIREBASE_AUTH_DOMAIN=your-project.firebaseapp.com
PUBLIC_FIREBASE_PROJECT_ID=your-project-id
PUBLIC_FIREBASE_STORAGE_BUCKET=your-project.appspot.com
PUBLIC_FIREBASE_MESSAGING_SENDER_ID=your-sender-id
PUBLIC_FIREBASE_APP_ID=your-app-id
```

Authentication Implementation

```
// src/lib/auth.ts
import { createUserWithEmailAndPassword, signInWithEmailAndPassword, signOut } from 'fire
import { auth } from './firebase';

export const firebaseCreateUser = async (email: string, password: string) => {
  try {
    const userCredential = await createUserWithEmailAndPassword(auth, email, password);
    return { user: userCredential.user, error: null };
  } catch (error) {
    return { user: null, error: error.message };
  }
};

export const firebaseSignIn = async (email: string, password: string) => {
  try {
```

```
    const userCredential = await signInWithEmailAndPassword(auth, email, password);
    return { user: userCredential.user, error: null };
  } catch (error) {
    return { user: null, error: error.message };
  }
};

export const firebaseSignOut = async () => {
  try {
    await signOut(auth);
    return { error: null };
  } catch (error) {
    return { error: error.message };
  }
};
```

Firestore Data Structure

```
// Collections Structure
interface User {
  id: string;
  email: string;
  displayName?: string;
  createdAt: Date;
  role: 'user' | 'admin';
}

interface Ticket {
  id: string;
  userId: string;
  subject: string;
  description: string;
  status: 'open' | 'in-progress' | 'closed';
  priority: 'low' | 'medium' | 'high';
  createdAt: Date;
  updatedAt: Date;
}
```

Firestore Operations

```
// src/lib/firestore.ts
import { collection, doc, addDoc, getDoc, getDocs, updateDoc, query, where, orderBy } fro
import { db } from './firebase';

// Create user profile
export const createUserProfile = async (userId: string, userData: Partial<User>) => {
  try {
    await setDoc(doc(db, 'users', userId), {
      ...userData,
      createdAt: new Date(),
      role: 'user'
    });
    return { error: null };
```

```
  } catch (error) {
    return { error: error.message };
  }
};

// Create ticket
export const createTicket = async (ticketData: Omit<Ticket, 'id' | 'createdAt' | 'updated
  try {
    const docRef = await addDoc(collection(db, 'tickets'), {
      ...ticketData,
      createdAt: new Date(),
      updatedAt: new Date()
    });
    return { ticketId: docRef.id, error: null };
  } catch (error) {
    return { ticketId: null, error: error.message };
  }
};

// Get user tickets
export const getUserTickets = async (userId: string) => {
  try {
    const q = query(
      collection(db, 'tickets'),
      where('userId', '==', userId),
      orderBy('createdAt', 'desc')
    );
    const querySnapshot = await getDocs(q);
    const tickets = querySnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
    return { tickets, error: null };
  } catch (error) {
    return { tickets: [], error: error.message };
  }
};
```

4. Authentication Components

Onboarding Page Component

```
// src/components/auth/AuthForm.tsx
'use client';
import { useState } from 'react';
import { useRouter } from 'next/router';
import { firebaseCreateUser, firebaseSignIn } from '@/lib/auth';
import { createUserProfile } from '@/lib/firestore';

export default function AuthForm() {
  const [isSignUp, setIsSignUp] = useState(false);
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [loading, setLoading] = useState(false);
```

```
  const [error, setError] = useState('');

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setLoading(true);
    setError('');

    if (isSignUp) {
      const { user, error } = await firebaseCreateUser(email, password);
      if (user) {
        await createUserProfile(user.uid, { email, displayName: email });
        // Redirect to dashboard
      } else {
        setError(error);
      }
    } else {
      const { user, error } = await firebaseSignIn(email, password);
      if (user) {
        // Redirect to dashboard
      } else {
        setError(error);
      }
    }
    setLoading(false);
  };

  return (
    <div className="max-w-md mx-auto p-6 bg-white shadow-lg rounded-lg">
      <h2 className="text-2xl font-bold mb-6 text-center">
        {isSignUp ? 'Create Account' : 'Sign In'}
      </h2>

      <form onSubmit={handleSubmit} className="space-y-4">
        <div>
          <label className="block text-sm font-medium mb-2">Email</label>
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="w-full p-3 border border-gray-300 rounded-md focus:outline-none fo
            required
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-2">Password</label>
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="w-full p-3 border border-gray-300 rounded-md focus:outline-none fo
            required
          />
        </div>

        {error && (
```

```
          <div className="text-red-600 text-sm">{error}</div>
        )}

        <button
          type="submit"
          disabled={loading}
          className="w-full py-3 bg-gray-800 text-white rounded-md hover:bg-gray-700 disa
        >
          {loading ? 'Processing...' : (isSignUp ? 'Sign Up' : 'Sign In')}
        </button>
      </form>

      <div className="mt-4 text-center">
        <button
          onClick={() => setIsSignUp(!isSignUp)}
          className="text-gray-600 hover:text-gray-800"
        >
          {isSignUp ? 'Already have an account? Sign In' : 'Need an account? Sign Up'}
        </button>
      </div>
    </div>
  );
}
```

5. Protected Routes & Middleware

```
// src/middleware.ts
import type { MiddlewareHandler } from 'astro';
import { auth } from './lib/firebase';

export const onRequest: MiddlewareHandler = async ({ request, redirect }) => {
  // Check if user is authenticated for protected routes
  if (request.url.pathname.startsWith('/dashboard')) {
    const user = auth.currentUser;
    if (!user) {
      return redirect('/onboarding');
    }
  }
};
```

6. CRM Dashboard Implementation

Dashboard Layout

```
// src/components/dashboard/DashboardLayout.tsx
'use client';
import { useState, useEffect } from 'react';
import { auth } from '@/lib/firebase';
import { getUserTickets } from '@/lib/firestore';
import { User, LogOut, Plus } from 'lucide-react';

export default function DashboardLayout({ children }) {
```

```
  const [user, setUser] = useState(null);
  const [tickets, setTickets] = useState([]);

  useEffect(() => {
    const unsubscribe = auth.onAuthStateChanged(async (user) => {
      if (user) {
        setUser(user);
        const { tickets } = await getUserTickets(user.uid);
        setTickets(tickets);
      }
    });
    return () => unsubscribe();
  }, []);

  return (
    <div className="min-h-screen bg-gray-50">
      {/* Header */}
      <header className="bg-white shadow-sm border-b">
        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
          <div className="flex justify-between items-center h-16">
            <h1 className="text-xl font-semibold">TETRIX Dashboard</h1>
            <div className="flex items-center space-x-4">
              <span className="text-sm text-gray-600">{user?.email}</span>
              <button className="p-2 text-gray-400 hover:text-gray-600">
                <LogOut size={20} />
              </button>
            </div>
          </div>
        </div>
      </header>

      {/* Main Content */}
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
        {/* KPI Cards */}
        <div className="grid grid-cols-1 md:grid-cols-3 gap-6 mb-8">
          <div className="bg-white p-6 rounded-lg shadow">
            <h3 className="text-lg font-medium mb-2">Open Tickets</h3>
            <p className="text-3xl font-bold text-blue-600">
              {tickets.filter(t => t.status === 'open').length}
            </p>
          </div>
          <div className="bg-white p-6 rounded-lg shadow">
            <h3 className="text-lg font-medium mb-2">In Progress</h3>
            <p className="text-3xl font-bold text-yellow-600">
              {tickets.filter(t => t.status === 'in-progress').length}
            </p>
          </div>
          <div className="bg-white p-6 rounded-lg shadow">
            <h3 className="text-lg font-medium mb-2">Resolved</h3>
            <p className="text-3xl font-bold text-green-600">
              {tickets.filter(t => t.status === 'closed').length}
            </p>
          </div>
        </div>

        {/* New Ticket Button */}
```

```
        <div className="mb-6">
          <button className="bg-gray-800 text-white px-4 py-2 rounded-md hover:bg-gray-70
            <Plus size={20} />
            <span>New Ticket</span>
          </button>
        </div>

        {children}
      </div>
    </div>
  );
}
```

7. Route Specifications

Route	Purpose	Layout	Authentication
/	Landing page with hero, services, partners	MainLayout	Public
/about	Company info, team, mission	ContentLayout	Public
/solutions	Product solutions, case studies	ContentLayout	Public
/services	Service catalog, booking forms	ContentLayout	Public
/contact	Universal contact hub	ContactLayout	Public
/contact-information-phone	Phone contact details	ContactLayout	Public
/email	Email contact options	ContactLayout	Public
/address	Physical location, map	ContactLayout	Public
/onboarding	SignUp/SignIn forms	AuthLayout	Public
/dashboard	CRM dashboard, tickets	DashboardLayout	Protected

8. Implementation Timeline

Week 1: Project setup, Firebase configuration, core layouts
Week 2: Authentication system, onboarding flow
Week 3: Public pages (home, about, solutions, services)
Week 4: Contact hub, dashboard implementation
Week 5: Ticket system, testing, optimization, deployment

9. Security & Best Practices

- **Firestore Security Rules**: Implement rules to restrict user access to their own data[4]
- **Authentication**: Use Firebase ID tokens for client requests[4]
- **Data Validation**: Validate all inputs before Firestore operations
- **Error Handling**: Implement comprehensive error handling for all Firebase operations

- **Performance**: Use Firestore queries efficiently with proper indexing[5]

This comprehensive guide provides your development team with all necessary specifications to build the TETRIX website using Google Firestore instead of Supabase, maintaining the same functionality while leveraging Firebase's ecosystem for authentication and data management.

⁂

1. Tetrix.txt
2. https://github.com/astronomer/ask-astro/blob/main/docs/api/google_firestore.md
3. https://docs.astro.build/en/guides/backend/google-firebase/
4. https://firebase.google.com/docs/firestore/use-rest-api
5. https://firebase.google.com/docs/firestore