**📋 وضعیت فعلی پروژه**

الان یک اپلیکیشن داریم که:

* ۱۱۱ مرحله (Stage) برای راهنمایی کارآفرینان داره
* با هوش مصنوعی Gemini صحبت می‌کنه
* کاربر لاگین می‌کنه و مرحله به مرحله پیش میره
* مشکلش اینه که هوش مصنوعی خیلی حرف می‌زنه!

**مثال مشکل:** وقتی از کاربر می‌خواد اسم استارتاپش رو انتخاب کنه، هوش مصنوعی یک پاراگراف توضیح می‌ده + چند تا پیشنهاد می‌ده. کاربر فقط می‌خواد پیشنهادها رو ببینه بدون هیچ حرف اضافه‌ای.

**🎯 هدف نهایی**

می‌خوایم برسیم به جایی که:

1. **هر مرحله رفتار مخصوص به خودش داشته باشه**
   * مرحله انتخاب نام → فقط لیست گزینه‌ها
   * مرحله نوشتن چکیده → فقط متن کامل
   * مرحله تحلیل بازار → فقط آمار و ارقام
   * مرحله اهداف کسب‌وکار → لیست ساختار یافته
2. **چند زبان پشتیبانی بشه**
   * انگلیسی، فارسی، عربی، فرانسه، اسپانیایی، آلمانی
   * محتوای استاتیک (دکمه‌ها، منوها) از فایل بیاد
   * محتوای داینامیک (توضیحات مراحل، پاسخ هوش مصنوعی) از دیتابیس یا API بیاد
3. **سیستم ترجمه هوشمند باشه**
   * اول توی حافظه نگاه کنه (سریع‌ترین)
   * بعد توی دیتابیس نگاه کنه (سریع)
   * در آخر از هوش مصنوعی بخواد ترجمه کنه (کندتر)
4. **کاربر راحت‌تر کار کنه**
   * ببینه هر مرحله چی ازش می‌خواد
   * پاسخ هوش مصنوعی رو ببینه (بدون حرف اضافه)
   * بتونه قبول یا رد کنه
   * اگر چیزی خودش نوشته بود، به هوش مصنوعی فرستاده بشه

**📦 چهار فاز اصلی**

**فاز ۰: سیستم ترجمه (پایه و اساس)**

**چرا اول؟** چون اگر این سیستم آماده باشه، بقیه کارها خودکار ترجمه می‌شن.

**چی می‌خواد بشه:**

1. **یک جدول توی دیتابیس Supabase**
   * نام جدول: stage\_translations
   * ستون‌ها: شناسه مرحله، کد زبان، عنوان ترجمه شده، توضیحات ترجمه شده، تاریخ ایجاد
2. **یک سرویس ترجمه در TypeScript**
   * تابعی که بگیم "این متن رو به فارسی ترجمه کن"
   * اول توی حافظه برنامه نگاه کنه (یک آبجکت ساده)
   * اگر نبود، توی دیتابیس نگاه کنه
   * اگر اونجا هم نبود، از Gemini بخواد ترجمه کنه
   * ترجمه جدید رو ذخیره کنه (توی حافظه + دیتابیس)
3. **یک Context برای کل برنامه**
   * زبان فعلی رو نگه داره (مثلاً fa یا en)
   * تابع تغییر زبان داشته باشه
   * تابع گرفتن ترجمه داشته باشه
4. **یک کامپوننت برای انتخاب زبان**
   * دراپ‌داون یا لیست پرچم‌های کشورها
   * وقتی کاربر کلیک کنه، زبان تغییر کنه
   * کل صفحه دوباره رندر بشه با زبان جدید
5. **یک فایل برای متن‌های استاتیک**
   * مثل "ورود"، "خروج"، "بعدی"، "قبلی"
   * برای هر زبان، یک آبجکت جداگانه
   * این متن‌ها توی کد هارد‌کد هستن (نیازی به API نیست)

**چی اتفاق می‌افته:**

* اولین بار که کاربر زبان فارسی رو انتخاب می‌کنه، برنامه از Gemini می‌خواد همه‌چیز رو ترجمه کنه
* بار دوم که اون کاربر یا کاربر دیگه‌ای فارسی رو انتخاب می‌کنه، از دیتابیس میاد (خیلی سریع‌تر)
* توی همون سشن، از حافظه میاد (فوق‌العاده سریع)

**فاز ۱: بررسی و اصلاح مراحل**

**چرا مهمه؟** چون اگر مراحل درست نباشن، پرامپت نویسی فایده نداره.

**چی می‌خواد بشه:**

1. **بررسی تک‌تک ۱۱۱ مرحله**
   * آیا ترتیب منطقیه؟
   * آیا مرحله‌ای اضافیه؟
   * آیا مرحله‌ای کم داریم؟
   * آیا نام هر مرحله واضحه؟
2. **تصمیم‌گیری برای هر مرحله**
   * نگه داشتن بدون تغییر
   * حذف کردن (اگر غیر ضروریه)
   * اضافه کردن مرحله جدید (اگر چیزی کم داریم)
   * تغییر نام یا ترتیب
3. **مشخص کردن نوع خروجی هر مرحله**
   * selection\_list → لیستی از گزینه‌ها برای انتخاب (مثل انتخاب نام)
   * full\_text → یک متن کامل (مثل نوشتن چکیده)
   * structured\_list → لیست ساختار یافته (مثل اهداف با عنوان + توضیح)
   * analysis → تحلیل یا بررسی (مثل تحلیل رقبا)
4. **ثبت تغییرات**
   * در یک فایل Excel یا Google Sheets
   * یا در یک فایل Markdown به نام STAGE\_CHANGELOG.md
   * شامل: شماره مرحله، نام قدیم، نام جدید، دلیل تغییر، نوع خروجی
5. **بروزرسانی فایل‌های TypeScript**
   * فایل types.ts → enum مراحل
   * فایل useStartupJourney.ts → لیست مراحل به ترتیب

**چی اتفاق می‌افته:**

* یک نقشه کامل از مراحل درست می‌شه
* می‌دونیم دقیقاً چند مرحله داریم (شاید ۱۱۱، شاید ۱۲۰، شاید ۱۰۵)
* می‌دونیم هر مرحله چه نوع خروجی‌ای باید بده

**فاز ۲: طراحی پرامپت‌های اختصاصی**

**چرا مهمه؟** این قلب اصلی راه‌حله! اینجاست که مشکل "حرف زیاد زدن" هوش مصنوعی حل می‌شه.

**چی می‌خواد بشه:**

1. **نوشتن یک پرامپت برای هر مرحله**
   * اگر ۱۱۴ مرحله داریم = ۱۱۴ پرامپت
   * هر پرامپت شامل:
     + **زمینه (Context):** این مرحله توی کجای فرایند کارآفرینیه؟
     + **ورودی‌ها:** چه اطلاعاتی از کاربر یا مراحل قبل داریم؟
     + **خروجی مورد انتظار:** دقیقاً چه فرمتی می‌خوایم؟
     + **محدودیت‌ها:** چی نباید باشه؟ (مثلاً "بدون سلام، بدون توضیح اضافه")
2. **تعریف فرمت‌های خروجی**

**برای selection\_list:**

1. گزینه اول

2. گزینه دوم

3. گزینه سوم

(بدون هیچ توضیح قبل یا بعد)

**برای full\_text:**

متن کامل و روان بدون نقطه چین یا شماره‌گذاری

**برای structured\_list:**

### عنوان اول

توضیح مختصر

### عنوان دوم

توضیح مختصر

**برای analysis:**

- نکته اول: توضیح

- نکته دوم: توضیح

1. **ایجاد یک فایل پیکربندی**
   * نام فایل: stagePrompts.ts
   * ساختار: یک آبجکت بزرگ که کلیدش نام مرحله‌ست
   * برای هر مرحله:
     + guidanceText: راهنمایی برای کاربر (چی باید بنویسه؟)
     + systemPrompt: پرامپت سیستمی برای Gemini
     + outputType: نوع خروجی
     + maxTokens: حداکثر طول پاسخ
     + temperature: خلاقیت هوش مصنوعی (۰ تا ۱)
2. **تست و اصلاح پرامپت‌ها**
   * اجرای چند بار با ورودی‌های مختلف
   * بررسی اینکه آیا هوش مصنوعی قانون‌ها رو رعایت می‌کنه
   * اگر باز هم حرف اضافه زد، پرامپت رو سخت‌گیرانه‌تر کردن

**چی اتفاق می‌افته:**

* وقتی کاربر به مرحله "انتخاب نام" می‌رسه، سیستم پرامپت مخصوص اون مرحله رو می‌فرسته
* Gemini فقط ۵ تا گزینه برمی‌گردونه (بدون هیچ حرف اضافه)
* وقتی کاربر به مرحله "نوشتن چکیده" می‌رسه، سیستم پرامپت دیگه‌ای می‌فرسته
* Gemini یک متن کامل ۲۰۰ کلمه‌ای برمی‌گردونه (بدون شماره‌گذاری)

**فاز ۳: یکپارچه‌سازی UI/UX**

**چرا مهمه؟** چون باید کاربر بتونه این تغییرات رو ببینه و باهاش کار کنه.

**چی می‌خواد بشه:**

1. **کامپوننت راهنمایی مرحله (StageGuidance)**
   * یک باکس در بالای صفحه چت
   * نمایش: "در این مرحله، لطفاً ایده اصلی خود را توضیح دهید" (از stagePrompts.ts)
   * استایل: پس‌زمینه آبی روشن، آیکون لامپ، فونت Vazirmatn
2. **کامپوننت نمایش پاسخ هوش مصنوعی (AIResponseDisplay)**
   * نمایش پاسخ Gemini (با فرمت مناسب)
   * اگر نوع selection\_list بود → نمایش به صورت دکمه‌های انتخابی
   * اگر نوع full\_text بود → نمایش به صورت متن ساده
   * اگر نوع structured\_list بود → نمایش به صورت لیست تاشو
   * دو دکمه: "قبول" و "دوباره بساز"
3. **بروزرسانی ChatInterface**
   * اضافه کردن <StageGuidance /> در بالا
   * تغییر نمایش پیام‌ها: اگر از AI بود → استفاده از <AIResponseDisplay />
   * اگر کاربر دکمه "قبول" رو زد → ذخیره پاسخ و رفتن به مرحله بعد
   * اگر دکمه "دوباره بساز" رو زد → فرستادن پرامپت دوباره
4. **اتصال به سرویس ترجمه**
   * هر جایی که متن ثابت داریم → استفاده از t('key')
   * هر جایی که متن داینامیک داریم → استفاده از translateStage()
   * اگر کاربر زبان رو عوض کرد → کل رابط کاربری بروزرسانی بشه
5. **بهبود تجربه کاربری**
   * اگر هوش مصنوعی داره فکر می‌کنه → نمایش لودینگ
   * اگر خطا رخ داد → نمایش پیام خطا به زبان کاربر
   * اگر کاربر ورودی داد → ارسال به همراه پرامپت مرحله
   * نمایش پیشرفت: "مرحله ۱۵ از ۱۱۴"

**چی اتفاق می‌افته:**

* کاربر وارد یک مرحله می‌شه → بالای صفحه می‌بینه "در این مرحله چی باید بکنم؟"
* (اختیاری) چیزی می‌نویسه یا خالی ارسال می‌کنه
* هوش مصنوعی پاسخ می‌ده (دقیقاً همون فرمتی که می‌خواستیم)
* کاربر قبول می‌کنه → می‌ره مرحله بعد
* یا رد می‌کنه → دوباره از هوش مصنوعی می‌خواد بسازه

**⏱️ تخمین زمان**

* **فاز ۰:** ۳-۴ ساعت (یک بار برای همیشه)
* **فاز ۱:** ۲-۳ ساعت (بررسی و تصمیم‌گیری)
* **فاز ۲:** ۶-۸ ساعت (نوشتن ۱۱۴ پرامپت)
* **فاز ۳:** ۴-۶ ساعت (کامپوننت‌ها و تست)

**جمع کل:** ۱۵-۲۱ ساعت

**🔄 فلو کامل از نگاه کاربر**

1. کاربر وارد برنامه می‌شه
2. زبان فارسی رو انتخاب می‌کنه
3. لاگین می‌کنه
4. روی "شروع سفر کارآفرینی" کلیک می‌کنه
5. می‌بینه: "مرحله ۱: عنوان ایده - لطفاً ایده خود را در یک جمله بنویسید"
6. می‌نویسه: "یک اپلیکیشن برای یادگیری زبان با واقعیت مجازی"
7. کلیک می‌کنه روی ارسال
8. هوش مصنوعی جواب می‌ده: یک پاراگراف کامل (۱۰۰-۱۵۰ کلمه)
9. دکمه "قبول" می‌زنه
10. می‌ره مرحله ۲: "نام تجاری - لطفاً یک نام مناسب انتخاب کنید"
11. هیچی نمی‌نویسه، فقط ارسال می‌کنه
12. هوش مصنوعی جواب می‌ده:

1. VirtuaLingo

2. MetaSpeak

3. ImmersiLang

4. RealityTongue

5. HoloGlot

1. روی "VirtuaLingo" کلیک می‌کنه (قبول)
2. می‌ره مرحله ۳...

این فرایند تا مرحله ۱۱۴ ادامه داره.

**🎨 جزئیات مهم معماری**

**چطور پرامپت‌ها اجرا می‌شن؟**

1. کاربر روی مرحله X هست
2. سیستم از stagePrompts.ts اطلاعات مرحله X رو میگیره
3. یک پرامپت کامل می‌سازه:
   * پرامپت سیستمی از stagePrompts.ts
   * اطلاعات کاربر از مراحل قبلی
   * ورودی کاربر (اگر داده باشه)
   * زبان انتخابی
4. پرامپت رو به geminiService.ts می‌فرسته
5. geminiService.ts چک می‌کنه: PHP proxy یا Direct API?
6. به Gemini میفرسته
7. جواب رو دریافت می‌کنه
8. پارس می‌کنه و به کامپوننت برمی‌گردونه
9. کامپوننت بر اساس outputType نمایش می‌ده

**چطور ترجمه کار می‌کنه؟**

1. کاربر زبان عربی رو انتخاب می‌کنه
2. LanguageContext زبان فعلی رو به ar تغییر می‌ده
3. کل برنامه ری‌رندر می‌شه
4. هر کامپوننت که متن داره، t('کلید') یا translateStage() رو صدا می‌زنه
5. سرویس ترجمه اول توی حافظه نگاه می‌کنه
6. اگر نبود، توی stage\_translations جستجو می‌کنه با WHERE stage\_id = X AND language = 'ar'
7. اگر پیدا شد → برمی‌گردونه
8. اگر نبود → از Gemini می‌خواد: "Translate this to Arabic: ..."
9. ترجمه رو دریافت می‌کنه
10. توی حافظه + دیتابیس ذخیره می‌کنه
11. به کامپوننت برمی‌گردونه

**چطور کش کار می‌کنه؟**

درخواست ترجمه → حافظه (RAM) → دیتابیس (Supabase) → API (Gemini)

↓ ↓ ↓

0.001s 0.1-0.3s 1-3s

* اولین بار: کند (API call)
* بار دوم (همون سشن): خیلی سریع (حافظه)
* بار سوم (سشن جدید): سریع (دیتابیس)

**📊 مقایسه قبل و بعد**

**قبل:**

کاربر: من می‌خوام یک استارتاپ بسازم

AI: سلام! چه خوشحال شدم که می‌خواهید یک استارتاپ بسازید.

انتخاب نام یکی از مهم‌ترین تصمیمات شماست. یک نام خوب

باید کوتاه، به یاد ماندنی و مرتبط با کسب‌وکار شما باشه.

اینجا چند پیشنهاد دارم:

1. TechVenture

2. InnoStart

3. FutureLabs

همچنین توصیه می‌کنم که...

**بعد:**

کاربر: من می‌خوام یک استارتاپ بسازم

AI:

1. TechVenture

2. InnoStart

3. FutureLabs

4. NexGen Solutions

5. Quantum Leap

**🛡️ چالش‌ها و راه‌حل‌ها**

**چالش ۱: هوش مصنوعی گاهی قانون‌ها رو نقض می‌کنه**

**راه‌حل:**

* پرامپت رو سخت‌گیرانه‌تر نوشتن
* افزودن جمله: "CRITICAL: If you add ANY text before or after, the system will fail"
* چک کردن خروجی با regex و اگر درست نبود، دوباره فرستادن

**چالش ۲: ترجمه گاهی نادرست میاد**

**راه‌حل:**

* افزودن زمینه به پرامپت ترجمه: "This is for a startup mentoring app"
* ذخیره ترجمه‌های دستی در دیتابیس و استفاده از اونها به جای API
* اضافه کردن دکمه "گزارش ترجمه نادرست"

**چالش ۳: ۱۱۴ پرامپت خیلی زیاده**

**راه‌حل:**

* شروع با ۱۰-۱۵ مرحله مهم
* استفاده از الگوهای تکراری (مثلاً همه مراحل "تحلیل" شبیه همن)
* نوشتن یک جنریتور پرامپت که از تمپلیت استفاده کنه

**چالش ۴: زمان پاسخ‌دهی API**

**راه‌حل:**

* نمایش لودینگ جذاب (نه فقط چرخنده)
* استفاده از streaming اگر Gemini پشتیبانی کنه (کلمه به کلمه نمایش بده)
* پیش‌بینی مرحله بعدی و از قبل ترجمه کردن

**🎯 اولویت‌بندی (اگر زمان محدوده)**

**حتماً باید:**

* فاز ۰: سیستم ترجمه (پایه)
* فاز ۲: پرامپت برای ۱۵ مرحله اصلی
* فاز ۳: کامپوننت AIResponseDisplay برای نمایش درست

**خوبه داشته باشیم:**

* فاز ۱: بررسی کامل ۱۱۱ مرحله
* فاز ۲: پرامپت برای همه مراحل
* فاز ۳: انیمیشن‌ها و ریز کاری‌ها

**اگر زمان اضافه داشتیم:**

* سیستم گزارش ترجمه نادرست
* داشبورد ادمین برای ویرایش پرامپت‌ها
* A/B Testing پرامپت‌های مختلف
* تحلیل رفتار کاربر