

مستند فنی تدریسینو

معماری، ساختار کد، مدل داده، APIها، دستیار هوش مصنوعی و داده‌ی Qdrant،
گیمیفیکیشن، امنیت، استقرار و مسیر مقیاس‌پذیری – برای خواننده‌ای که با پروژه آشنایی قبلی
ندارد و می‌خواهد آن را توسعه دهد یا نگهداری کند.

تاریخ	۸ مهر ۱۴۰۵ (۳۰ سپتامبر ۲۰۲۶)
نویسنده	علیرضا زارعیان – بک‌اند و فرانت‌اند
مخاطب	توسعه‌دهنده‌ی تازه‌وارد، مدیر فنی، و هر کسی که باید درباره‌ی ادامه‌ی پروژه تصمیم بگیرد
کد مبنا	بک‌اند 957c9d0 (مخزن Tadrino/MathApp) · فرانت 0e56d9c (مخزن Tadrino/MathApp_frontend)
نشانی‌ها	اپ: math.tadrino2.ir · API: mathapi.tadrino2.ir · مستندات زنده‌ی API: /api/docs/

فهرست

۱ معرفی پروژه در یک نگاه	۱۰ محتوای آموزشی و ساخت سؤال
۲ انتخاب‌های فناوری و چرایی‌شان	۱۱ گیمیفیکیشن و همراه‌ها
۳ معماری کلی	۱۲ امنیت و احراز هویت
۴ ساختار فایل‌ها و نقش هر فایل	۱۳ استقرار (Deploy) و نگهداری
۵ سفر یک درخواست، فایل‌به‌فایل (با کد واقعی)	۱۴ مقیاس‌پذیری و پیوستن توسعه‌دهنده‌ی تازه
۶ سناریوهای اصلی	۱۵ تست و کیفیت
۷ مدل داده	۱۶ وضعیت فعلی، محدودیت‌ها و تصمیم‌های باز
۸ API: قواعد و فهرست کامل	پ پیوست: واژه‌نامه، دستورها، نقشه‌ی مستندات
۹ دستیار هوش مصنوعی و داده‌ی Qdrant	

۱ معرفی پروژه در یک نگاه

تدریسینو یک اپ آموزش ریاضی به سبک دولینگو است: دانش آموز روی نقشه‌ای از «جزیره»ها جلو می‌رود، هر جزیره یک آزمون کوتاه از مسئله‌های خود کتاب درسی است، و هر جا گیر کرد از دستیار هوش مصنوعی («پی») می‌پرسد که جوابش را از روی همان کتاب می‌سازد.

۸ پایه

سوم، ششم، هفتم، هشتم تا دوازدهم

۲,۵۴۵

سؤال چهارگزینه‌ای از مسئله‌های کتاب

۶۱

اندپوینت API، مستند در Swagger

۶۸۵

تست خودکار، همه سبز

۱۵

اپ جنگو، هرکدام یک مسئولیت

۳,۵۱۷

تکه‌ی کتاب در (۱۱ Qdrant collection)

۴

همراه گیمیفیکیشن، هرکدام ۶ حالت چهره

۲ مخزن

پکاند و فرانت؛ یک قرارداد API

کاربرها و کاری که می‌کنند

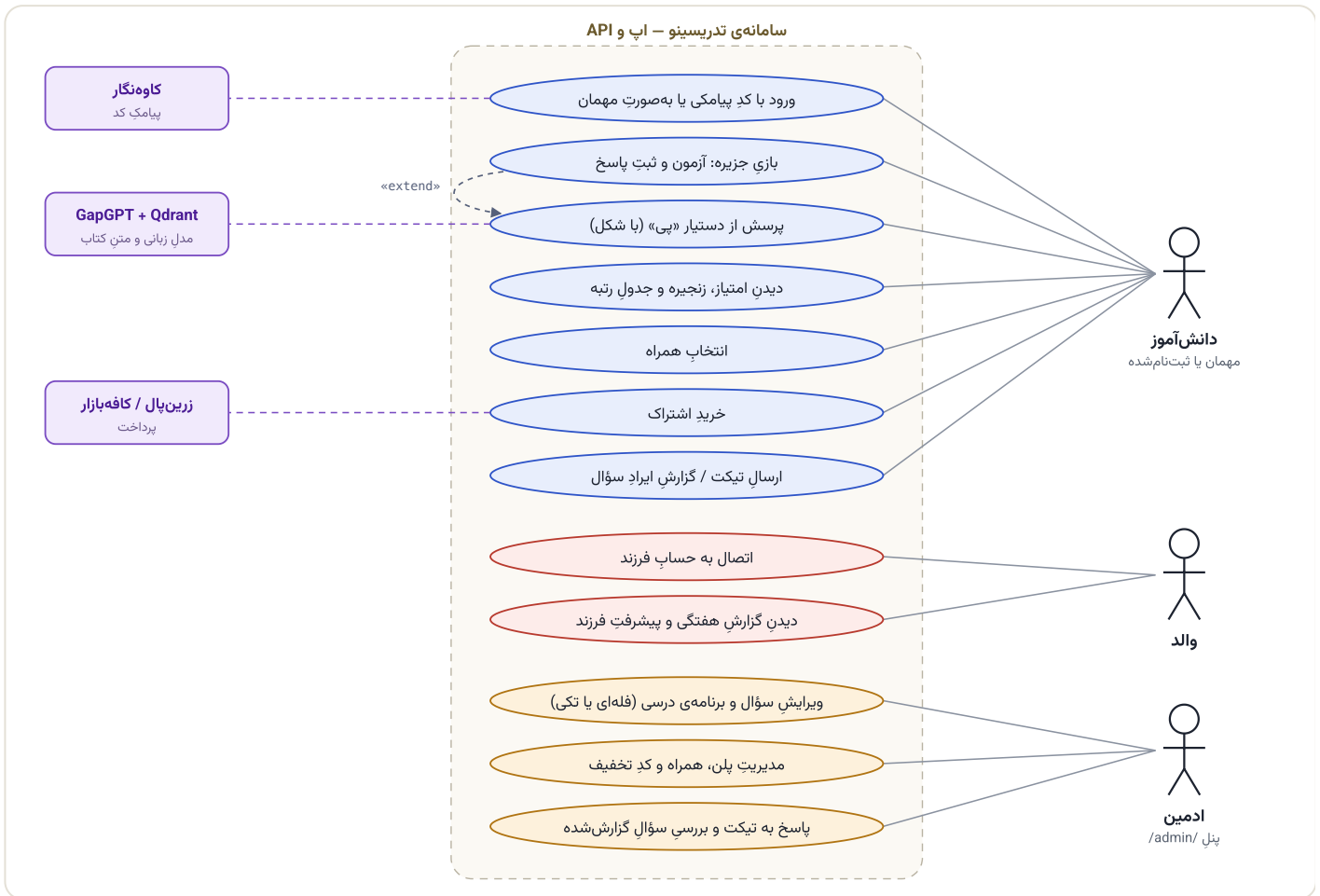
کاربر	چه می‌کند	کجای کد
دانش آموز مهمان	بدون شماره وارد می‌شود، پایه‌اش را انتخاب می‌کند و بازی می‌کند. اگر بعداً با شماره ثبت‌نام کند، همان حساب می‌ماند و امتیازش گم نمی‌شود.	accounts/views.py
دانش آموز	با شماره‌ی موبایل و کد پیامکی وارد می‌شود؛ جزیره‌ها را بازی می‌کند، از دستیار می‌پرسد، امتیاز و سکه و زنجیره‌ی روزانه جمع می‌کند، همراه انتخاب می‌کند، اشتراک می‌خرد، تیکت پشتیبانی می‌فرستد.	progress/ content/ ai_tutor/
والد	به حساب فرزند وصل می‌شود و گزارش هفتگی و پیشرفتش را می‌بیند.	family/
ادمین (کارمند شرکت)	از پنل /admin/ سؤال، درس، پلن، همراه و تیکت را بدون برنامه‌نویس ویرایش می‌کند؛ سؤال‌ها را فله‌ای وارد و خارج می‌کند.	*/admin.py

تاریخچه‌ی کوتاه

زمان	اتفاق
مرداد ۱۴۰۵	طراحی معماری و مدل‌های داده؛ هویت کاربر؛ جداسازی اپ‌ها. بازبینی امنیتی اول و بستن حفره‌ها، محدودیت نرخ در کل سیستم.
شهریور	دستیار از پاسخ ساختگی به RAG واقعی روی کتاب رسید؛ اتصال زیرین پال؛ گیمیفیکیشن؛ خواندن پنج کتاب ریاضی و ساخت هزاران سؤال تأییدشده؛ حسابان ۱۱ و ۱۲.
۲ تا ۷ مهر	فرانت به ما سپرده شد و از نو ساخته شد؛ هر دو روی دامنه‌ی تازه زنده شدند؛ پیامک واقعی؛ شکل در جواب دستیار؛ همراه چهارم؛ بازبینی نهایی روی سایت زنده.

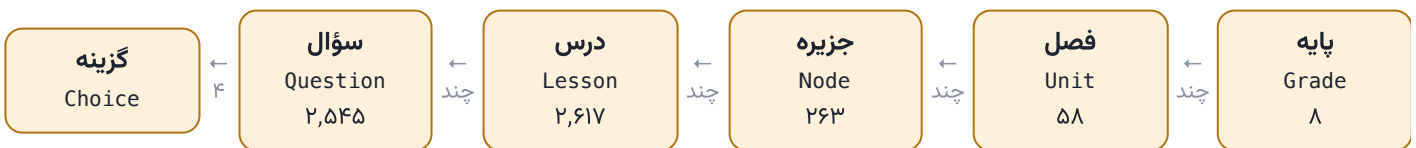
مدل سناریو (Use Case)

چه کسانی با سامانه کار می‌کنند و هرکدام چه کاری انجام می‌دهند. خط‌چین بنفش یعنی آن کار به یک سرویس بیرونی وابسته است. «extend» یعنی دانش‌آموز از دلِ آزمون، پس از پاسخِ نادرست، مستقیم از دستیار می‌پرسد «چرا غلط شد؟». جزئیاتِ گام‌به‌گامِ مهم‌ترین سناریوها در بخش ۶ آمده است.



شکل ۱-۱ - مدل سناریو: سه نقش کاربری، دوازده کار اصلی، و سه سرویس بیرونی.

واحدهای محتوا (این واژه‌ها در کل سند تکرار می‌شوند)

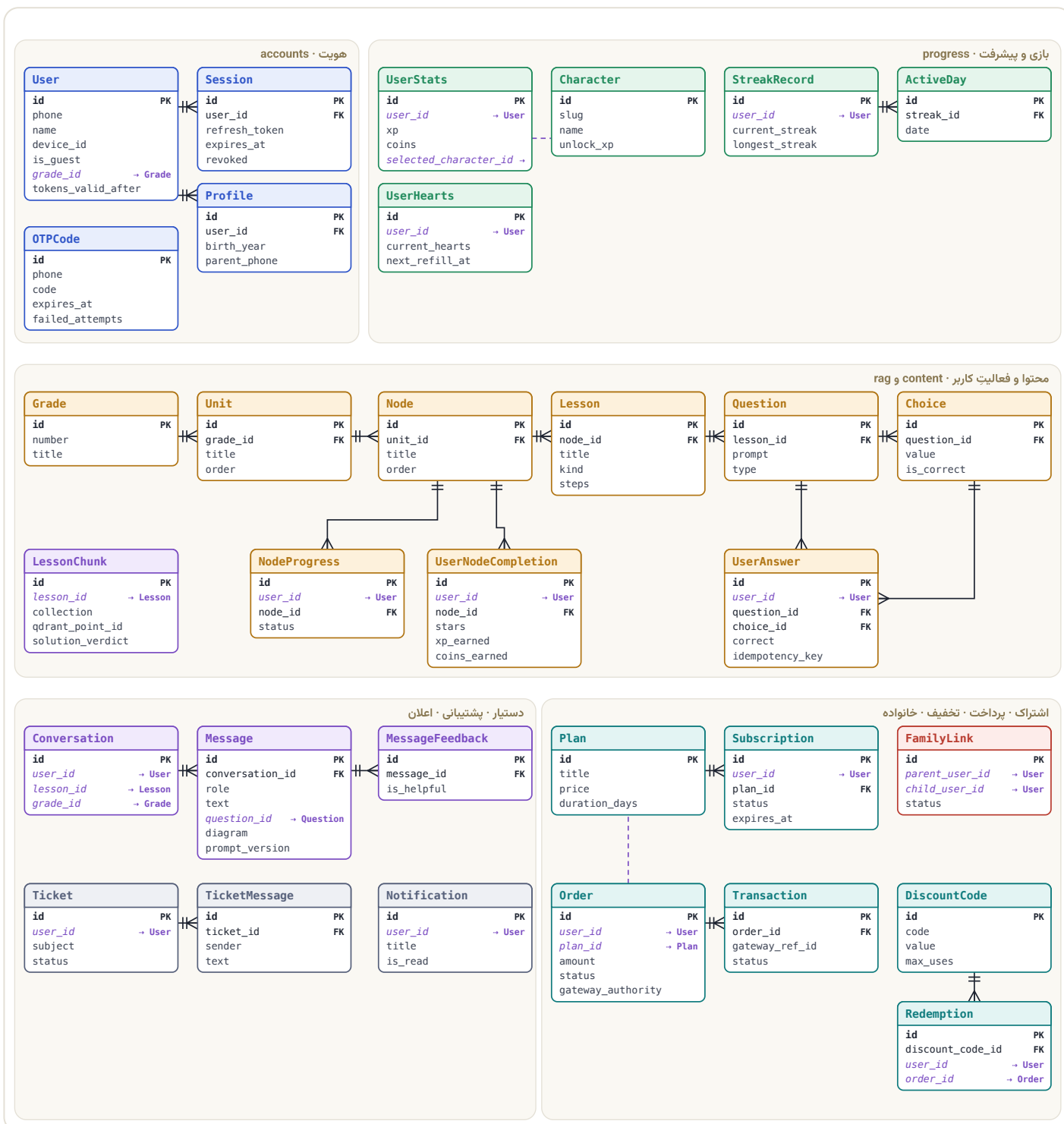


«درس» در این پروژه یعنی یک «تیپ‌مسئله»ی کتاب، نه یک جلسه‌ی کلاس. برای همین درس خیلی ریز است و اپ در سطحِ جزیره (Node) بازی می‌شود: آزمونِ هر جزیره از درس‌هایش سؤال برمی‌دارد.

مدل رابطه‌ای (ER)

جدول‌های اصلی دیتابیس و ارتباط میان آن‌ها، دسته‌بندی‌شده بر اساس اپ. **خط ممتد** یک کلید خارجی واقعی (FK) در داخل یک اپ است: دو خط موازی سمت «دقیقاً یک» و پای کلاغی سمت «صفر یا چند» (مثلاً هر پایه چند فصل دارد). **فیلد بنفش** با → ارجاع سبک به جدول اپ دیگر است: فقط شناسه ذخیره می‌شود و قید دیتابیزی ندارد تا هر اپ مستقل بماند (بخش ۳).

خط چین بنفش دو نمونه از همین ارجاع‌های سبک را نشان می‌دهد. فیلدهای کم‌اهمیت حذف شده‌اند؛ فهرست کامل در بخش ۷ و در `<app>/models.py`.



شکل ۲-۱ - مدل رابطه‌ای؛ نام‌ها و ارتباط‌ها مستقیم از مدل‌های جنگو استخراج شده‌اند.

۲ انتخاب‌های فناوری و چرایی‌شان

هر انتخاب یک دلیل و یک هزینه دارد. ستون آخر هزینه را صادقانه می‌گویند، چون همان چیزی است که نفر بعدی باید بداند.

لایه	انتخاب	چرا	چرا نه جایگزین	هزینه / ریسک
بک‌اند	Django 5.2 + Django REST Framework	ORM و migration و پنل ادمین آماده در یک بسته؛ پنل ادمین یعنی محتوا بدون برنامه‌نویس ویرایش می‌شود. DRF احراز هویت، مجوز، محدودیت نرخ و سریال‌سازی را استاندارد می‌دهد. پایتون همان زبان کتابخانه‌های هوش مصنوعی است (openai ، qdrant client ، pymupdf).	FastAPI سریع‌تر و async است ولی ادمین، ORM و migration را باید جدا سرهم کرد. Node.js زبان بک‌اند را از زبان لایه‌ی AI جدا می‌کند.	جنگو پیش‌فرض همگام (sync) است؛ جواب طولانی دستیار یک نخ را نگه می‌دارد. با ۲۴ نخ هم‌زمان حل شد (بخش ۱۴).
قرارداد API	OpenAPI با drf-spectacular	اسکیما از خود کد ساخته می‌شود، پس هیچ‌وقت از کد عقب نمی‌ماند. Swagger در /api/docs/ قرارداد مشترک دو مخزن است.	سند دستی یا «در ذهن کسی» — که دقیقاً باعث ناهمخوانی اولیه‌ی فرانت و بک‌اند شد.	—
فرانت	React 18 + TypeScript (strict) + Vite + Tailwind	فرانت اولیه را توسعه‌دهنده‌ی قبلی با React ساخته بود؛ در بازسازی نگه داشته شد چون بزرگ‌ترین اکوسیستم را دارد و نیروی جایگزینش راحت پیدا می‌شود. TypeScript ناهمخوانی شکل داده با API را در زمان build می‌گیرد. Vite build سریع و Tailwind رنگ و فاصله‌ی یکدست. وب موبایل‌محور یعنی بدون نصب و بدون فروشگاه روی هر گوشی باز می‌شود.	React Native / Flutter نیاز انتشار در فروشگاه و build جدا داشت؛ نسخه‌ی اندروید (کافه‌بازار) بعداً می‌تواند اضافه شود.	بدون کتابخانه‌ی router و state؛ کمتر وابستگی، ولی الگوی داخلی را باید خواند (بخش ۴).
دیتابیس	SQLite (فعال)	روی سرور مشترک شرکت صفر نگهداری دارد: یک فایل، بکاپ با یک دستور. برای حجم فعلی کافی است. کد از ORM استفاده می‌کند و به SQLite گره نخورده.	Postgres روی همان سرور موجود است؛ مهاجرت به هماهنگی با ادمین سرور نیاز دارد و به مرحله‌ی بعد موکول شده است.	فقط یک نویسنده در لحظه. اولین گلوگاه مقیاس است (بخش ۱۴).

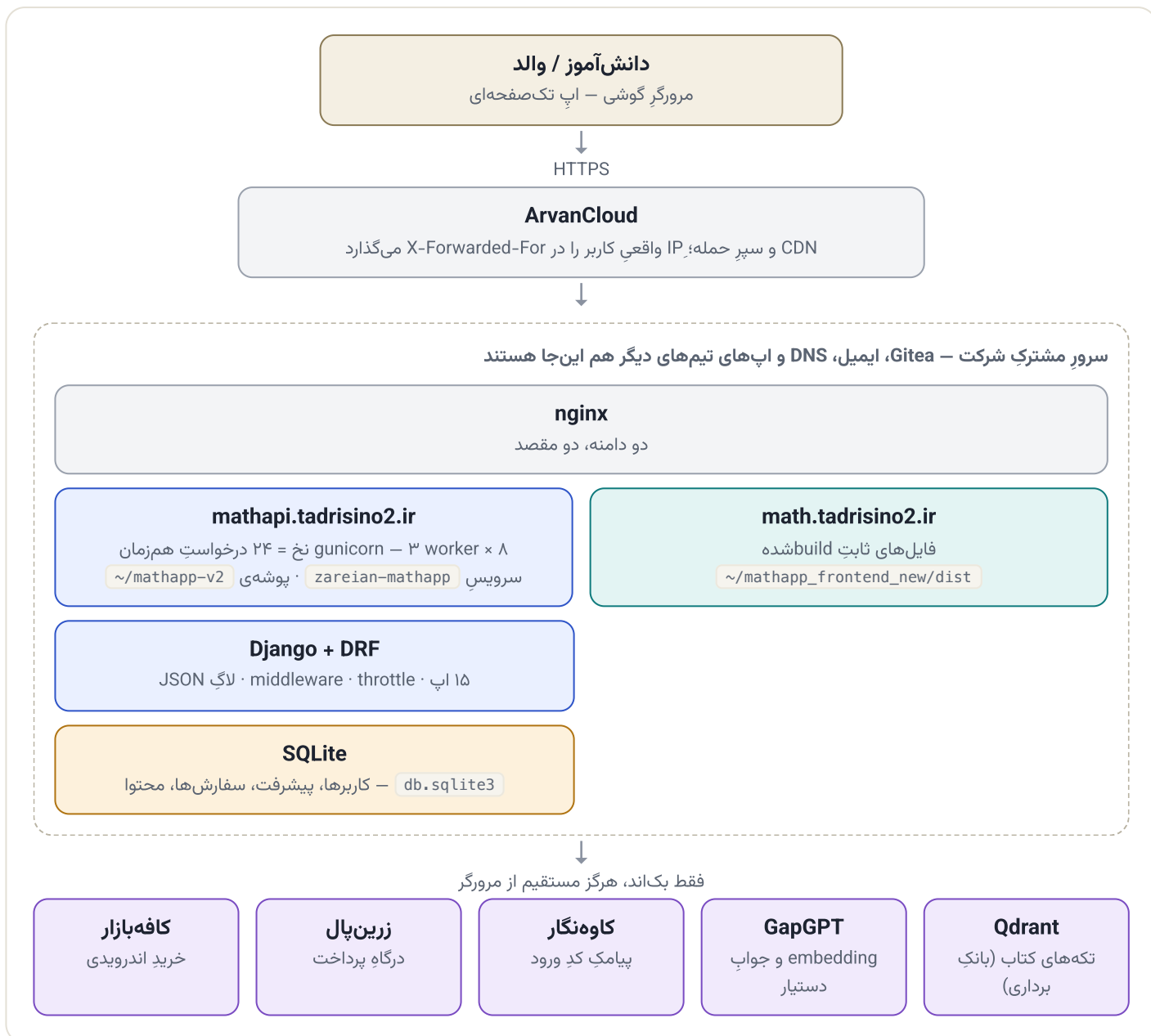
لایه	انتخاب	چرا	چرا نه جایگزین	هزینه / ریسک
جست‌وجوی کتاب	Qdrant (بانک برداری)	تیم دیگر شرکت کتاب‌های سوم / ششم/هفتم را از قبل در یک مشترک گذاشته بود؛ همان را ادامه دادیم تا داده دوباره ساخته نشود. فیلتر روی فیلد «فصل» دارد.	Postgres به pgvector نیاز داشت؛ FAISS سرور ندارد و بین دو تیم اشتراک‌پذیر نیست.	سرویس مشترک با تیم دیگر؛ پاک‌کردن باید با احتیاط باشد (بخش ۹).
مدل زبانی	GapGPT ← gpt-4.1-mini	GapGPT یک واسط سازگار با OpenAI است که از داخل ایران جواب می‌دهد. مدل با سنجش انتخاب شد، نه با جدول قیمت: روی ۱۰۲ مسئله‌ی تصحیح‌شده ۹۵-۹۶ درست در برابر ۸۵-۹۰ برای gpt-4o-mini؛ هر جواب حدود ۰/۱۲ سنت.	OpenAI مستقیم از ایران در دسترس نیست. مدل‌های گران‌تر در دور اول سنجش (۱۴ مدل) مزیتی نشان ندادند.	گنبد سرویس (حتی «سلام» ۷ ثانیه) دست ما نیست؛ اعتبار دو بار تمام شده.
ورود	توکن امضاشده‌ی خودمان + کد پیامکی	کاربرها با شماره وارد می‌شوند، نه نام‌کاربری/رمز؛ مدل کاربر جنگو به این نمی‌خورد. توکن با <code>django.core.signing</code> امضا می‌شود؛ یک وابستگی کمتر از کتابخانه‌ی JWT.	JWT همان کار را می‌کند با یک کتابخانه‌ی اضافه.	امنیتش به کلید امضا بسته است (بخش ۱۲، مورد ۱).
پیامک / پرداخت	کاوه‌نگار · زرین‌پال · کافه‌بازار	سرویس‌های ایرانی رایج؛ کاوه‌نگار با «ارسال با الگو» که روی خط شرکت فعال است؛ زرین‌پال برای وب، بازار برای خرید درون‌برنامه‌ای اندروید.	—	پرداخت واقعی تا انتها هنوز آزموده نشده.
اجرا	gunicorn (پشت‌پشت nginx) ArvanCloud	الگوی استاندارد اجرای جنگو. آروان CDN و سپر حمله‌ی ایرانی است. WhiteNoise فایل‌های استاتیک ادمین را از خود جنگو سرو می‌کند.	—	سرور مشترک است؛ مال ما فقط یک پوشه و یک سرویس.

اصل کلی انتخاب‌ها

هر جا ممکن بود، ابزار آماده و رایج انتخاب شده است؛ هر جا انتخاب بر کیفیت اثر مستقیم داشت (مانند مدل هوش مصنوعی)، گزینه‌ها پیش از انتخاب با داده‌ی واقعی سنجیده شده‌اند.

۳ معماری کلی

دو برنامه‌ی جدا که فقط از راه HTTP با هم حرف می‌زنند: فرانت (فایل‌های ثابت در مرورگر) و بک‌اند (API). هر دو روی یک سرور مشترک شرکت‌اند.



شکل ۱-۳ - نقشه‌ی استقرار. همه‌ی کلیدهای سرویس‌های بیرونی فقط در `.env` سرورند و هیچ‌وقت به مرورگر نمی‌رسند.

درون بک‌اند: «یک برنامه، چند اپ مستقل» (Modular Monolith)

بک‌اند یک برنامه‌ی جنگوست که به ۱۵ «اپ» تقسیم شده. هر اپ یک پوشه است با مدل‌ها، API‌ها و تست‌های خودش. این همان الگویی است که اجازه می‌دهد چند نفر هم‌زمان روی بک‌اند کار کنند (بخش ۱۴).

مدل‌ها (جدول‌ها)	مسئولیت	اپ
User Profile Session OTPCode	هویت، ورود با کد پیامکی، نشست ورود، پروفایل	accounts
Grade Unit Node Lesson Question Choice UserAnswer UserLessonCompletion UserNodeCompletion NodeProgress QuestionReport	برنامه‌ی درسی، سؤال، آزمون جزیره، ثبت جواب و پیشرفت	content
UserStats UserHearts StreakRecord ActiveDay LeaderboardEntry Character	امتیاز، سکه، قلب، زنجیره، جدول رتبه، همراه‌ها	progress
Conversation Message MessageFeedback AskIdempotency	گفتگو با دستیار، سهمیه، بازخورد / 👍 / 🗋️	ai_tutor
LessonChunk LessonChunkImage IngestedPage	خواندن کتاب، Qdrant، ساخت پرامپت، شکل‌ها	rag
Plan Subscription	پلن‌ها و اشتراک کاربر	subscription
Order Transaction BazaarCredential	سفارش، تراکنش، زرین‌پال، بازار	payment
DiscountCode Redemption Offer UserOffer	کد تخفیف و پیشنهاده‌ها	discount
AffiliateAccount Conversion	معرفی به دوستان و پورسانت	affiliate
Notification	اعلان‌های داخل اپ	notifications
Ticket TicketMessage BugReport	تیکت و گزارش باگ	support
FamilyLink ParentalSettings WeeklyReportLog	اتصال والد و فرزند، گزارش هفتگی	family
MessageCampaign CampaignDelivery	پیام گروهی (ذخیره و هدف‌گیری؛ زمان‌بندی خودکار ندارد)	campaigns
StatusChangeLog	ثبت عمومی «وضعیت فلان چیز از X به Y رفت»	statuslog
—	تنظیمات، مسیرهای اصلی، middleware، سلامت، مدیریت خطا	config

۱. بین اپ‌ها کلید خارجی (FK) نیست

اپ `progress` کاربر را با یک عدد ساده نگه می‌دارد، نه با اتصال دیتابیزی به `accounts`:

```
# progress/models.py
class UserStats(models.Model):
    user_id =
models.IntegerField(unique=True) # ارجاع
سبک
    xp =
models.PositiveIntegerField(default=0)
    coins =
models.PositiveIntegerField(default=0)
```

چرا: اگر روزی «پرداخت» یا «دستیار» سرویس جدا شد، زنجیره‌ای از وابستگی پاره نمی‌شود. داخل یک اپ FK آزاد است (مثلاً `Question → Lesson`).

۲. اپ‌ها با «رویداد» خبر می‌دهند

وقتی پرداخت تأیید شد، `payment` خودش اشتراک و اعلان را نمی‌سازد؛ رویداد می‌فرستد یا تابع سرویس اپ دیگر را صدا می‌زند، و هر اپ کار خودش را می‌کند. فایل‌های مربوط:

• رویدادهایی که یک اپ می‌فرستد: `*/signals.py`
• شنونده‌ها: `*/receivers.py` ، `*/services.py`
توابعی که اپ دیگر می‌تواند صدا بزند (مثلاً `notifications.services.notify`، `affiliate.services.record_paid_order`).

۴ ساختار فایل‌ها و نقش هر فایل

این بخش نقشه‌ی راه است: هر فایل چه چیزی دارد و به کدام فایل وصل است.

MathApp/

└─ config/	تنظیمات و سیم‌کشی کل پروژه
└─ settings.py	همه‌ی تنظیمات؛ خواندن env؛ نرخ throttle؛ کلید امضا؛ مدل‌های AI
└─ urls.py	ریشه‌ی مسیرها: admin، /api/docs، /api/health/ و include هر اپ
└─ middleware.py	لاگ JSON هر درخواست، no-store، سقف ورود ادمین
└─ exceptions.py	هر خطا → پاسخ JSON یکدست، بی‌جزئیات داخلی
└─ health.py	GET /api/health: دیتابیس، migration، کش، محتوا، هشدارها
└─ log_formatters.py	قالب JSON لاگ
└─ accounts/ content/ progress/ ai_tutor/ rag/ subscription/ payment/	
└─ discount/ affiliate/ notifications/ support/ family/ campaigns/ statuslog/	
└─ content/seed/	questions.json و curriculum.json – کل محتوا به‌صورت فایل در git
└─ data/textbooks/	PDF کتاب‌ها (در git نیست؛ فقط README)
└─ templates/tadrino/	پوسته‌ی راست‌به‌چپ Swagger و ReDoc
└─ tools/fake_openai.py	جایگزین GapGPT برای تست بی‌هزینه
└─ docs/	... operations، architecture، security-before-production، content-authoring
└─ deploy.sh	کل انتشار در یک اسکریپت (بخش ۱۳)
└─ gunicorn.conf.py	۸ نخ در هر ۱۲۰ timeout، worker
└─ requirements.txt	... Django 5.2، DRF، drf-spectacular، openai، qdrant-client، pymupdf
└─ docker-compose.yml	فقط محلی: Loki + Alloy + Grafana برای دیدن لاگ
└─ manage.py	

۴-۲ آناتومی یک اپ (همه‌ی اپ‌ها همین شکل‌اند)

مثال: اپ `content`. ترتیب جدول همان ترتیبی است که یک درخواست از آن رد می‌شود.

فایل	چه چیزی در آن است	نمونه‌ی واقعی
<code>urls.py</code>	نگاشتِ آدرس به کلاسِ ویو	<code>questions/<int:question_id>/answer</code> → <code>QuestionAnswerView</code>
<code>views.py</code>	منطقِ هر اندپوینت: چه کسی مجاز است (<code>permission_classes</code>)، چند بار (<code>throttle_scope</code>)، و کار اصلی	<code>NodeQuizView</code> ، <code>QuestionAnswerView</code> <code>NodeQuizCompleteView</code>
<code>serializers.py</code>	شکلِ ورودی و خروجی؛ اعتبارسنجی؛ همین فایل Swagger را می‌سازد	<code>AnswerRequestSerializer</code> (فقط (<code>selectedOptionId</code>)
<code>services.py</code>	منطقِ مشترکی که چند ویو یا اپ دیگر لازم دارند	درصدِ پیشرفت، جزیره‌ی بعدی
<code>models.py</code>	جدول‌های دیتابیس و قیدهای یکتایی	<code>UserAnswer</code> با قیدِ یکتا روی کلیدِ <code>idempotency</code>
<code>migrations/</code>	تاریخچه‌ی تغییرِ جدول‌ها؛ <code>migrate</code> روی سرور اعمال‌شان می‌کند	<code>id: 0008</code> هر پایه = شماره‌ی پایه
<code>admin.py</code>	صفحه‌های پنلِ ادمین، اکشن‌ها، واردکردنِ فله‌ای	«وارد کردنِ سؤال‌ها از فایل»
<code>receivers.py</code>	واکنش به رویدادِ اپ‌های دیگر	باطل کردنِ کشِ نقشه وقتی سؤال عوض شد
<code>management/commands/</code>	دستورهای خطِ فرمان (<code>python manage.py ...</code>)	<code>seed_content</code> ، <code>generate_questions_from_chunks</code>
<code>tests*.py</code>	تست‌ها؛ هر باگِ پیداشده یک تست دارد	<code>tests_frontend_contract.py</code> – نخبانِ قرارداد با فرانت

۳-۴ مخزنِ فرانت (`Tadrisino/MathApp_frontend`)

`src/`

<code>main.tsx</code>	نقطه‌ی شروع؛ App را سوار می‌کند
<code>config.ts</code>	نشانی <code>/api</code> ← <code>VITE_API_BASE_URL</code> ← <code>dist/config.js</code> API:
<code>api/</code>	تنها در ارتباط با سرور – هیچ جای دیگر <code>fetch</code> نمی‌زند
<code>http.ts</code>	توکن، <code>timeout</code> ، تکرار <code>refresh</code> ، <code>GET</code> ، تک‌نسخه‌ای، <code>SSE</code> ، <code>Idempotency-Key</code>
<code>index.ts</code>	یک تابع برای هر اندپوینت: <code>api.lesson.answer</code> ، <code>api.nodeQuiz</code> ، ...
<code>types.ts</code>	شکلِ پاسخ‌ها، از روی پاسخِ واقعیِ سرور
<code>app/</code>	<code>App.tsx</code> : جریانِ ورود <code>MainApp.tsx</code> : تب‌ها و صفحه‌ها
<code>nav/</code>	ناوبریِ پشت‌پایِ خودمان، هماهنگ با دکمه‌ی <code>back</code> گوشی
<code>state/</code>	داده‌ی مشترک (پروفایل، نقشه) و گفتگوی دستیار
<code>screens/</code>	یک فایل برای هر صفحه: <code>FamilyScreen</code> ، <code>AskScreen</code> ، <code>QuizScreen</code> ، <code>HomeScreen</code> ، ...
<code>components/</code>	اجزای مشترک: مسیرِ جزیره‌ها، برگه‌ی پایین، نوارِ پایین
<code>mascots/</code>	سیگما، دلتا، پی، تدی – SVG از کد، ۶ حالتِ چهره
<code>diagram/</code>	JSONِ شکلِ دستیار → SVG؛ مختصات، ریشه، زاویه این‌جا حساب می‌شود
<code>lib/</code>	رقم فارسی، تاریخ شمسی، <code>MathText</code> ، پیام خطای دوستانه، صدا

قاعده‌ی فرانت: صفحه‌ها هرگز مستقیم سرور را صدا نمی‌زنند؛ همیشه `screens/*` → `api/index.ts` → `api/http.ts`

۵ سفر یک درخواست، فایل به فایل

نمونه: ثبت پاسخ یک دانش آموز به یک سؤال در آزمون جزیره. در ادامه مسیر کامل این درخواست از رابط کاربری تا دیتابیس و بازگشت پاسخ آمده است. سایر درخواست‌ها نیز همین مسیر را طی می‌کنند و تنها ویو پاسخ‌دهنده تفاوت دارد.

۱ فرانت · صفحه `src/screens/QuizScreen.tsx` – کاربر گزینه‌ای را انتخاب می‌کند و صفحه تابع API مربوط را فراخوانی می‌کند. صفحه نمی‌داند نشانی سرور چیست.

```
const r = await api.lesson.answer(question.id, optionId, answerKey);
```

۲ فرانت · api `src/api/index.ts` – نام تابع را به نشانی API ترجمه می‌کند. برای هر جواب یک کلید یکتا (Idempotency-Key) ساخته می‌شود تا اگر اینترنت قطع شد و دوباره فرستاده شد، دوبار حساب نشود.

```
answer: (questionId, selectedOptionId, idempotencyKey =
newIdempotencyKey()) =>
  http.create(`/questions/${questionId}/answer`, { selectedOptionId }, {
idempotencyKey })
```

۳ فرانت · http `src/api/http.ts` – توکن را از `localStorage` می‌خواند، هدرها را تنظیم می‌کند و درخواست را با محدودیت زمانی (timeout) می‌فرستد. اگر ۴۰۱ برگشت، یک بار توکن را تازه می‌کند و دوباره می‌فرستد.

```
POST https://mathapi.tadrisino2.ir/api/questions/1834/answer
Authorization: Bearer 10:1tQx9k:Xy3... ← زمان: امضا user_id
Idempotency-Key: 7f3c... {"selectedOptionId": 7290}
```

۴ شبکه ← اینترنت ← `nginx` ← `ArvanCloud` روی سرور ← سوکت یونیکس ← یک نخ آزاد از `gunicorn` (`gunicorn.conf.py`).

۵ بک‌اند · middleware `config/settings.py` → MIDDLEWARE – لایه‌های عمومی به ترتیب: امنیت، CORS (آیا درخواست از `math.tadrisino2.ir` آمده؟)، فایل‌های استاتیک، سقف ورود ادمین، ... ، `APINoStoreMiddleware` (پاسخ در کش آروان نماند)، و در آخر `APIRequestLoggingMiddleware` که در پایان یک خط JSON لاگ می‌نویسد (`config/middleware.py`).

۶ بک‌اند · مسیر `config/urls.py` همه‌ی `/api/` را به اپ‌ها می‌سپارد؛ `content/urls.py` این آدرس را پیدا می‌کند:

```
path('api/', include('content.urls')), #
config/urls.py
path('questions/<int:question_id>/answer', QuestionAnswerView.as_view()),
# content/urls.py
```

accounts/authentication.py – امضا و انقضای توکن را برپایه دیتابیس می‌سند و request.user را می‌سازد. بی‌هادر یعنی «مهمان ناشناس»، نه خطا؛ ویو تصمیم می‌گیرد.

```
value = TimestampSigner(salt='tadrisino-access-token').unsign(token,
max_age=3600)
if user.tokens_valid_after and issued_at(token) <
user.tokens_valid_after: raise AuthenticationFailed
```

content/views.py → QuestionAnswerView – فقط کاربر وارد شده، حداکثر ۶۰ جواب در دقیقه (نرخ در settings DEFAULT_THROTTLE_RATES)

```
permission_classes = [IsAuthenticated]
throttle_scope = 'question_answer' # 'question_answer': '60/min'
```

content/serializers.py – ورودی باید دقیقاً یک عدد باشد؛ هر چیز دیگر ۴۰۰ می‌گیرد. userId هرگز از کلاینت خوانده نمی‌شود، فقط از توکن.

```
class AnswerRequestSerializer(serializers.Serializer):
    selectedOptionId = serializers.IntegerField()
```

ثبت جواب و کم کردن قلب «همه یا هیچ» در یک تراکنش (content/models.py ، progress/models.py):

```
with transaction.atomic():
    UserAnswer.objects.create(user_id=user_id, question=question,
choice=choice,
correct=choice.is_correct,
idempotency_key=idempotency_key)
    if not choice.is_correct:
        hearts =
UserHearts.objects.select_for_update().get(user_id=user_id)
        hearts.apply_regen(); hearts.lose_heart(); hearts.save()
```

قید یکتای (user_id, question, idempotency_key) در دیتابیس آخرین خط دفاع است: درخواست تکراری نمی‌تواند ردیف دوم بسازد. بعد از روز فعال زنجیره ثبت می‌شود (progress/services.py).

```
{"correct": false, "correctOptionId": "7288", "xpEarned": 0,
"heartsRemaining": 4, "mascotMood": "encourage"}
```

اگر هر جا خطا شد، config/exceptions.py آن را به JSON یکدست تبدیل می‌کند و traceback فقط در لاگ می‌ماند.

QuizScreen.tsx گزینه‌ی نادرست را قرمز و گزینه‌ی درست را سبز نمایش می‌دهد، قلب را کم نشان می‌دهد، چهره‌ی همراه را عوض می‌کند و دکمه‌ی «از پی‌پرس چرا غلط شد» را نشان می‌دهد. پیام خطا همیشه از lib/errors.ts → friendlyError می‌آید، نه متن خام سرور.

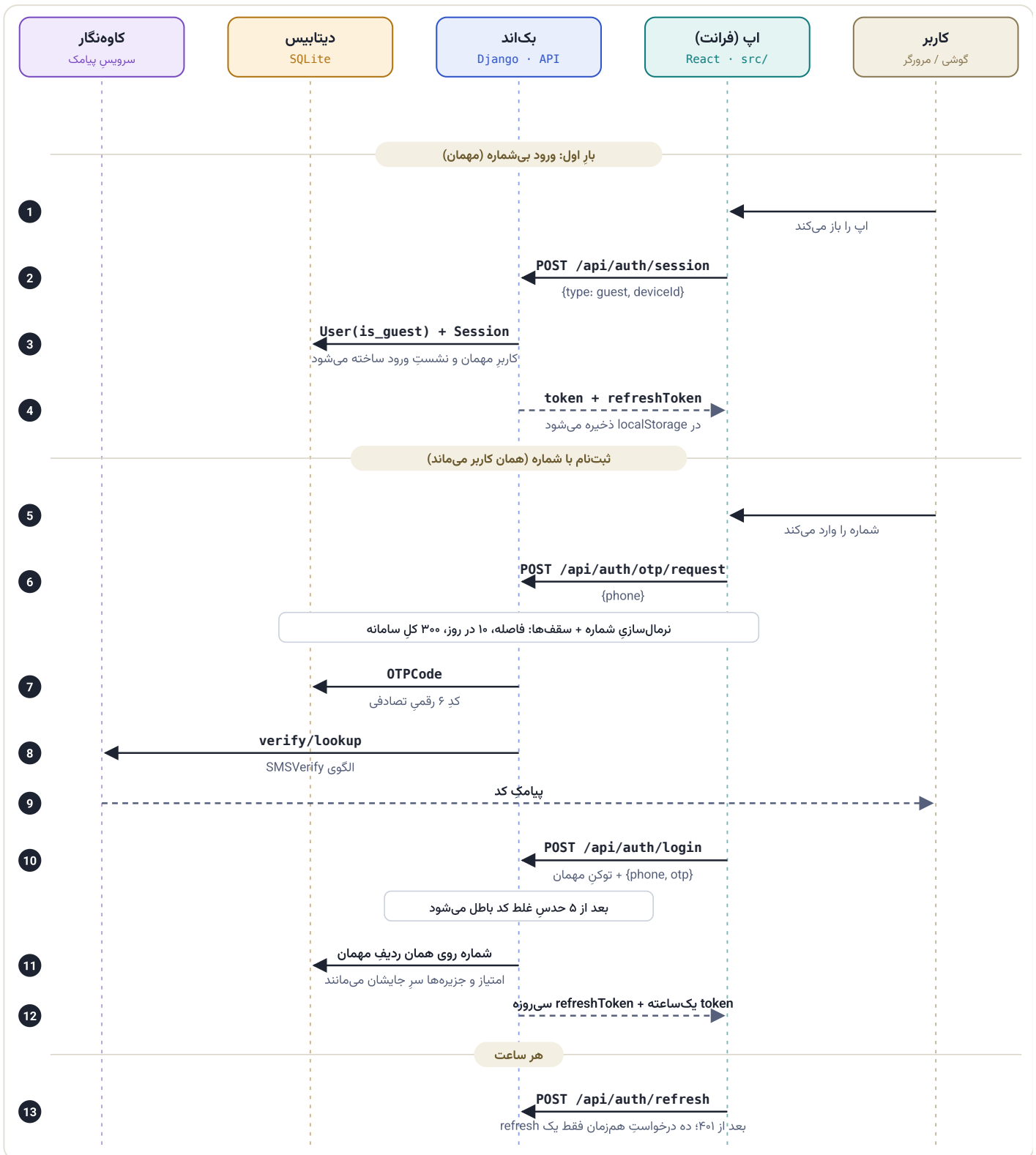
خلاصه‌ی مسیر درخواست

صفحه تابع API را صدا می‌زند؛ `http.ts` توکن را می‌گذارد و می‌فرستد. از آروان و `nginx` به `gunicorn` می‌رسد. جنگو از `urls.py` ویو را پیدا می‌کند، توکن در `authentication.py` سنجیده می‌شود، ویو مجوز و `middleware` ها رد می‌کند، `serializer` ورودی را، و منطق در یک تراکنش اتمیک در دیتابیس می‌نویسد. پاسخ JSON است و برای هر درخواست یک خط لاگ JSON ثبت می‌شود.

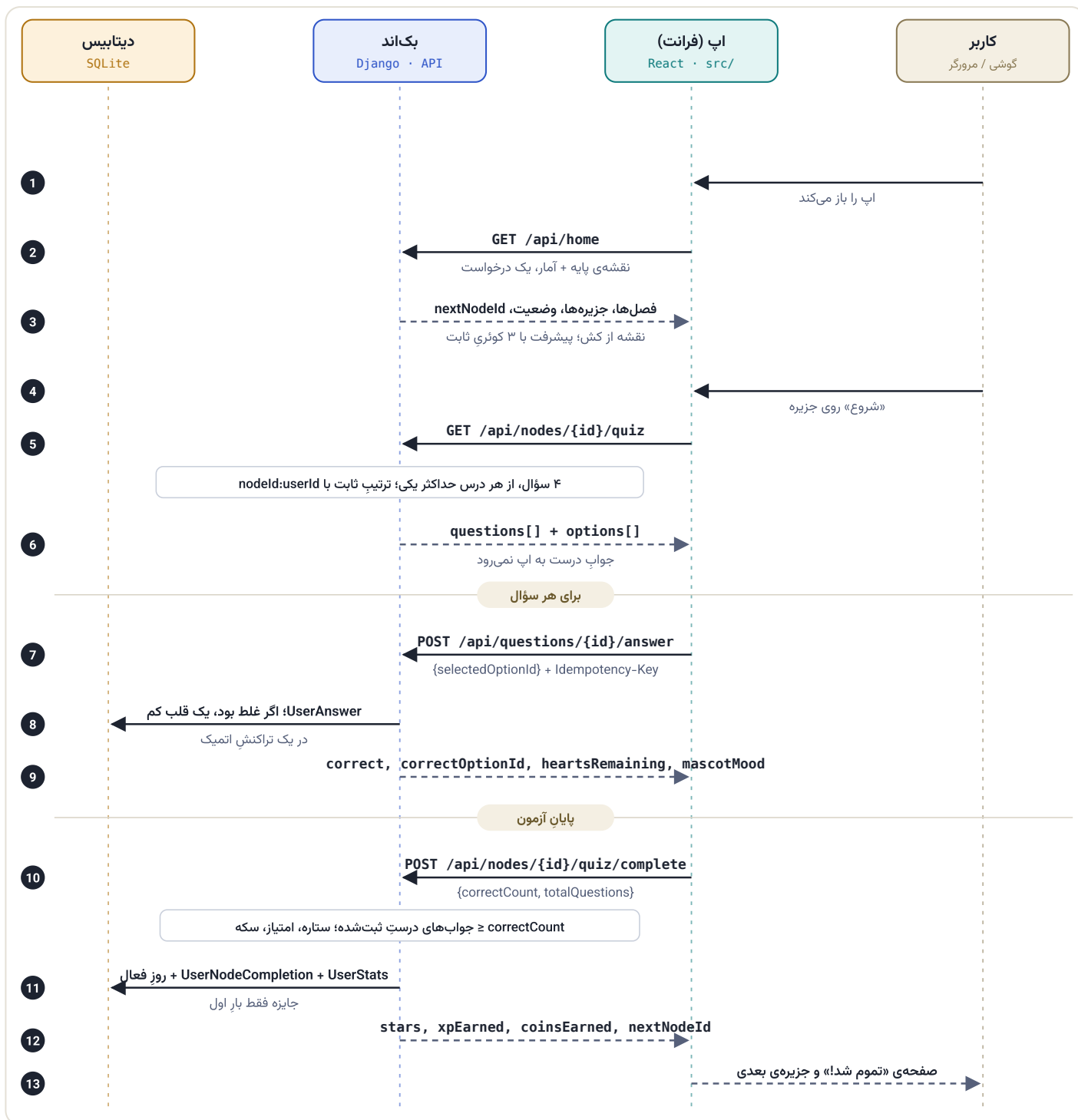
۶ سناریوهای اصلی

هر نمودار از راست به چپ خوانده می‌شود: ستون‌ها بازیگرها هستند، پیکان‌ها درخواست‌ها. پیکان خط‌چین یعنی پاسخ. شماره‌ها ترتیب زمانی‌اند.

۶-۱ ورود: مهمان، بعد ثبت نام با شماره

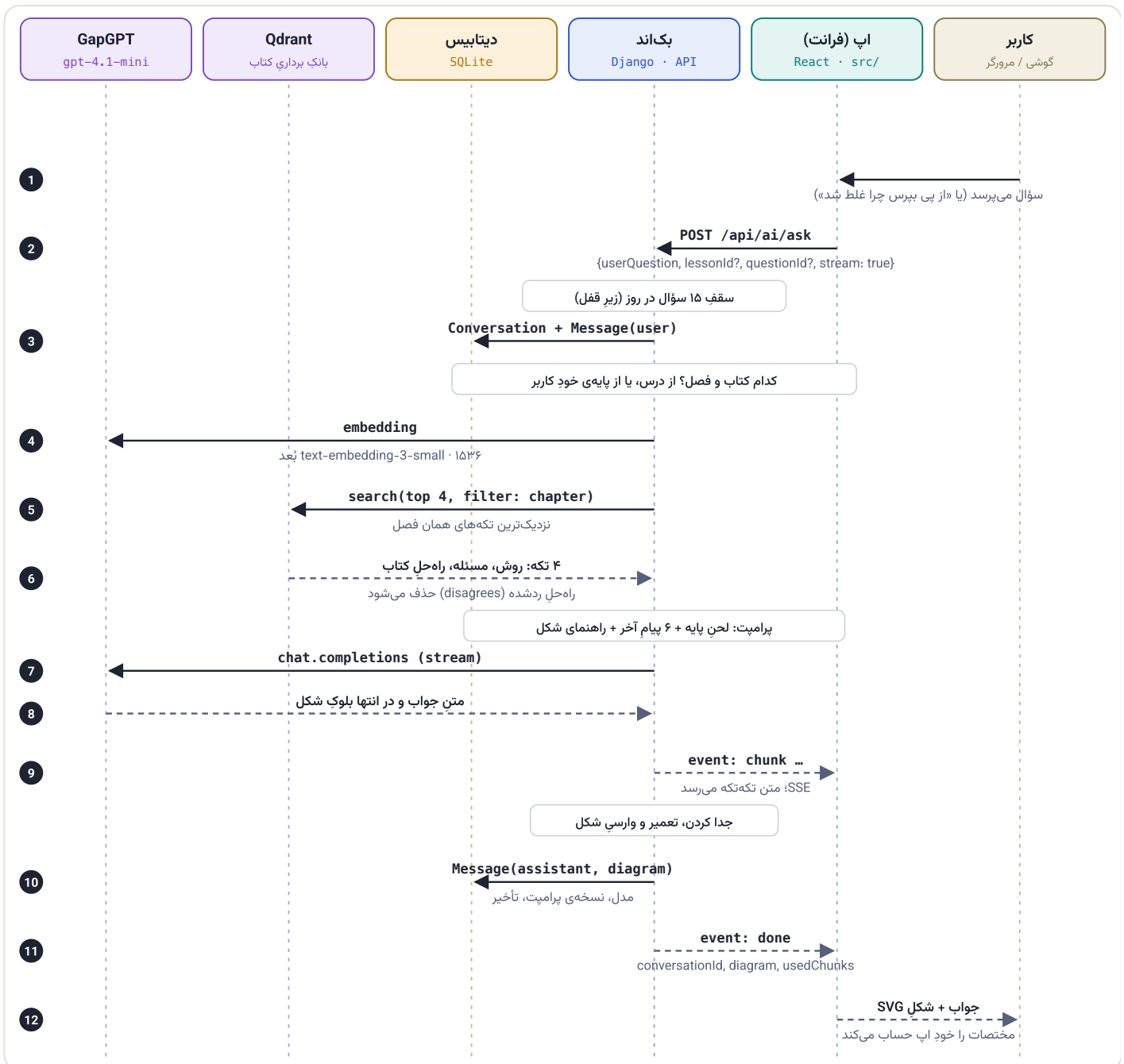


کد: `accounts/views.py` · `accounts/phone.py` (نرمال سازی شماره) · `accounts/services.py` (ارسال پیامک) · `accounts/authentication.py` (توکن) · تست ها: `accounts/tests_otp_limits.py`



کد: `content/views.py` (`NodeQuizCompleteView` ، `QuestionAnswerView` ، `NodeQuizView` ، `HomeView`) · فرانت: `screens/QuizScreen.tsx` ، `screens/HomeScreen.tsx`

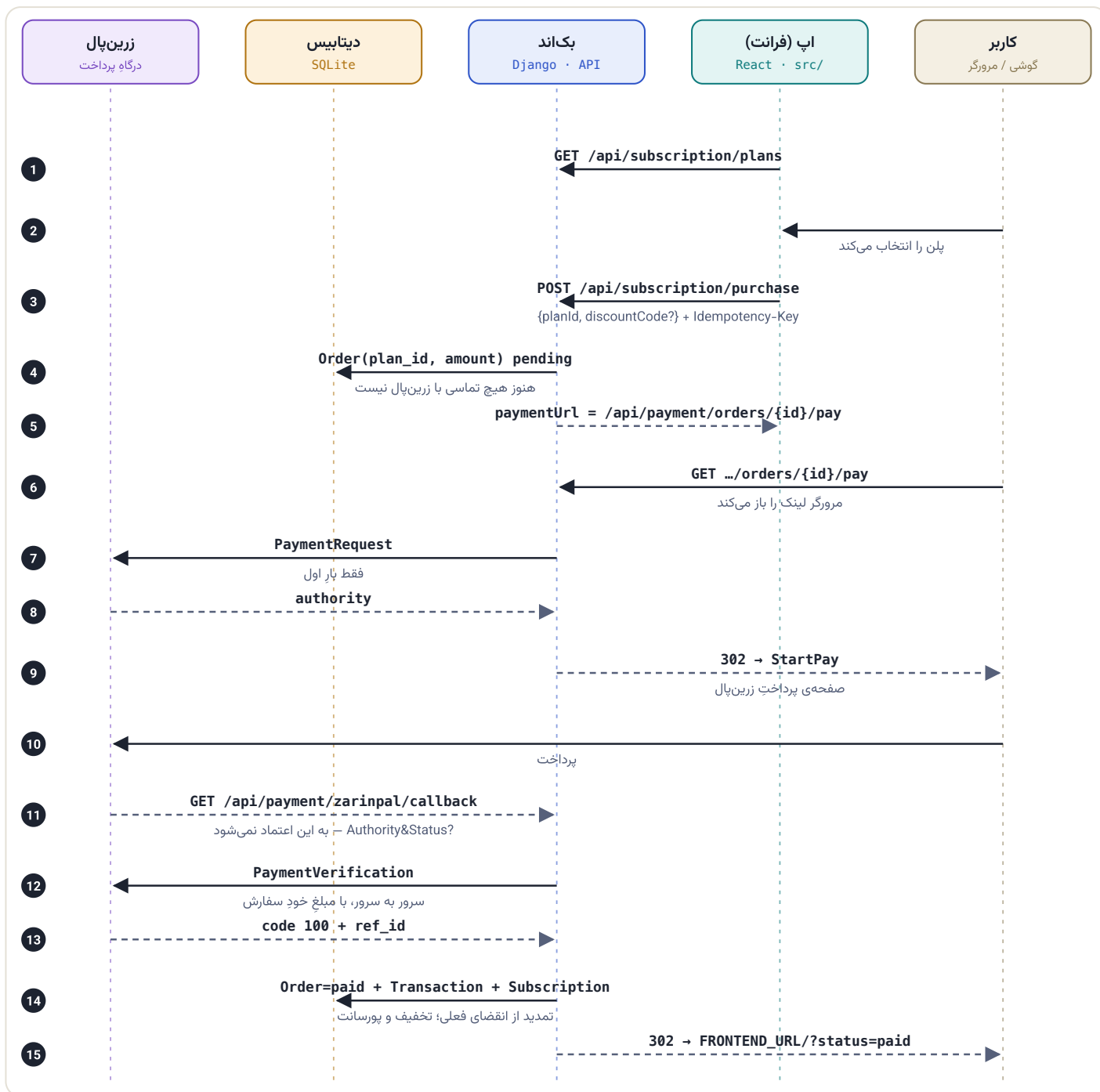
۳-۶ «از پی بی‌پرس» – دستیار با کتاب



کد: `rag/services/retrieval.py` · `rag/services/lesson_context.py` · `ai_tutor/views.py` → `AskAIView`
 فرانت: `rag/services/diagram.py` · `rag/services/generation.py` · `rag/services/qdrant_store.py`
`diagram/` ، `screens/AskScreen.tsx`

سه نکته‌ی طراحی: (۱) تماس با GapGPT بیرون از تراکنش دیتابیس است، وگرنه ۳۰ ثانیه انتظار دیتابیس را قفل می‌کرد؛ (۲) اگر GapGPT خطا داد، سهمیه‌ی رزرو شده پس داده می‌شود و کاربر ۵۰۳ می‌گیرد، نه ۵۰۰؛ (۳) اولین پیام هر گفتگو کش می‌شود – سؤال تکراری از کش ۰/۱ ثانیه‌ای برمی‌گردد در برابر ۱۱ ثانیه.

۴-۶ خرید اشتراک با زرین پال



کد: `subscription/views.py` · `payment/services.py` · `payment/views.py` (`PaymentRedirectView`)
`payment/zarinpal.py` · تست‌ها: `payment/tests_audit_fixes.py` (`ZarinpalCallbackView`)

چرا این قدر پیچیده؟

- ساختِ سفارش به زرین پال وابسته نیست: اگر زرین پال قطع باشد، سفارش ساخته می‌شود و فقط رفتن به درگاه عقب می‌افتد.
 - برگشتِ کاربر هیچ چیزی را ثابت نمی‌کند: هر کس می‌تواند آدرس برگشت را با `Status=OK` بسازد. فقط تأیید سرور-به-سرور سفارش را «پرداخت‌شده» می‌کند.
 - پرداختِ جاافتاده بازسنجی می‌شود: اگر زرین پال موقع برگشت جواب نداد، هر بار که اپ وضعیت اشتراک را می‌خواند سفارش دوباره سنجیده می‌شود (و دستور `payment_reconcile`).
 - پلن روی خودِ سفارش است؛ کد تخفیف ۱۰۰٪ بی‌درگاه تمام می‌کند؛ تمدید از تاریخ انقضای فعلی حساب می‌شود.
- آزموده‌نشده زرین پال درخواست پرداخت را با کد ۱۰۰ می‌پذیرد، ولی یک پرداخت واقعی تا انتها هنوز انجام نشده.

۵-۶ والد

#	درخواست	چه می‌شود
۱	<code>POST /api/family/link/invite</code>	والد یک کد اتصال می‌گیرد (سقف ۵ در ساعت).
۲	<code>POST /api/family/link/accept</code>	فرزند کد را در اپ خودش وارد می‌کند؛ <code>FamilyLink</code> «accepted» می‌شود. یک والد می‌تواند چند فرزند داشته باشد.
۳	<code>GET /api/family/weekly-report</code>	امتیاز، درس، جزیره و زنجیره‌ی فرزند از شنبه.
۴	<code>GET /api/progress/nodes?userId=</code>	فقط وقتی مجاز است که اتصال تأییدشده وجود داشته باشد؛ وگرنه ۴۰۳. (نمونه‌ی جلوگیری از دیدن داده‌ی دیگران.)

۷ مدل داده

جدول‌های اصلی، دسته‌بندی‌شده بر اساس اپ. → یعنی کلید خارجی واقعی (داخل همان اپ)، و `user_id` یعنی ارجاع سبک به اپ دیگر.

content	محتوا	هویت
فعالیت کاربر		
<code>UserAnswer</code> <code>user_id</code> , → <code>Question</code> , → <code>Choice</code>	شماره‌ی پایه (۸ = <code>id</code>)	<code>User</code> <code>phone</code> , <code>name</code> , <code>is_guest</code> , <code>device_id</code> , <code>grade_id</code> , <code>tokens_valid_after</code>
<code>UserLessonCompletion</code> <code>user_id</code> , → <code>Lesson</code>	<code>Unit</code> → <code>Grade</code> · <code>title</code> , <code>blurb</code> , <code>order</code>	<code>Profile</code> → <code>User</code> · <code>birth_year</code> , <code>parent_phone</code>
<code>UserNodeCompletion</code> <code>user_id</code> , → <code>Node</code> , <code>stars</code> , <code>xp_earned</code> , <code>coins_earned</code>	<code>Node</code> → <code>Unit</code> · <code>order</code> (جزیره)	<code>Session</code> → <code>User</code> · <code>refresh_token</code> , <code>expires_at</code> , <code>revoked</code>
<code>NodeProgress</code> <code>user_id</code> , → <code>Node</code> , <code>status</code>	<code>Lesson</code> → <code>Node</code> · <code>kind</code> , <code>example</code> , <code>steps</code>	<code>OTPCode</code> <code>phone</code> , <code>code</code> , <code>expires_at</code> , <code>failed_attempts</code>
<code>QuestionReport</code> این سؤال ایراد دارد	<code>Question</code> → <code>Lesson</code> · <code>prompt</code> , <code>type</code>	
	<code>Choice</code> → <code>Question</code> · <code>value</code> , <code>is_correct</code>	

subscription · payment پول	ai_tutor · rag دستیار	progress بازی
Plan price, duration_days, features, badge Subscription user_id, → Plan, status, expires_at Order user_id, plan_id, amount, status, gateway_authority Transaction → Order · gateway_ref_id, status, paid_at BazaarCredential توکن اتصال به بازار	Conversation user_id, lesson_id, grade_id Message → Conversation · role, text, diagram, model_name, prompt_version, latency_ms MessageFeedback → Message · is_helpful LessonChunk problem, solution, solution_verdict, qdrant_point_id IngestedPage collection, page	UserStats user_id, xp, coins, selected_character_id UserHearts current_hearts, max_hearts=5, next_refill_at, full_refill_at StreakRecord current_streak, longest_streak ActiveDay → StreakRecord, date LeaderboardEntry period, xp_in_period, rank Character slug, name, unlock_xp
family · campaigns · statuslog	support · notifications ارتباط	discount · affiliate بازاریابی
FamilyLink parent_user_id, child_user_id, status ParentalSettings , WeeklyReportLog MessageCampaign , CampaignDelivery × کمپین × کاربر × کانال StatusChangeLog label, object_id, old_status → new_status	Ticket user_id, subject, status TicketMessage → Ticket · sender, text BugReport Notification user_id, title, body, is_read	DiscountCode code, type, value, max_uses Redemption یکتا: هر کاربر × هر کد Offer / UserOffer پیشنهاد خودکار AffiliateAccount user_id, referral_code, commission_rate Conversion → AffiliateAccount · order_id, commission_amount

فهرست فیلدها خلاصه است؛ منبع کامل `<app>/models.py` است. نام‌ها و ارتباط‌ها از کد خوانده شده‌اند.

قیدهایی که در خود دیتابیس گذاشته شده‌اند (نه فقط در کد)

از چه جلوگیری می‌کند	قید
ثبت دوباره‌ی یک جواب با تکرار درخواست	یکتا: <code>UserAnswer(user_id, question, idempotency_key)</code>
استفاده‌ی دوباره از یک کد تخفیف	یکتا: <code>Redemption</code> (کاربر، کد)
پیام تکراری وقتی کمپین دو بار همزمان اجرا شود	یکتا: <code>CampaignDelivery</code> (کمپین، کاربر، کانال)
تکه‌ی تکراری در Qdrant وقتی خواندن کتاب از سر گرفته می‌شود	یکتا: <code>LessonChunk(collection, qdrant_point_id)</code> و <code>IngestedPage(collection, page)</code>

کد و محتوا دو مسیر جدا دارند. `db.sqlite3` در git نیست. محتوا (برنامه‌ی درسی و سؤال‌ها) به‌صورت JSON در `content/seed/` است و `deploy.sh` در هر انتشار آن را وارد می‌کند. داده‌ی کاربرها فقط در فایل دیتابیس سرور است.

API : قواعد و فهرست کامل

مستند زنده و قابل امتحان: <https://mathapi.tadrisino2.ir/api/docs/> (Swagger). جدول‌های زیر از همان اسکیمای استخراج شده‌اند.

قواعد مشترکِ همه‌ی اندپوینت‌ها

موضوع	قاعده
نشانی پایه	<code>https://mathapi.tadrisino2.ir/api/...</code> — دامنه در کد نیامده و از تنظیمات (<code>PUBLIC_BASE_URL</code>) خوانده می‌شود.
ورود	هدر <code>Authorization: Bearer <token></code> . توکن یک ساعت اعتبار دارد و با <code>POST /api/auth/refresh</code> تازه می‌شود.
قالب داده	JSON؛ نام فیلدها <code>camelCase</code> (<code>selectedOptionId</code>)، شناسه‌ها در پاسخ رشته‌اند.
خطا	همیشه JSON با <code>detail</code> (و گاهی <code>code</code> مثل <code>no_hearts</code>): <code>۴۰۰</code> ورودی بد، <code>۴۰۱</code> توکن نامعتبر، <code>۴۰۳</code> اجازه نداری، <code>۴۰۴</code> ، <code>۴۲۹</code> زیاد درخواست دادی، <code>۵۰۳</code> سرویس بیرونی در دسترس نیست.
تکرار امن	هدر اختیاری <code>Idempotency-Key</code> روی خرید، دستیار، جواب، پایان درس و جزیره: درخواست تکراری با همان کلید، همان نتیجه‌ی قبلی را برمی‌گرداند.
سقف نرخ	هر اندپوینت حساس یک <code>throttle_scope</code> دارد؛ نرخ‌ها همه یک‌جا در <code>config/settings.py → DEFAULT_THROTTLE_RATES</code> .
استریم	<code>POST /api/ai/ask</code> با <code>"stream": true</code> پاسخ را SSE می‌فرستد: رویدادهای <code>chunk</code> ، <code>done</code> ، <code>error</code> .

فهرست ۶۱ اندپوینت

عمومی · بی‌ورود · ورود · توکن لازم · داخلی · با هدر کلید داخلی (سرور-به-سرور)

روش	مسیر	دسترسی	کار
هویت و پروفایل — <code>accounts</code>			
POST	<code>/api/auth/session</code>	عمومی	ساخت کاربرِ مهمان با <code>deviceId</code>
POST	<code>/api/auth/otp/request</code>	عمومی	فرستادن کد ورود
POST	<code>/api/auth/login</code>	عمومی	ورود با شماره و کد
POST	<code>/api/auth/refresh</code>	عمومی	توکن تازه با <code>refresh token</code>
POST	<code>/api/auth/logout</code>	ورود	خروج؛ توکن‌های قبلی باطل
GET · PATCH	<code>/api/user/profile</code>	ورود	خواندن و ویرایش پروفایل

روش	مسیر	دسترسی	کار
POST	/api/user/grade	ورود	ثبت پایه
محتوا و آزمون – content			
GET	/api/grades	عمومی	فهرست پایه‌ها
GET	/api/home	ورود	نقشه‌ی جزیره‌ها + آمار، در یک درخواست
GET	/api/nodes/{id}/lessons	عمومی	درس‌های یک جزیره
GET	/api/nodes/{id}/quiz	عمومی	آزمون ۴ سؤال‌ی جزیره
POST	/api/nodes/{id}/quiz/complete	ورود	پایان جزیره؛ ستاره، امتیاز، سکه، جزیره‌ی بعد
GET	/api/lessons/{id}/intro	عمومی	مثال و قدم‌های درس
GET	/api/lessons/{id}/questions	عمومی	سؤال‌های یک درس
POST	/api/lessons/{id}/complete	ورود	پایان درس
POST	/api/questions/{id}/answer	ورود	ثبت جواب
POST	/api/questions/{id}/report	ورود	گزارش ایراد سؤال
GET	/api/progress/nodes	ورود	وضعیت کاربر (یا فرزند) روی جزیره‌ها
پیشرفت و بازی – progress			
GET	/api/user/summary	ورود	پروفایل + آمار + زنجیره در یک درخواست
GET	/api/user/hearts	ورود	قلب‌ها و زمان پر شدن
GET	/api/user/streak/calendar	ورود	تقویم روزهای فعال
GET	/api/leaderboard	عمومی	جدول رتبه: هفتگی یا همه‌ی زمان‌ها
GET	/api/characters	ورود	همراه‌ها و باز/قفل بودنشان
POST	/api/characters/select	ورود	انتخاب همراه (فقط بازشده)
دستیار – ai_tutor و rag			
POST	/api/ai/ask	ورود	پرسش؛ JSON یا استریم
GET	/api/ai/conversations/{id}/messages	ورود	تاریخچه‌ی گفتگو
POST	/api/ai/messages/{id}/feedback	ورود	👍 / 👎 روی یک جواب
POST	/api/rag/ingest/text · /pdf	داخلی	افزودن متن یا PDF به Qdrant
GET	/api/rag/chunks	داخلی	مرور تکه‌های ذخیره‌شده
اشتراک، پرداخت، تخفیف			
GET	/api/subscription/plans	عمومی	پلن‌ها
POST	/api/subscription/purchase	ورود	شروع خرید؛ سفارش و لینک پرداخت
GET	/api/subscription/status	ورود	وضعیت اشتراک (و بازسنجی پرداخت جافتاده)
POST	/api/payment/orders	ورود	ساخت سفارش

روش	مسیر	دسترسی	کار
GET	/api/payment/orders/{id}	ورود	وضعیت سفارش
GET	/api/payment/orders/{id}/pay	عمومی	رفتن به درگاه زرین پال
GET	/api/payment/zarinpal/callback	عمومی	برگشت از درگاه + تأیید سرور-به-سرور
POST	/api/payment/bazaar/verify	ورود	تأیید خرید کافه بازار
GET	/api/discount/validate	ورود	سنجش کد تخفیف
POST	/api/discount/redeem	ورود	ثبت مصرف کد
GET · POST	/api/discount/offers/mine · /redeem	ورود	پیشنهادهای کاربر
اجتماعی و پشتیبانی			
GET · POST	/api/affiliate/me · /track · /conversions	ورود	کد معرف، ثبت معرف، پورسانت‌ها
GET	/api/notifications	ورود	اعلان‌ها
PATCH	/api/notifications/{id}/read · /read-all	ورود	خوانده‌شده
GET · POST	/api/support/tickets	ورود	تیکت‌های من / تیکت تازه
GET · POST	/api/support/tickets/{id} · /messages	ورود	جزئیات و پیام تازه
POST	/api/support/bug-reports	ورود	گزارش باگ
GET · POST · PATCH	/api/family/link · /invite · /accept · /settings · /weekly-report	ورود	اتصال والد، تنظیمات، گزارش هفتگی
سیستم			
GET	/api/health	عمومی	سلامت سرویس (بخش ۱۳)

۹ دستیار هوش مصنوعی و داده‌ی Qdrant

دستیار «از حافظه‌ی مدل» جواب نمی‌دهد؛ اول تکه‌های مرتبط همان کتاب و همان فصل را پیدا می‌کند و مدل جواب را بر پایه‌ی آن‌ها می‌نویسد. به این روش RAG (Retrieval-Augmented Generation) می‌گویند.

۹-۱ Qdrant چیست و چه چیزی در آن است

Qdrant یک «بانک برداری» است: هر تکه‌ی کتاب به یک بردار (۱۵۳۶ عدد) تبدیل می‌شود که «معنا»ی آن را نشان می‌دهد؛ سؤال دانش‌آموز هم به بردار تبدیل می‌شود و نزدیک‌ترین تکه‌ها پیدا می‌شوند. هر **collection** یک کتاب است و هر **نقطه (point)** یک تکه — معمولاً یک تیپ‌مسئله از یک صفحه، با روش حل، صورت مسئله و راه‌حل کتاب.

collection (کتاب)	تکه	صفحه‌ی خوانده‌شده	چه کسی ساخت
ریاضی هشتم	۵۱۸	۱۵۰	ما – خواندن PDF با مدلِ بینایی
ریاضی نهم	۴۵۰	۱۴۳	ما
ریاضی دهم	۴۵۸	۱۷۰	ما
حسابان یازدهم	۳۷۶	۱۵۱	ما
حسابان دوازدهم	۳۲۶	۱۴۶	ما
ریاضی سوم · ششم · هفتم	۰ ۴۳۶ ۰ ۴۳۷ ۳۷۹	—	تیمِ دیگرِ شرکت؛ با rag_import_collection خوانده شد
علوم تجربی سوم · مطالعات اجتماعی ششم و چهارم	۰ ۴۰۰ ۷۰ ۲۷	—	تیمِ دیگر (در اپ ریاضی استفاده نمی‌شود)
جمع	۳,۵۱۷	۷۶۰	نشانی: qdrant.tadrisino2.ir – سرویسِ مشترک با تیمِ دیگر

مشخصاتِ فنی

مدلِ بردار (embedding)	text-embedding-3-small
ابعاد و فاصله	۱۵۳۶ · کسینوسی
تعدادِ تکه در هر جست‌وجو	۴ (RAG_RETRIEVAL_TOP_K)
فیلترِ ریاضی	فیلدِ chapter = «فصل ۲: عددهای اول»
نامِ collection	"{پایه} {درس}" مثلِ «ریاضی نهم»

شکلِ یک نقطه (payload)

```
{ "chapter": "فصل ۸: حجم و مساحت",
  "grade": "نهم", "type": "تعریف کره",
  "keywords": ["کره", "شعاع", "مرکز"],
  "generalSolution": "تعریف دایره را ۱۰.",
  "problem": "...", "solution": "...",
  "solution_verdict": "agrees",
  "source_page": 139, "chapter_number": 8 }
```

ساخته در rag/services/ingest.py → _payload

۲-۹ داده چطور وارد شد: از PDF تا Qdrant



قاعده‌ی مهم پرامپت (`rag/services/pdf.py`): اگر مسئله به «شکل زیر» ارجاع می‌دهد، مدل باید اندازه‌ها و برچسب‌های شکل را داخل متن بنویسد. این قاعده سهم مسئله‌های بی‌استفاده‌ی وابسته به شکل را از ~۴۲٪ به ~۱۴٪ رساند. هزینه‌ی خواندن هر پنج کتاب در یک روز حدود ۳/۴۴ دلار بود.

۳-۹ کنترل کیفیت: راستی‌آزمایی راه‌حل‌های کتاب

مدل بینایی گاهی عدد را بد می‌خواند یا راه‌حل را به مسئله‌ی کناری می‌چسباند. دستور `rag_verify_solutions` هر مسئله را مستقل دوباره حل می‌کند و رأی سه‌حالتی می‌دهد:

رأی	تعداد	اثر
agrees راه‌حل کتاب درست است	۲,۴۶۶	فقط از این‌ها سؤال ساخته می‌شود
disagrees راه‌حل کتاب غلط است	۳۵۴	راه‌حلش به دستیار داده نمی‌شود؛ فقط روش و صورت مسئله
unjudgeable بی‌شکل قابل قضاوت نیست	۵۰۸	کنار می‌ماند
بررسی نشده	۱۸۹	علوم و مطالعات تیم دیگر

متن اصلی کتاب هیچ‌وقت بازنویسی نمی‌شود؛ جواب مستقل جدا در `checked_answer` ذخیره می‌شود تا آدم بتواند داوری کند.

۴-۹ افزودن و پاک کردن داده

کار	روش	هزینه
افزودن کتاب تازه	۱) PDF را در <code>data/textbooks/</code> بگذارید و فصل‌ها و <code>page_offset</code> را در <code>rag/services/textbooks.py</code> ثبت کنید. ۲) <code>python manage.py rag_ingest_book --grade نهم --plan</code> فقط نشان می‌دهد چه چیزی و با چه هزینه‌ای خوانده می‌شود. ۳) همان دستور بدون <code>--plan</code> (قابل قطع و ازسرگیری؛ صفحه‌ی خوانده‌شده دوباره خریده نمی‌شود). ۴) <code>rag_verify_solutions</code> (۵ ← <code>seed_curriculum_from_textbook</code> (۶ ← ساخت سؤال (بخش ۱۰).	~۰/۲۲ سنت / صفحه
افزودن یک متن یا PDF	<code>POST /api/rag/ingest/text</code> یا <code>/pdf</code> با هدر <code>X-Internal-Key</code> .	کم
آوردن collection تیم دیگر	"ریاضی ششم" <code>rag_import_collection --collection</code> – فقط می‌خواند، چیزی در Qdrant نمی‌نویسد.	رایگان
بازخوانی صفحه‌های معیوب	<code>rag_ingest_book --grade نهم --refresh figures</code> یا <code>--refresh-pages</code> 12, 15, 20-24. هر صفحه اول پاک می‌شود (اول Qdrant، بعد دیتابیس – تابع <code>clear_page</code>) و دوباره خوانده می‌شود.	مثلی خواندن
پاک کردن	دستور جداگانه‌ی «فقط پاک کن» نداریم. برای یک صفحه تابع <code>rag.services.ingest.clear_page(collection, page)</code> از <code>manage.py shell</code> برای کل collection، <code>delete_collection</code> کلاینت Qdrant – فقط با هماهنگی تیم دیگر، چون Qdrant مشترک است.	–

دو احتیاط (از خواندن کد؛ آزموده نشده)

- پاک کردن یک LessonChunk از پنل ادمین نقطه‌ی Qdrant آن را پاک نمی‌کند (هیچ شنونده‌ای برایش نیست)؛ نقطه‌ی صاحب می‌ماند و در جست‌وجو پیدا می‌شود. از clear_page استفاده کنید.
- ردیف‌های LessonChunk (نسخه‌ی ساختاریافته با بردار) در دیتابیس محلی توسعه‌اند، نه روی سرور؛ سرور فقط از Qdrant می‌خواند. از آن فایل بکاپ جدا نگه دارید – با آن، Qdrant بی‌هزینه‌ی embedding دوباره ساخته می‌شود.

۵-۹ از سؤال تا جواب (خلاصه‌ی سناریوی ۶-۳)

مرحله	فایل	نکته
کدام کتاب؟	rag/services/lesson_context.py	از lessonId زنجیره‌ی درس → جزیره → فصل → پایه؛ یا پایه‌ی خود کاربر
جست‌وجو	retrieval.py embedding.py qdrant_store.py	متن جست‌وجو = سؤال + سؤال قبلی کاربر (برای «نفهمیدم»)
پرامپت	generation.py	نسخه‌ی tutor-v10-shape-anchor؛ لحن سه‌سطحی (دبستان/متوسطه‌ی اول/دبیرستان)؛ متن ساده بی‌LaTeX؛ حل مرحله‌به‌مرحله با واری و «✓» جواب؛ ۶ پیام آخر گفتگو
جواب درست/غلط دانش‌آموز	generation.py → question_focus	با questionId دستیار صورت سؤال و آخرین جواب دانش‌آموز را می‌بیند و می‌گوید اشتباه کجا بود
تمیزکاری	mathtext.py	هرچه LaTeX از دست مدل در برود به یونیکد برمی‌گردد ($\sqrt{3}$ ، x^2)
شکل	diagram.py ← فرانت src/diagram/	مدل فقط توصیف JSON می‌نویسد؛ سرور جدا، تعمیر و با فهرست سفید واری می‌کند؛ اپ مختصات را حساب و SVG رسم می‌کند. ۱۵ نوع شکل – شرح کامل در کادر بنفش پایین.
ثبت	ai_tutor/models.py → Message	مدل، نسخه‌ی پرامپت، تعداد تکه، تأخیر و شکل برای هر جواب ذخیره می‌شود ← کیفیت قابل اندازه‌گیری است

مهم‌ترین ایده‌ی فنی دستیار

شکل را هوش مصنوعی نمی‌کشد: مدل می‌گوید «چه شکلی»، اپ آن را با فرمول می‌کشد

هوش مصنوعی ریاضی را می‌فهمد ولی در حساب دقیق مختصات اشتباه می‌کند؛ کامپیوتر برعکس، هیچ‌وقت در حساب اشتباه نمی‌کند ولی مسئله را نمی‌فهمد. پس کار بین این دو تقسیم شد: مدل فقط توصیف شکل را به صورت داده (JSON) می‌نویسد، سرور آن را واری می‌کند، و کد اپ رأس‌ها، ریشه‌ها و زاویه‌ها را خودش حساب و رسم می‌کند.

۱

سرور → مدل

سرور به مدل می‌گوید اگر شکل لازم است، فقط JSON بنویسد – و فقط بخش مربوط از راهنمای شکل‌ها را کنار سؤال می‌گذارد (برای «۳۴۵ × ۱۲» هیچ).

`rag/services/diagram.py`

متن واقعی قاعده (DIAGRAM_RULE): «در پایان پاسخ دقیقاً یک بلوک `<diagram>...</diagram>` بگذار که داخلش فقط یک JSON است، نه SVG و نه HTML. برای حساب عددی یا ساده‌کردن عبارت جبری که شکلی ندارد، شکل نگذار.»

۲

مدل

مدل حلِ قدم‌به‌قدم را می‌نویسد و در آخر فقط توصیفِ شکل (۱۵۰۰ توکن، نه صدها خط کدِ تصویر):

```
جواب: ✓ ... BC = 10
<diagram>{"type":"triangle", "names":["A","B","C"],
"sides":{"AB":6,"AC":8}, "angles":{"A":90}, "labels":{"BC":"x"}}
</diagram>
```

۳

سرور

سرور بلوک را از متن جدا، اشتباه‌های قالبی مدل را تعمیر، و با فهرست سفید واری می‌کند (رقم فارسی در عدد، 2π ، کامنت، گیومه‌ی جاافتاده؛ نوع و رنگ و کلید ناشناخته رد؛ عددها محدود). شکل خراب فقط حذف می‌شود و جوابِ متنی سر جایش می‌ماند. نتیجه در `Message.diagram` ذخیره و در رویداد `done` فرستاده می‌شود. کار سرور ~۰/۴ میلی‌ثانیه.

```
# rag/services/diagram.py
def extract(text) -> tuple[str, dict | None]:
    """(text without the block, spec or None)"""
    match = _BLOCK.search(text) # <diagram>(.*?)
    </diagram>
    spec = parse(match.group(1)) if match else None # _repair ->
    json -> _normalize -> validate
    return _BLOCK.sub('', text).rstrip(), spec
```

۴

اپ (گوشی)

اپ هر عددی را که قابلِ حساب است خودش حساب می‌کند و SVG می‌کشد: رأس‌های مثلث با قانون کسینوس‌ها، ریشه‌ی نمودار با نصف‌کردن بازه تا دقت 10^{-11} . رسم روی گوشی خودِ دانش‌آموز انجام می‌شود، پس هزار کاربر یعنی بارِ صفر روی سرور.

`src/diagram/expr.ts` `src/diagram/shapes.ts`

```
// src/diagram/shapes.ts – law of cosines: third side from two
sides + angle
s[z] = Math.sqrt(s[x] ** 2 + s[y] ** 2 - 2 * s[x] * s[y] *
Math.cos(g[opp[z]] * RAD));
// src/diagram/expr.ts – roots: scan 800 points, bisection where the
sign flips
if (Math.sign(y) !== Math.sign(py)) { const r = bisection(f, px, x);
if (Math.abs(f(r)) < 1e-6) out.push(r); }
```

مثال ۱. مثلث قائم الزاویه

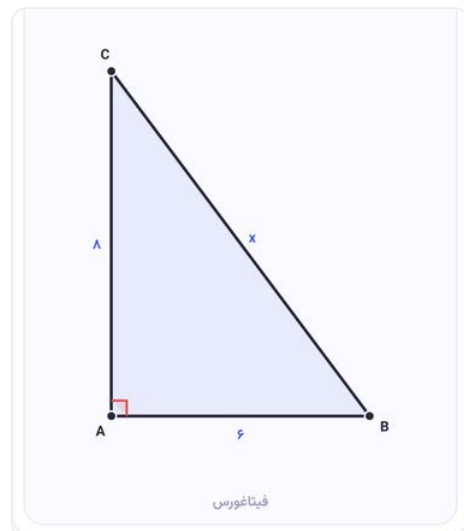
مدل می نویسد

```
{
  "type": "triangle",
  "names": ["A", "B", "C"],
  "sides": {"AB": 6, "AC": 8},
  "angles": {"A": 90},
  "labels": {"BC": "x"}
}
```

اپ حساب می کند

قانون کسینوس ها: $BC^2 = 6^2 + 8^2 - 2 \cdot 6 \cdot 8 \cdot \cos 90^\circ = 100$
پس $BC = 10$ و رأس ها $A(0,0) \cdot B(6,0) \cdot C(0,8)$

دانش آموز می بیند



مثال ۲. ریشه و رأس سهمی

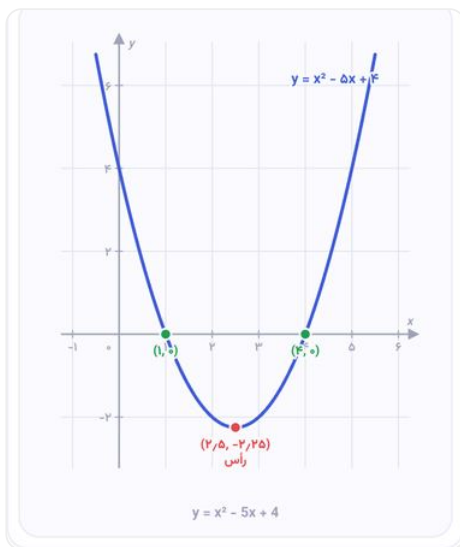
مدل می نویسد

```
{
  "type": "function",
  "expr": "x^2-5*x+4",
  "marks": ["roots", "vertex"]}
}
```

اپ حساب می کند

۸۰۰ نقطه بررسی می شود؛ هر جا علامت عوض شد، با نصف کردن بازه ریشه پیدا می شود: $x = 1$ و $x = 4$
جایی که شیب صفر است، رأس: $(2.5, -2.25)$

دانش آموز می بیند



روش رایج: مدل خودش SVG/HTML بنویسد	روش ما: مدل توصیف، اپ رسم
امنیت	کد مدل در صفحه اجرا می‌شود؛ متن مخرب در سؤال می‌تواند کد تزریق کند
دقت	هیچ کدی از مدل اجرا نمی‌شود؛ فقط JSON واریسی شده
هزینه	مختصات را حدس می‌زند؛ پنج ضلعی کج، ریشه جابه‌جا
اگر خراب شد	چند برابر توکن برای هر شکل
	۱۵۰۰ توکن؛ برای سؤال بی‌شکل صفر (۳/۰۰۰ تا ۸/۰۰۰ دلار در هر جواب شکل‌دار)
	فقط شکل نمی‌آید؛ جواب متنی سالم می‌ماند
	صفحه یا چت به هم می‌ریزد



چت واقعی: الگوی چوب کبریت

چت واقعی: دوزنقه

چت واقعی روی سایت: سهمی

فایل‌ها – بک‌اند: `rag/services/diagram.py` (قاعده، تشخیص، تعمیر، واریسی) · `rag/services/generation.py` (پرامپت، جدا کردن در استریم) · `ai_tutor/models.py → Message.diagram` · تست‌ها `rag/tests_diagram.py` – فرانت: `src/diagram/index.tsx` (انتخاب نوع، محافظ خطا) · `plane.tsx` · `shapes.ts` · `expr.ts`

۹-۶ کیفیت با عدد

سنجش	نتیجه
انتخاب مدل جواب: ۱۰۲ مسئله‌ی تصحیح‌شده‌ی دستی	<code>gpt-4.1-mini</code> : ۹۵-۹۶ درست · <code>gpt-4o-mini</code> : ۸۵-۹۰
شکل‌ها: ۶۴ مسئله‌ی تصادفی کتاب، واریسی چشمی تک‌تک	۴۰ درست، ۱۲ قابل قبول، ۷ غلط، ۵ بی‌شکل – حدود یکی از پنج هنوز ایراد دارد؛ برای همین شکل همیشه کنار حل متنی می‌آید
سرعت	جواب کامل با شکل ~ ۳۰ ثانیه؛ اندازه‌گیری شد که در خود GapGPT است («فقط بگو سلام» = ۷ ثانیه)
سقف‌ها	۱۵ سؤال در روز برای هر کاربر؛ ۲۰ در ساعت؛ مهمان‌ها بر اساس IP (تا با ساخت حساب مهمان تازه دور زده نشود)

۱۰ محتوای آموزشی و ساخت سؤال

هدف: محتوا بدون برنامه‌نویس قابل ویرایش باشد، و یک سرور خالی با یک دستور پر شود.

محتوا کجا زندگی می‌کند

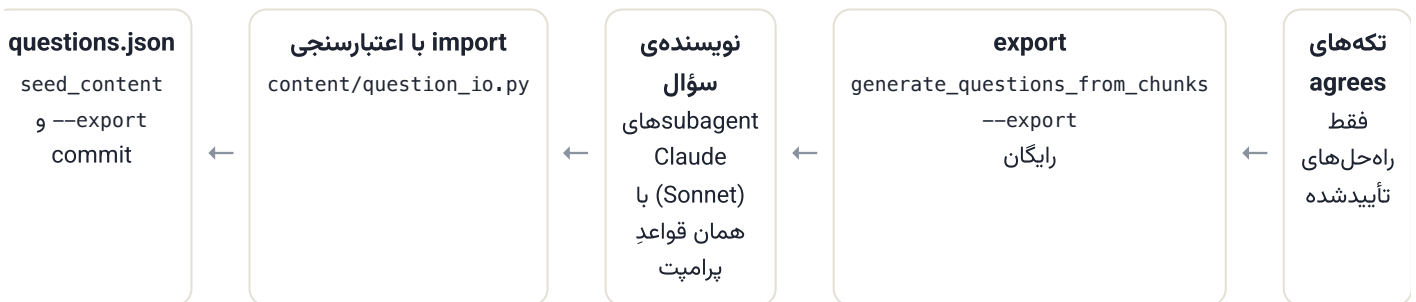
فایل	محتوا
<code>content/seed/curriculum.json</code>	۸ پایه، ۵۸ فصل، ۲۶۳ جزیره، ۲,۶۱۷ درس
<code>content/seed/questions.json</code>	۲,۵۴۵ سؤال چهارگزینیه‌ای

- `seed_content` در هر انتشار اجرا می‌شود؛ هیچ‌وقت چیزی پاک نمی‌کند و اجرای دوباره‌اش بی‌خطر است.
- سؤال با مسیر **عنوان** (پایه ، فصل ، جزیره ، درس) به درس وصل می‌شود، نه با شناسه — چون شناسه‌ها بین دو دیتابیس فرق دارند.
- متنی که ادمین در پنل نوشته با انتشار بعدی بازنویسی نمی‌شود؛ `seed` فقط خانه‌ی خالی را پر می‌کند.

کارهای روزمره از پنل ادمین

کار	کجا
افزودن یا ویرایش یک سؤال	<code>/admin/content/question/</code> ← «افزودن»
واردکردن فله‌ای (همه یا هیچ)	همان صفحه ← «وارد کردن سؤال‌ها از فایل»
خروجی سؤال یا برنامه‌ی درسی	انتخاب ← اکشن «دریافت فایل» (همان قالبی که ورودی می‌پذیرد)
پایه‌ی تازه	<code>/admin/content/grade/add/</code> — عنوان «پایه چهارم»، شناسه خودکار ۴
پلن، همراه، تیکت، کد تخفیف	صفحه‌ی اپ مربوط در پنل
سؤال‌های گزارش‌شده	<code>QuestionReport</code> — دانش‌آموز «این سؤال ایراد دارد» زده

سؤال‌ها چگونه ساخته شدند



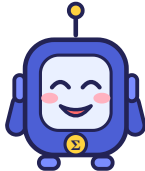
بازده حدود ۸۷-۹۲٪ بود؛ بقیه عمداً رد شدند (بیشتر مسئله‌هایی که بی‌شکل کتاب خودکفا نمی‌شوند). همین مسیر با API پولی هم ممکن است ولی اندازه‌گیری‌شده ~۷۷٪ سنت برای هر سؤال گران‌تر بود بی‌کیفیت بهتر. نویسنده‌های سؤال چند بار خطای حساب خود کتاب را هم گرفتند.

پوشش فعلی: هر جزیره‌ی هر ۸ پایه دست‌کم ۴ سؤال دارد. ۵۹۰ درس و ۵۸ فصل متن معرفی و مثال دارند (`fill_lesson_texts` ، بی‌هزینه، هرگز همان مسئله‌ی آزمون).

۱۱ گیمیفیکیشن و همراه‌ها

سازوکارهای شناخته‌شده‌ی اپ‌های آموزشی موفق (مانند دولینگو): مسیر مرحله‌به‌مرحله، امتیاز، قلب، زنجیره‌ی روزانه، رقابت و شخصیت‌هایی که با دانش‌آموز واکنش نشان می‌دهند. همه در سرور حساب می‌شوند، نه در گوشی، تا قابلِ تقلب نباشند.

همراه‌ها

 تدی ★ رفیقِ همیشگیِ تدریسینو ۵۰۰ امتیاز	 پی Π همراهِ دلگرمی؛ نامِ دستیار ۳۰۰ امتیاز	 دلتا Δ هم‌تیمیِ تمرین‌های سریع ۱۰۰ امتیاز	 سیگما Σ راهنمایِ آزمون و کوییز از ابتدا باز
---	--	--	---

شش حالتِ چهره‌ی هر همراه؛ اپ بر اساسِ نتیجه‌ی یکی را نشان می‌دهد (سرور در پاسخِ هر جواب `mascotMood` می‌فرستد):

 دلخور	 عصبانی	 ناراحت	 هیجان‌زده	 خوشحال	 عادی
--	---	---	--	---	---

بخش	چطور کار می‌کند
کاتالوگ	جدول <code>Character</code> در <code>progress/models.py</code> ؛ ادمین می‌سازد و آستانه‌ی <code>unlock_xp</code> را تعیین می‌کند (مثل پلن‌ها).
باز شدن	<code>UserStats.xp ≥ unlock_xp</code> — جدول مالکیت جدا ندارد؛ ساده و بی‌خطا.
انتخاب	<code>POST /api/characters/select</code> فقط همراه بازشده را می‌پذیرد و در <code>UserStats.selected_character_id</code> می‌نشانند.
تصویر	هیچ فایلِ تصویری نیست: <code>src/mascots/seyar.ts</code> هر همراه و هر حالت را از کد به SVG می‌سازد (تصاویر بالا همان خروجی‌اند) ← حجم کم، واضح روی هر صفحه.

سازوکار	قاعده	کجای کد
امتیاز (XP)	پایان درس یا جزیره: ۱۰ × جواب درست، فقط بار اول. عدد جواب درستی که اپ می‌فرستد به جواب‌های درست ثبت شده محدود می‌شود؛ صدا زدن مستقیم API امتیاز مجانی نمی‌دهد.	<code>content/views.py</code> → <code>NodeQuizCompleteView</code>
ستاره	۹۰٪ به بالا ۳۰٪ به بالا ۲۰٪ بیشتر از صفر ۱	همان
سکه	۵ × ستاره، فقط بار اول	همان
قلب	۵ قلب؛ هر جواب غلط یکی کم؛ هر ۵ ساعت یکی برمی‌گردد و ۲۴ ساعت بعد از اولین قلب ازدست‌رفته همه پر می‌شوند. مشترک فعال قلب از دست نمی‌دهد.	<code>progress/models.py</code> → <code>UserHearts</code>
زنجیره (Streak)	جواب، پایان درس یا جزیره امروز را (به وقت تهران) روز فعال می‌کند. اگر آخرین روز فعال نه امروز باشد نه دیروز، صفر نشان داده می‌شود.	<code>progress/services.py</code>
جدول رتبه	هفتگی (امتیاز ۷ روز اخیر) و همه‌ی زمان‌ها؛ زنده حساب و ۶۰ ثانیه کش می‌شود.	<code>progress/leaderboard.py</code>
پیشرفت روی نقشه	پایان آزمون، جزیره را با هر نمره‌ای تمام می‌کند و جزیره‌ی بعدی (<code>nextNodeId</code>) باز می‌شود؛ نمره فقط ستاره و امتیاز را تعیین می‌کند (تصمیم محصولی: ساده بماند).	<code>content/services.py</code>
اعلان	«جزیره را با ۳ ستاره تمام کردی و ۴۰ امتیاز گرفتی» – با رقم فارسی.	<code>notifications/services.py</code>

محدودیت قلب در سرور فعلاً خاموش است (`HEARTS_ENFORCED=0`): با صفر قلب، اپ جلوی کاربر را می‌گیرد ولی API هنوز جواب را می‌پذیرد، چون فرانت صفحه‌ی «قلب‌ها تمام شد» را ندارد. روشن کردنش یک خط در `.env` است. تنها مزیت واقعی اشتراک در کد فعلاً همین «قلب نامحدود» است – یک تصمیم محصولی.

۱۲ امنیت و احراز هویت

دو دور بازبینی بدبینانه روی کل بک‌اند انجام شد (مرداد و ۳ مهر)، و یک وارسی نهایی روی سایت زنده (۷ مهر) با این سؤال: «یک دانش‌آموز متقلب، یک مهاجم یا یک والدِ سردرگم چطور این را می‌شکند؟» هر یافته با یک تست بسته شد. این بخش صادقانه می‌گوید چه بسته است و چه هنوز باز.

۱۲-۱ هویت و توکن



کاربرهای اپ (`accounts.User`) عمداً از کاربرهای پنل ادمین (کاربرِ استاندارد جنگو، با رمز) جدا هستند؛ دانش‌آموز هیچ راهی به پنل ندارد.

۲-۱۲ آنچه رعایت شده

حوزه	چه کرده‌ایم	کجا
دسترسی به داده‌ی دیگران	شناسه‌ی کاربر همیشه از توکن خوانده می‌شود، هرگز از ورودی؛ دیدنِ پیشرفتِ فرزند فقط با اتصالِ تأییدشده.	همه‌ی <code>views.py</code> ها
سقفِ نرخ	روی همه‌ی نقاطِ حساس (ورود، دستیار، خرید، کدِ تخفیف، جواب...); مهمان بر اساسِ IP واقعی (<code>NUM_PROXIES=2</code> : آروان + nginx).	<code>config/settings.py</code>
پیامک و هزینه‌اش	شماره به <code>09XXXXXXXX</code> نرمال می‌شود؛ فاصله بین دو درخواست، ۱۰ در روز برای هر شماره، ۳۰۰ در روز برای کلِ سامانه (ضدِ SMS pumping) – شمارش در دیتابیس، نه حافظه‌ی موقت.	<code>accounts/phone.py</code> <code>tests_otp_limits.py</code>
پرداخت	فقط تأییدِ سرور-به-سرور؛ برگشتِ جعلی اثری ندارد؛ webhook عمومی ناامن حذف شد؛ هر درگاهی بی‌کلید «بسته» می‌ماند (fail-closed)، نه موفقیتِ ساختگی.	<code>payment/</code>
تقلبِ امتیاز	امتیاز فقط برای جوابِ ثبت‌شده؛ جایزه فقط بارِ اول.	<code>content/views.py</code>
درخواستِ تکراری و هم‌زمانی	Idempotency-Key در شش نقطه؛ <code>transaction.atomic</code> ؛ قیدِ یکتا در دیتابیس؛ SQLite در حالتِ <code>IMMEDIATE</code> تا قفل‌ها واقعاً قفل کنند.	<code>config/settings.py</code>
خطا و لاگ	هر خطای JSON یکدست بی‌جزئیاتِ داخلی؛ لاگ بدونِ بدنه، هدر، توکن یا رمز.	<code>config/exceptions.py</code> <code>middleware.py</code>
محتوای مدل	HTML یا SVG تولیدِ مدل هرگز وارد صفحه نمی‌شود؛ فقطِ JSON واریسی‌شده با فهرستِ سفید.	<code>rag/services/diagram.py</code>
مرورگر	CORS فقط برای دامنه‌ی فرانت؛ پاسخِ API با <code>no-store</code> در کشِ آروان نمی‌ماند؛ نامِ نمایشی از شماره و لینک پاک می‌شود؛ سقفِ حدسِ رمزِ ادمین برای هر IP.	<code>config/middleware.py</code>
کلیدها	همه در <code>.env</code> ، سرور؛ هیچ کلیدی در git یا در پاسخِ API نیست؛ <code>/api/health</code> فقط «بله/خیر» می‌گوید.	<code>.env.example</code>

۳-۱۲ آنچه عمداً باز مانده (پیش از کاربرِ واقعی باید بسته شود)

این موارد در مرحله‌ی نمایشی (دمو) برای کارِ تیمِ عمداً باز گذاشته شده‌اند؛ فهرستِ کامل با خطر و راهِ بستنِ هرکدام در `docs/security-before-production.md` است (۱۸ مورد).

#	مورد	خطر	راه بستن
۱	کلید امضای توکن روی سرور همان کلید پیش‌فرضِ داخلِ مخزن است؛ هر کس مخزن را ببیند می‌تواند توکن جعل کند.	بحرانی	حذف از <code>.env</code> تا تنظیمات خودش کلید تصادفی بسازد (کد آماده است)؛ همه یک بار دوباره وارد می‌شوند.
۲	۷ شماره‌ی دمو با کد ثابت روی سرور واقعی	بالا	پاک کردن <code>TEST_LOGIN_PHONES</code>
۳	داده و حساب‌های دمو و آزمایشی در دیتابیس واقعی	بالا	پاک‌سازی پس از بکاپ
۴	دسترسی سرور: حساب مشترک با رمز و <code>sudo</code> کامل	بالا	کلید SSH و حساب محدود – کار ادمین سرور
۵	<code>/api/health</code> ، Swagger و جدول رتبه بی‌ورود عمومی‌اند؛ پنل ادمین روی دامنه‌ی عمومی	متوسط	محدود کردن با ورود یا IP
۶	سرور بی‌آروان هم روی پورت ۸۰ جواب می‌دهد (امکان جعل IP برای دور زدن سقف)	متوسط	بستن پورت جز برای آروان – ادمین سرور
۷	کش مشترک (Redis) تنظیم نشده؛ سقف‌های نرخ برای هر worker جداست	متوسط	<code>CACHE_URL</code> روی Redis موجود سرور
۸	کد پیامکی و <code>refresh token</code> بی‌هش در دیتابیس؛ توکن در <code>localStorage</code> مرورگر	کم	هش کردن؛ کوکی <code>httpOnly</code>
۹	آزمون جزیره قفل بودن جزیره را نمی‌سنجد (می‌شود جزیره‌ی جلوتر را با API تمام کرد)	کم	بررسی جزیره‌ی قبلی در <code>NodeQuizCompleteView</code>

جمع‌بندی وضعیت امنیت

لایه‌ی برنامه‌بازبینی و تست شده است: دسترسی، سقف نرخ، پرداخت، تکرار درخواست و تقلب امتیاز بسته‌اند. چیزهایی که باز مانده عمدی و مستند است – مهم‌ترین آن‌ها کلید امضای سرور است که کد رفعش آماده است و اعمالش به زمانی موکول شده که خروجی همه‌ی کاربرها از حساب مشکلی ایجاد نکند. بخشی از موارد نیز به زیرساخت مربوط است و بر عهده‌ی ادمین سرور است.

۱۳ استقرار (Deploy) و نگهداری

هدف: هیچ مرحله‌ی دستی یادآوردنی در انتشار نباشد. یک دستور، و اسکریپت خودش ثابت می‌کند که کار شد.

۱۳-۱ انتشار بک‌اند

```
ssh -t administrator@<server> 'cd ~/mathapp-v2 && ./deploy.sh; sudo systemctl restart zareian-mathapp'
```

health · ۸ poll تا سالم	restart · ۷ اثبات + ری‌استارت	· ۶ collectstatic	· ۵ seed_content محتوا	migrate · ۴	pip · ۳ install	reset · ۲ تغییر دستی ← patch در · deploy- backups/	fetch · ۱ از Gitea
----------------------------	-------------------------------------	----------------------	------------------------------	-------------	--------------------	--	-----------------------

- **سرور را تعمیر می‌کند، جلوی‌ش نمی‌ایستد:** فایلی که کسی دستی روی سرور عوض کرده ذخیره و بعد دور ریخته می‌شود؛ سرور دقیقاً روی چیزی می‌رود که در Gitea است.
- **ری‌استارت را ثابت می‌کند:** زمان شروع سرویس را قبل و بعد مقایسه می‌کند. یک بار سرور دو روز کِـ قدیمی سرو کرد درحالی‌که همه‌ی فایل‌ها تازه بودند — از آن به بعد «فایل روی دیسک» را «منتشرشده» حساب نمی‌کنیم.
- ری‌استارت جداست چون `sudo` داخل اسکریپت نمی‌تواند رمز بپرسد.

۲-۱۳ انتشار فرانت

```
ssh -t administrator@<server> 'cd ~/mathapp_frontend_new && ./deploy.sh'
```

fetch ← `npm ci` ← بررسی نوع‌ها ← build در `dist.next` ← جابه‌جایی بی‌وقفه؛ build قبلی در `dist.prev` می‌ماند و برگشت یک دستور است. نشانی API در `dist/config.js` است و بی‌build عوض می‌شود.

۳-۱۳ پیکربندی (env . سرور)

متغیر	اگر نباشد
FRONTEND_URL · PUBLIC_BASE_URL	دامنه‌های فعلی. همه‌ی تنظیمات دامنه (CORS، callback، زرین‌پال، ...) از همین دو ساخته می‌شوند؛ جابه‌جایی دامنه یعنی عوض کردن همین دو.
DJANGO_SECRET_KEY	یک کلید تصادفی ساخته و در <code>django_secret_key</code> نگه داشته می‌شود (حالت مطلوب).
QDRANT_API_KEY · OPENAI_API_KEY	دستیار جواب ثابت نمایش می‌دهد.
KAVENEGAR_API_KEY · KAVENEGAR_OTP_TEMPLATE	درخواست کد ۵۰۳ صریح می‌گیرد. الگو باید <code>SMSVerify</code> بماند.
BAZAAR_* · ZARINPAL_MERCHANT_ID	پرداخت بسته می‌ماند (۵۰۳).
CACHE_URL	کش جدا برای هر worker؛ <code>/api/health</code> هشدار می‌دهد.
HEARTS_ENFORCED · TEST_LOGIN_PHONES	خاموش · هیچ حساب آزمایشی.
DJANGO_DEBUG	خاموش — و هرگز روی سرور روشن نشود (کد ورود همگانی <code>0000</code> را فعال می‌کند).

مقدارهای `env` با پیش‌فرض‌های کد ادغام می‌شوند؛ یک `env` ناقص سرویس را نمی‌خواهاند. شرح کامل در `env.example` و `docs/operations.md`.

بکاپ

دیتابیس در git نیست و تنها نسخه‌ی داده‌ی کاربرهاست. بکاپ با `sqlite3.Connection.backup()` پایتون (نه `cp`) پیش از هر انتشارِ مهم، در `deploy-backups/`. **هنوز خودکار نیست** – ارزشمندترین کارِ بعدی.

GET /api/health

بی‌ورود و بی‌SSH می‌گوید: دیتابیس سالم است؟ migration جامانده هست؟ کش کار می‌کند؟ چند پایه/درس/سؤال؟ کدام سرویس بیرونی تنظیم است؟ اعتبارِ دستیار تمام شده (`aiProvider`)؟ هشدارهای امنیتی و ترافیکِ اخیر. ۲۰۰ = سالم، ۵۰۳ = خرابیِ واقعی.

سرور مالِ ما نیست؛ ما یکی از مستأجرهاییم

- مالِ ما فقط `/home/administrator/mathapp-v2` و سرویسِ `zareian-mathapp` است. `unicorn` خالی سرویسِ تیمِ دیگری است.
- هرگز `git clean` یا «پاک کن و از نو clone کن»: `db.sqlite3`، `.env` و کلیدِ امضا در همان پوشه‌اند و در git نیستند.
- پیش از هر کارِ پرحجم `df -h /`: دیسک حدود ۹۵٪ پر است و پر شدنش Gitea و ایمیلِ شرکت را هم می‌خواباند.
- `.env` فقط از راهِ جنگو خوانده شود؛ مقدارش هرگز در چت یا گزارش نیاید.

۱۴ مقیاس‌پذیری و پیوستنِ توسعه‌دهنده‌ی تازه

دو معنا دارد: کد چقدر برای تیمِ بزرگ‌تر آماده است، و سیستم چقدر برای کاربرِ بیشتر.

۱۴-۱ آیا دو سه نفرِ دیگر می‌توانند به بک‌اند اضافه شوند؟ بله – به این دلایل

ویژگی	چرا کارِ هم‌زمان را ممکن می‌کند
۱۵ اپ با مرز روشن	هر نفر صاحبِ چند اپ می‌شود (مثلاً یکی پرداخت و اشتراک، یکی محتوا و گیمیفیکیشن، یکی دستیار)؛ تغییرِ هر کس کمتر به فایلِ دیگری می‌خورد.
بی‌FK بینِ اپ‌ها	تغییرِ جدول‌های یک اپ migration اپِ دیگر را نمی‌شکند؛ هر اپ بعداً قابلِ جدا شدن به سرویسِ مستقل است.
قراردادِ OpenAPI	فرانت و بک‌اند بی‌جلسه روی یک شکلِ داده توافق دارند؛ <code>tests_frontend_contract.py</code> تغییرِ ناخواسته را می‌گیرد.
۶۸۵ تستِ خودکار	تازه‌وارد بی‌ترس تغییر می‌دهد: اگر چیزی را شکست، تست قبل از سرور خبر می‌دهد.
قواعدِ یک‌جا	نرخ‌ها، دامنه‌ها و تنظیمات فقط در <code>config/settings.py</code> ؛ پیامِ خطا، لاگ و هویت یک مسیرِ مشترک دارند.
مستندات در مخزن	<code>docs/</code> + همین سند + Swagger + توضیحِ «چرا» کنارِ کدِ حساس.

۲-۱۴ دستورالعمل: افزودن یک اندپوینت تازه

1. اپ درست را انتخاب کنید (یا اپ تازه با `python manage.py startapp` و افزودن به `INSTALLED_APPS` و `config/urls.py`).
2. مدل در `models.py` – ارجاع به اپ دیگر فقط با `user_id` یا شناسه‌ی عددی – و `python manage.py makemigrations`.
3. ورودی/خروجی در `serializers.py`، با نام `camelCase`.
4. ویو در `views.py` با `permission_classes`؛ اگر حساس است `throttle_scope` و ردیف متناظر در `DEFAULT_THROTTLE_RATES`؛ اگر اثر ماندگار دارد، `Idempotency-Key` و `transaction.atomic`؛ تماسی شبکه‌ای بیرون از تراکنش.
5. مسیر در `urls.py` اپ و `@extend_schema` تا در Swagger بیاید.
6. تست در `tests.py` – از جمله «کاربر دیگری نمی‌تواند».
7. `python manage.py test` ← `push` ← `deploy.sh` ← `/api/health`.
8. فرانت: نوع در `src/api/types.ts` از روی پاسخ واقعی، تابع در `src/api/index.ts`.

۳-۱۴ مقیاس اجرا: کجا گلوگاه است و بعد چه

وضع فعلی: ۲۴ درخواست هم‌زمان؛ درخواست عادی چند ده میلی‌ثانیه، جواب دستیار ده‌ها ثانیه. توکن stateless است، پس هر تعداد نسخه‌ی دیگر از بک‌اند پشت `nginx` بی‌تغییر کد کار می‌کند – به شرط دیتابیس و کش مشترک.

#	گلوگاه	راه‌حل ساده‌ی بعدی	نیاز
۱	SQLite فقط یک نویسنده دارد	Postgres (همین حالا روی سرور هست)؛ تغییر تنظیمات + انتقال داده؛ حالت <code>IMMEDIATE</code> برداشته می‌شود و قفل ردیف واقعی می‌شود.	ادمین سرور
۲	کش و سقف نرخ جدا برای هر worker	<code>CACHE_URL</code> روی Redis موجود (db1 آزاد است).	یک خط در <code>.env</code>
۳	جواب دستیار نخ را نگه می‌دارد	صف کار (مثلاً Celery، که RabbitMQ و Redis اش روی سرور هست) یا ویوی <code>async</code> برای <code>/api/ai/ask</code> .	کد
۴	یک سرور برای همه	چند نسخه‌ی <code>gunicorn</code> پشت <code>nginx</code> ؛ فرانت از CDN آروان.	زیرساخت
۵	هزینه‌ی دستیار با کاربر زیاد	کش، سقف روزانه (هست)، و سنجش مدل ارزان‌تر روی همان ابزار <code>rag_model_benchmark</code> .	سنجش

بهینه‌سازی‌های انجام‌شده: رفع $N+1$ در صفحه‌ی اصلی (از $N+1$ کوئری به ۳ کوئری ثابت)، کش نقشه، کش جدول رتبه (۶۰ ثانیه)، کش جواب دستیار، ادغام چند درخواست در یکی (`/api/home`، `/api/user/summary`، `/api/nodes/{id}/quiz`)، و حجم بار اول فرانت از ۵۰۵ به ۸۵ کیلوبایت.

۱۵ تست و کیفیت

چه	چطور
تستِ خودکار بک‌اند	۶۸۵ تست، همه سبز (<code>python manage.py test</code>). روی فایل دیتابیسِ واقعی (<code>-test</code>). <code>db.sqlite3</code> اجرا می‌شوند تا رفتارِ قفل‌ها مثل سرور باشد. هر باگِ پیداشده در بازبینی‌ها یک تستِ اختصاصی دارد (<code>tests_audit_fixes.py</code> ، <code>tests_otp_limits.py</code> ، ...).
قرارداد با فرانت	<code>content/tests_frontend_contract.py</code> و <code>build</code> با <code>TypeScript strict</code> در فرانت.
تستِ روی سایتِ زنده	هر صفحه با حساب‌های نمایشی روی سرورِ واقعی؛ مرورگرِ خودکار (<code>puppeteer</code>) برای واریِ صفحه‌ها.
کیفیتِ دستیار	<code>rag_model_benchmark</code> (مسئله‌های واقعیِ کتاب از مسیرِ واقعی، با هزینه‌ی واقعی)، <code>ai_feedback_report</code> (👍/👎 به تفکیکِ نسخه‌ی پرامپت)، واریِ چشمیِ نمونه پیش از اعتماد به هر اجرای انبوه.
بی‌هزینه آزمودن	<code>tools/fake_openai.py</code> جای <code>GapGPT</code> می‌نشیند.
پایش	لاگِ JSON هر درخواست + <code>/api/health</code> ؛ محلی با <code>Grafana/Loki</code> (<code>docs/docker.md</code>).

فرهنگِ کد: توضیح‌ها کنارِ کد «چرا» را می‌گویند، نه «درست است». قاعده‌ی مخزن: کامنت ادعاست، نه مدرک — هر ادعا با تست یا اجرا سنجیده می‌شود. چند باگِ واقعی دقیقاً زیرِ کامنتی پیدا شد که کد را «امن» می‌خواند.

۱۶ وضعیتِ فعلی، محدودیت‌ها و تصمیم‌های باز

وضعیت در تاریخِ نگارشِ سند (۷ مهر ۱۴۰۵، روی سرورِ زنده)

بخش	وضعیت	توضیح
اپ و API	زنده	هر دو روی آخرین نسخه؛ همه‌ی صفحه‌ها با حساب‌های نمایشی واری شده
محتوا	کامل	۸ پایه؛ هیچ جزیره‌ای زیرِ ۴ سؤال نیست
پیامکِ ورود	کار می‌کند	کدِ واقعی به گوشی رسید (الگوی <code>SMSVerify</code>)؛ هر پیامک هزینه دارد
دستیار + شکل	کار می‌کند	کُند (~۳۰ ثانیه، از سمتِ <code>GapGPT</code>)؛ ~یک شکل از پنج ایراد دارد
زرین‌پال	نیمه	درخواستِ پرداخت پذیرفته می‌شود؛ پرداختِ واقعی تا انتها آزموده نشده
کافه‌بازار	منتظر	کد آماده؛ منتظرِ تأییدِ حسابِ شرکت و اپ اندرویدی
پیامِ گروهی	پایه	ذخیره و هدف‌گیری هست؛ زمان‌بندیِ خودکار نه
بکاپِ خودکار	ندارد	دستی پیش از هر انتشار

تصمیم‌هایی که با مدیریت است، نه با کد

موضوع	چرا تصمیم لازم است
عوض کردنِ کلیدِ امضا	کد آماده؛ همه یک بار از حساب بیرون می‌افتند – زمانش را باید تعیین کرد.
پاک‌سازیِ داده‌ی دمو	پیش از اولین کاربرِ واقعی: ۷ شماره‌ی آزمایشی، حساب‌های دمو و مهمان‌های تستی.
مزیتِ اشتراک	در حال حاضر فقط «قلبِ نامحدود»؛ دستیار و درس‌ها برای همه آزادند. چه چیزی پولی باشد؟
Postgres و Redis	هماهنگی با ادمینِ سرور؛ هر دو روی همان ماشین موجودند.
پورت ۸۰ و دسترسیِ SSH	کارِ ادمینِ سرور.
پیشرفت با هر نمره	در حال حاضر جزیره با هر نمره‌ای تمام می‌شود؛ اگر حدِ قبولی لازم است، یک تصمیمِ آموزشی است.

کارِ بعدی، به ترتیبِ ارزش

1. بستنِ موارد ۱ تا ۳ امنیت و پاک‌سازیِ داده‌ی دمو.
2. بکاپِ خودکارِ دیتابیس، بیرون از همان سرور.
3. یک پرداختِ واقعی کامل با زرین‌پال.
4. `CACHE_URL` (Redis) و بعد Postgres.
5. صفِ کار برای دستیار؛ سنجشِ شکل روی نمونه‌ی بزرگ‌تر.
6. روشن کردنِ `HEARTS_ENFORCED` همراه با صفحه‌ی «قلب‌ها تمام شد» در فرانت.

واژه	معنی در این پروژه
API / اندپوینت	نشانی‌ای که اپ برای گرفتن یا فرستادن داده صدا می‌زند، مثل <code>GET /api/home</code> .
اپ جنگو	یک پوشه‌ی مستقل با مدل، ویو و تست خودش؛ یک مسئولیت.
مدل / migration	تعریف یک جدول در پایتون / اسکرپتی که تغییر جدول را روی دیتابیس اعمال می‌کند.
ویو / serializer	کدی که درخواست را جواب می‌دهد / کدی که شکلی ورودی و خروجی را تعریف و واری می‌کند.
middleware	لایه‌ای که همه‌ی درخواست‌ها پیش و پس از ویو از آن رد می‌شوند.
توکن	«کارت شناسایی» امضا شده که اپ با هر درخواست می‌فرستد.
throttle	سقف تعداد درخواست در زمان.
Idempotency-Key	کلیدی که باعث می‌شود تکرار یک درخواست دو بار اثر نکند.
تراکنش اتمیک	چند نوشتن که یا همه انجام می‌شوند یا هیچ‌کدام.
RAG	اول پیدا کردن متن مرتبط از کتاب، بعد نوشتن جواب بر پایه‌ی آن.
embedding / بردار	تبدیل متن به عددی که معنا را نشان می‌دهند تا متن‌های هم‌معنا کنار هم بیفتند.
chunk / collection	یک تکه‌ی کتاب (یک تیپ‌مسئله) / همه‌ی تکه‌های یک کتاب در Qdrant.
SSE (استریم)	فرستادن جواب تکه‌تکه، تا کاربر پیش از تمام شدن چیزی ببیند.
seed	پر کردن دیتابیس از فایل‌های محتوای داخل مخزن.
fail-closed	اگر تنظیمات نبود، بسته بماند (خطای صریح)، نه اینکه وانمود کند کار کرد.

ب · دستورهای پرکاربرد

```
# توسعه‌ی محلی
python3 -m venv venv && source venv/bin/activate && pip install -r requirements.txt
cp .env.example .env # DJANGO_DEBUG=1 برای محلی
python manage.py migrate && python manage.py seed_content && python manage.py runserver
python manage.py test # کل تست‌ها
python manage.py spectacular --file schema.yaml # قرارداد API

# محتوا و دستیار
python manage.py seed_content --export --only questions
python manage.py rag_catalog # فقط خواندن (collection های Qdrant)
python manage.py rag_ingest_book --grade نهم --plan
python manage.py rag_verify_solutions
python manage.py generate_questions_from_chunks --subject ریاضی --grade نهم --export /tmp/b
python manage.py ai_feedback_report
python manage.py rebuild_leaderboard
python manage.py payment_reconcile --days 3

# فرانت
npm install && npm run dev # http://localhost:5173
npm run build # type-check + build در dist/
```

ج · نقشه‌ی مستندات موجود

سند	برای
README.md (هر دو مخزن)	در ورودی و راه‌اندازی
docs/architecture.md	تصمیم‌های معماری و دلیلشان، با جزئیات
docs/operations.md	انتشار، متغیرها، بکاپ، جاب‌جایی دامنه، ورود تیم
docs/security-before-production.md	۱۸ مورد امنیتی باز با خطر و راه بستن
docs/content-authoring.md	ویرایش محتوا و ساخت انبوه سؤال
docs/extraction-playbook.md	خواندن کتاب با مدل بینایی: هزینه و درس‌ها
data/textbooks/README.md	افزودن کتاب تازه
/api/docs/	قرارداد زنده و قابل امتحان API

تدریسینو · مستند فنی · علیرضا زارعیان · مهر ۱۴۰۵
عددها از کد، اسکیمای API، دیتابیس و سنجش‌های ثبت‌شده در مخزن استخراج شده‌اند.