مستندِ فنیِ تدریسینو

معماری، ساختارِ کد، مدلِ داده، APIها، دستیارِ هوشِ مصنوعی و داده‌ی Qdrant، گیمیفیکیشن، امنیت، استقرار و مسیرِ مقیاس‌پذیری — برای خواننده‌ای که با پروژه آشنایی قبلی ندارد و می‌خواهد آن را توسعه دهد یا نگهداری کند.

تاریخ	۱۰ مهر ۱۴۰۵ (۲ اکتبر ۲۰۲۶)
نویسنده	علیرضا زارعیان — بک‌اند و فرانت‌اند
مخاطب	توسعه‌دهنده‌ی تازه‌وارد، مدیرِ فنی، و هر کسی که باید درباره‌ی ادامه‌ی پروژه تصمیم بگیرد
کدِ مبنا	بک‌اند `8a4753b` (مخزنِ `Tadrisino/MathApp`) · فرانت `33f5f8d` (مخزنِ `Tadrisino/MathApp_frontend`)
نشانی‌ها	اپ: `math.tadrisino2.ir` · API: `mathapi.tadrisino2.ir` · مستنداتِ زنده‌ی API: `/api/docs/`

این فایل نسخه‌ی Markdown «مستند-فنی-تدریسینو» است. نسخه‌ی PDF با همان محتوا و صفحه‌آراییِ کامل کنارِ همین فایل است: `technical-handbook.pdf` .

۱. معرفیِ پروژه در یک نگاه

تدریسینو یک اپِ آموزشِ ریاضی به سبکِ دولینگو است: دانش‌آموز روی نقشه‌ای از «جزیره»ها جلو می‌رود، هر جزیره یک آزمونِ کوتاه از مسئله‌های خودِ کتابِ درسی است، و هر جا گیر کرد از دستیارِ هوشِ مصنوعی («پی») می‌پرسد که جوابش را از روی همان کتاب می‌سازد.

شاخص	مقدار
سوم، ششم، هفتم، هشتم تا دوازدهم	**۸ پایه**
سؤالِ چهارگزینه‌ای از مسئله‌های کتاب	**۲,۵۴۵**
اندپوینتِ API، مستند در Swagger	**۵۹**
تستِ خودکار، همه سبز	**۷۷۹**
اپِ جنگو، هرکدام یک مسئولیت	**۱۴**
تکه‌ی کتاب در Qdrant (۱۱ collection)	**۳,۵۱۷**
همراهِ گیمیفیکیشن، هرکدام ۶ حالتِ چهره	**۴**
بک‌اند و فرانت؛ یک قراردادِ API	**۲ مخزن**

کاربرها و کاری که می‌کنند

کاربر	چه می‌کند	کجای کد
دانش‌آموزِ مهمان	بدونِ 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`
دانش‌آموز	از هر سنی (بزرگسالی که فقط یک مبحث را می‌خواهد هم پایه‌ی همان مبحث را انتخاب می‌کند). با شماره‌ی موبایل و کدِ پیامکی وارد می‌شود؛ جزیره‌ها را بازی می‌کند، از دستیار می‌پرسد، امتیاز و سکه و زنجیره‌ی روزانه جمع می‌کند، همراه انتخاب می‌کند، اشتراک می‌خرد، تیکتِ پشتیبانی می‌فرستد.	`progress/` `content/` `ai_tutor/`
والد	حسابِ جدا ندارد: «گزارشِ والدین» را در حسابِ خودِ دانش‌آموز می‌بیند — زمانِ تمرین، سوال‌ها و مقایسه با هفته‌ی پیش، با دو نمودار.	`progress/report.py`
ادمین (کارمندِ شرکت)	از پنلِ `/admin/` سؤال، درس، پلن، همراه و تیکت را بدونِ برنامه‌نویس ویرایش می‌کند؛ سؤال‌ها را فله‌ای وارد و خارج می‌کند.	`*/admin.py`

تاریخچه‌ی کوتاه

زمان	اتفاق
مرداد ۱۴۰۵	طراحیِ معماری و مدل‌های داده؛ هویتِ کاربر؛ جداسازیِ اپ‌ها. بازبینیِ امنیتیِ اول و بستنِ حفره‌ها، محدودیتِ نرخ در کلِ سیستم.

زمان	اتفاق
شهریور	دستیار از پاسخِ ساختگی به RAG واقعی روی کتاب رسید؛ اتصالِ زرین‌پال؛ گیمیفیکیشن؛ خواندنِ پنج کتابِ ریاضی و ساختِ هزاران سؤالِ تأییدشده؛ حسابانِ ۱۱ و ۱۲.
۲ تا ۷ مهر	فرانت به ما سپرده شد و از نو ساخته شد؛ هر دو روی دامنه‌ی تازه زنده شدند؛ پیامکِ واقعی؛ شکل در جوابِ دستیار؛ همراهِ چهارم؛ بازبینیِ نهایی روی سایتِ زنده.

مدلِ سناریو (Use Case)

چه کسانی با سامانه کار می‌کنند و هرکدام چه کاری انجام می‌دهند. خطِ‌چینِ بنفش یعنی آن کار به یک سرویسِ 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 بخشِ ۶ آمده است.

شکلِ ۱-۱ — مدلِ سناریو: سه نقشِ کاربری (والد با حسابِ خودِ دانش‌آموز)، یازده کارِ اصلی، و سه سرویسِ بیرونی.

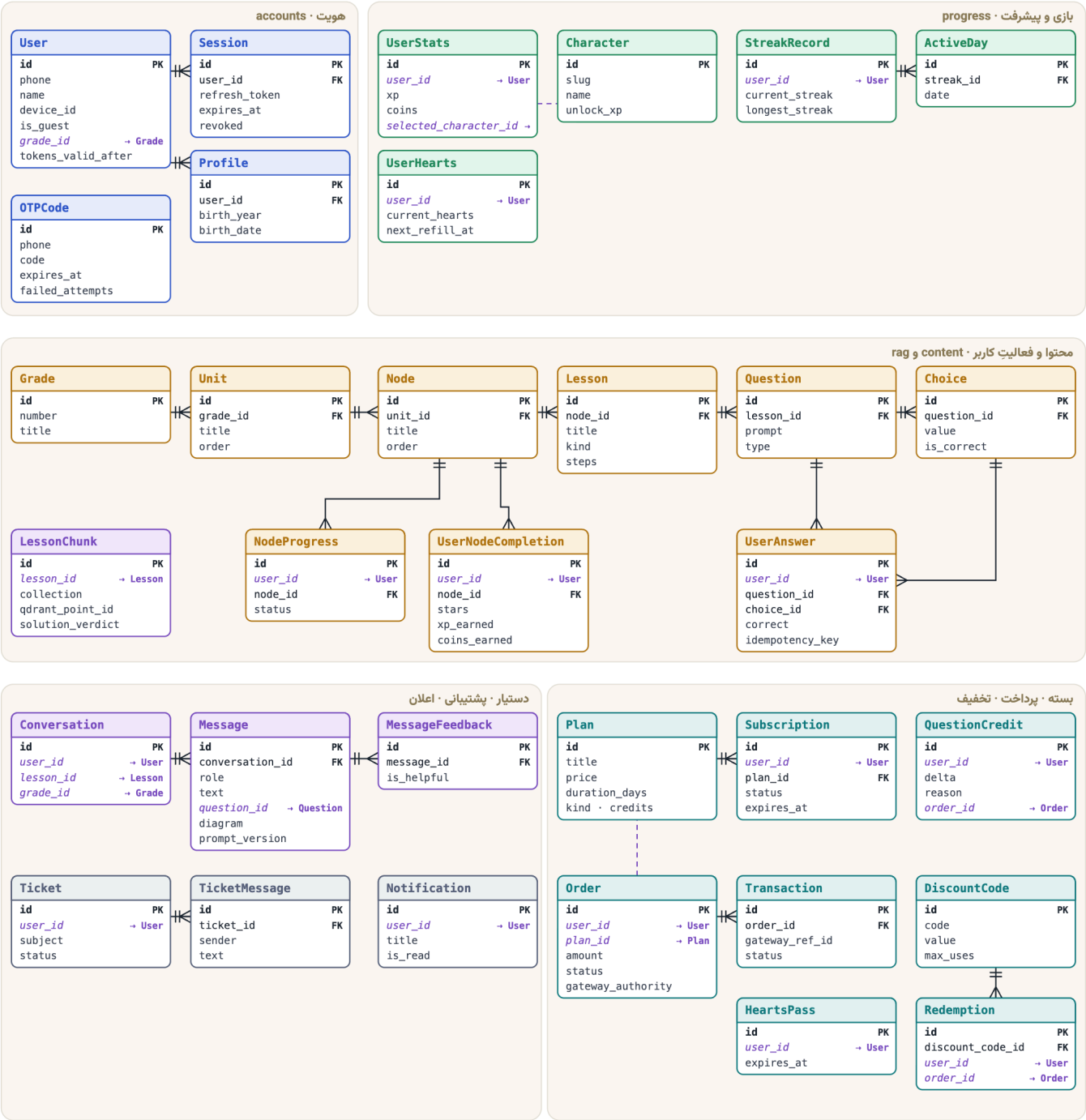
واحدهای محتوا (این واژه‌ها در کلِ سند تکرار می‌شوند)

پایه Grade ۸ ← فصل Unit ۵۸ ← جزیره Node ۲۶۳ ← درس Lesson ۲,۶۱۷ ← سؤال Question ۲,۵۴۵ ← گزینه Choice

«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.

مدلِ رابطه‌ای (ER)

جدول‌های اصلیِ دیتابیس و ارتباطِ میانِ آن‌ها، دسته‌بندی‌شده بر اساسِ اپ. **خطِ ممتد** یک کلیدِ خارجیِ واقعی (FK) در داخلِ یک اپ است: دو خطِ موازی سمتِ «دقیقاً یک» و پای کلاغی سمتِ «صفر یا چند» (مثلاً هر پایه چند فصل دارد). **فیلدِ بنفش با ⇢** ارجاعِ سبک به جدولِ اپِ دیگر است: فقط شناسه ذخیره می‌شود و قیدِ دیتابیسی ندارد تا هر اپ مستقل بماند (بخشِ ۳). **خطِ‌چینِ بنفش** دو نمونه از همین ارجاع‌های سبک را نشان می‌دهد. فیلدهای کم‌اهمیت حذف شده‌اند؛ فهرستِ کامل در بخشِ ۷ و در `<app>/models.py`.



شکلِ ۱-۲ — مدلِ رابطه‌ای؛ نام‌ها و ارتباط‌ها مستقیم از مدل‌های جنگو استخراج شده‌اند.

۲. انتخاب‌های فناوری و چرایی‌شان

هر انتخاب یک دلیل و یک هزینه دارد. ستونِ آخر هزینه را صادقانه می‌گوید، چون همان چیزی است که نفرِ بعدی باید بداند.

لایه	انتخاب	چرا	چرا نه جایگزین	هزینه / ریسک
بک‌اند	**Django 5.2** + **Django REST Framework**	ORM و migration و **پنلِ ادمینِ آماده** در یک بسته؛ پنلِ ادمین یعنی محتوا بدونِ برنامه‌نویس ویرایش می‌شود. DRF احرازِ هویت، مجوز، محدودیتِ نرخ و سریال‌سازی را استاندارد می‌دهد. پایتون همان زبانِ کتابخانه‌های هوشِ مصنوعی است (`openai` ، `qdrant-client` ، `pymupdf`).	FastAPI سریع‌تر و async است ولی ادمین، ORM و migration را باید جدا سرهم کرد. Node.js زبانِ بک‌اند را از زبانِ لایه‌ی AI جدا می‌کرد.	جنگو پیش‌فرض همگام (sync) است؛ جوابِ طولانیِ دستیار یک نخ را نگه می‌دارد. با ۲۴ نخِ هم‌زمان حل شد (بخشِ ۱۴).
قراردادِ API	**OpenAPI** با drf-spectacular	اسکیما از خودِ کد ساخته می‌شود، پس هیچ‌وقت از کد عقب نمی‌ماند. Swagger در `/api/docs/` قراردادِ مشترکِ دو مخزن است.	سندِ دستی یا «در ذهنِ کسی» — که دقیقاً باعثِ ناهمخوانیِ اولیه‌ی فرانت و بک‌اند شد.	—
فرانت	**React 18** + **TypeScript** (strict) + **Vite** + **Tailwind**	فرانتِ 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 ناهمخوانیِ شکلِ داده با API را در زمانِ build می‌گیرد. Vite build سریع و Tailwind رنگ و فاصله‌ی یکدست. وبِ موبایل‌محور یعنی بدونِ نصب و بدونِ فروشگاه روی هر گوشی باز می‌شود.	React Native / Flutter نیازِ انتشار در فروشگاه و build جدا داشت؛ نسخه‌ی اندروید (کافه‌بازار) بعداً می‌تواند اضافه شود.	بدونِ کتابخانه‌ی router و state؛ کمتر وابستگی، ولی الگوی داخلی را باید خواند (بخشِ ۴).
دیتابیس	**SQLite** (فعلاً)	روی سرورِ مشترکِ شرکت صفر نگهداری دارد: یک فایل، بکاپ با یک دستور. برای حجمِ فعلی کافی است. کد از ORM استفاده می‌کند و به SQLite گره نخورده.	Postgres روی همان سرور موجود است؛ مهاجرت به هماهنگی با ادمینِ سرور نیاز دارد و به مرحله‌ی بعد موکول شده است.	**فقط یک نویسنده در لحظه.** اولین گلوگاهِ مقیاس است (بخشِ ۱۴).
جست‌وجوی کتاب	**Qdrant** (بانکِ برداری)	تیمِ دیگرِ شرکت کتاب‌های سوم/ششم/هفتم را از قبل در یکِ Qdrant مشترک گذاشته بود؛ همان را ادامه دادیم تا داده دوباره ساخته نشود. فیلتر روی فیلدِ «فصل» دارد.	pgvector به Postgres نیاز داشت؛ FAISS سرور ندارد و بینِ دو تیم اشتراک‌پذیر نیست.	سرویسِ مشترک با تیمِ دیگر؛ پاک‌کردن باید با احتیاط باشد (بخشِ ۹).

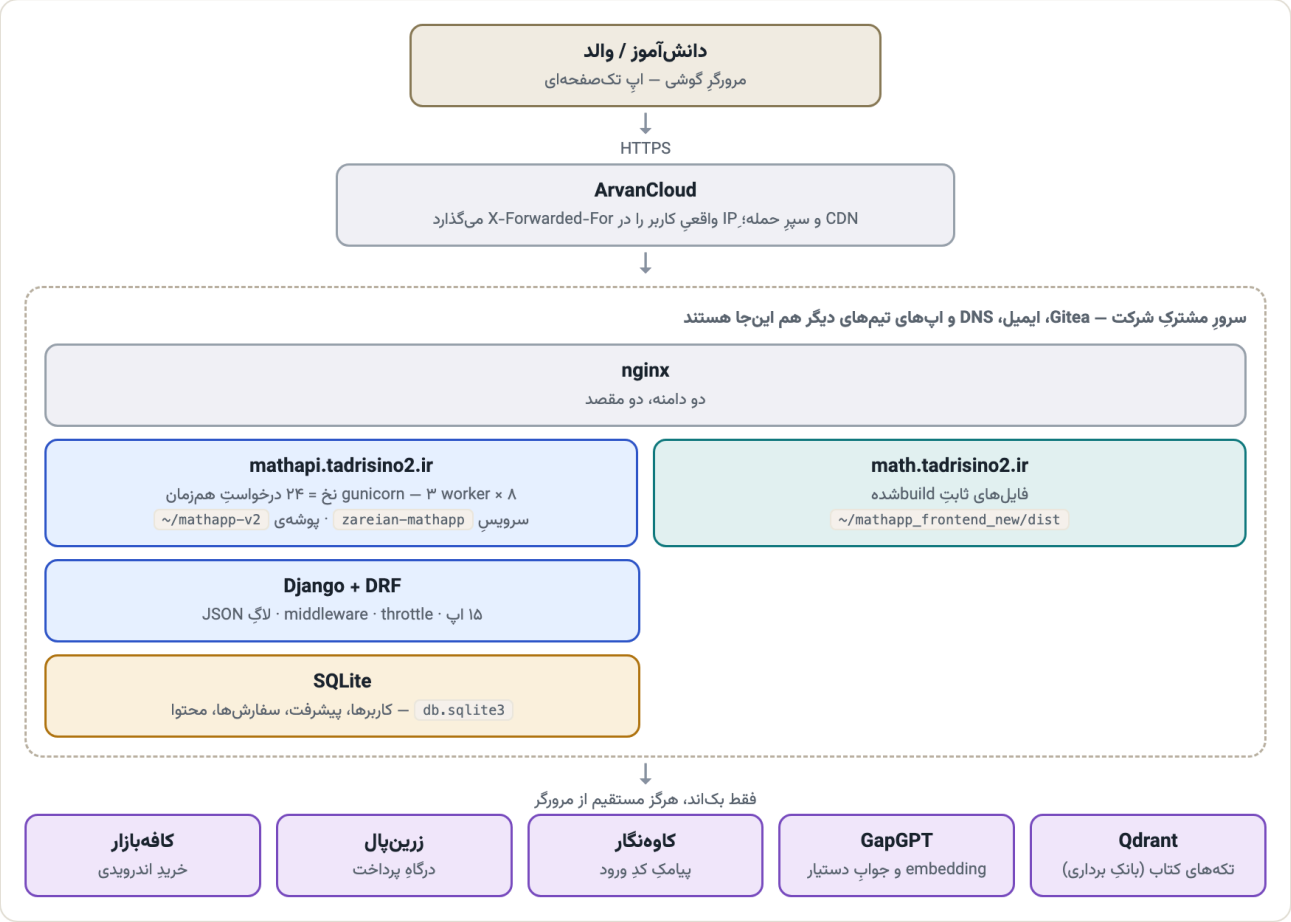
لایه	انتخاب	چرا	چرا نه جایگزین	هزینه / ریسک
مدلِ زبانی	**GapGPT** ← `gpt-4.1-mini`	GapGPT یک واسطِ سازگار با OpenAI است که از داخلِ ایران جواب می‌دهد. مدل با سنجش انتخاب شد، نه با جدولِ قیمت: روی ۱۰۲ مسئله‌ی تصحیح‌شده ۹۵–۹۶ درست در برابرِ ۸۵–۹۰ برای `gpt-4o-mini` ؛ هر جواب با پرامپتِ امروز حدودِ ۰٫۲ سنت (≈ ۶۰۰ تومان؛ `docs/ai-cost-and-guardrails.md`). دستیار فقط درباره‌ی ریاضی حرف می‌زند و سقفِ روزانه‌اش برای کاربرِ رایگان ۳ و برای مشترک به اندازه‌ی پلن است.	OpenAI مستقیم از ایران در دسترس نیست. مدل‌های گران‌تر در دورِ اولِ سنجش (۱۴ مدل) مزیتی نشان ندادند.	کُندیِ سرویس (حتی «سلام» ۷ ثانیه) دستِ ما نیست؛ اعتبار دو بار تمام شده.
ورود	**توکنِ امضاشده‌ی خودمان** + کدِ پیامکی	کاربرها با شماره وارد می‌شوند، نه نام‌کاربری/رمز؛ مدلِ کاربرِ جنگو به این نمی‌خورد. توکن با `django.core.signing` امضا می‌شود؛ یک وابستگیِ کمتر از کتابخانه‌ی JWT.	JWT همان کار را می‌کند با یک کتابخانه‌ی اضافه.	امنیتش به کلیدِ امضا بسته است (بخشِ ۱۲، موردِ ۱).
پیامک / پرداخت	**کاوه‌نگار** · **زرین‌پال** · **کافه‌بازار**	سرویس‌های ایرانیِ رایج؛ کاوه‌نگار با «ارسال با الگو» که روی خطِ شرکت فعال است؛ زرین‌پال برای وب، بازار برای خریدِ درون‌برنامه‌ایِ اندروید.	—	پرداختِ واقعی تا انتها هنوز آزموده نشده.
اجرا	**gunicorn** (gthread) پشتِ **nginx** پشتِ **ArvanCloud**	الگوی استانداردِ اجرای جنگو. آروان CDN و سپرِ حمله‌ی ایرانی است. WhiteNoise فایل‌های استاتیکِ ادمین را از خودِ جنگو سرو می‌کند.	—	سرور مشترک است؛ مالِ ما فقط یک پوشه و یک سرویس.

اصلِ کلیِ انتخاب‌ها

هر جا ممکن بود، ابزارِ آماده و رایج انتخاب شده است؛ هر جا انتخاب بر کیفیت اثر مستقیم داشت (مانندِ مدلِ هوشِ مصنوعی)، گزینه‌ها پیش از انتخاب با داده‌ی واقعی سنجیده شده‌اند.

۳. معماریِ کلی

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



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

درونِ بک‌اند: «یک برنامه، چند اپِ مستقل» (Modular Monolith)

بک‌اند یک برنامه‌ی جنگوست که به ۱۴ «اپ» تقسیم شده (اپِ `family` ۱۰ مهر ۱۴۰۵ حذف شد). 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 (بخشِ ۱۴).

اپ	مسئولیت	مدل‌ها (جدول‌ها)
`accounts`	هویت، ورود با کدِ پیامکی، نشستِ ورود، پروفایل	`User Profile Session OTPCode`
`content`	برنامه‌ی درسی، سؤال، آزمونِ جزیره، ثبتِ جواب و پیشرفت	`Grade Unit Node Lesson Question Choice UserAnswer UserLessonCompletion UserNodeCompletion NodeProgress QuestionReport`
`progress`	امتیاز، سکه، قلب، زنجیره، جدولِ رتبه، همراه‌ها	`UserStats UserHearts StreakRecord ActiveDay LeaderboardEntry Character`

اپ	مسئولیت	مدل‌ها (جدول‌ها)
`ai_tutor`	گفتگو با دستیار، سهمیه، بازخوردِ 👍/👎	`Conversation Message MessageFeedback AskIdempotency`
`rag`	خواندنِ کتاب، Qdrant، ساختِ پرامپت، شکل‌ها	`LessonChunk LessonChunkImage IngestedPage`
`subscription`	پلن‌ها (بسته‌ی سوال، اشتراکِ قدیمی)، دفترِ اعتبارِ سوال، قلبِ نامحدودِ بسته	`Plan Subscription QuestionCredit HeartsPass`
`payment`	سفارش، تراکنش، زرین‌پال، بازار، لغو و بازپرداخت از ادمین	`Order Transaction BazaarCredential`
`discount`	کدِ تخفیف و پیشنهادها	`DiscountCode Redemption Offer UserOffer`
`affiliate`	معرفی به دوستان و پورسانت	`AffiliateAccount Conversion`
`notifications`	اعلان‌های داخلِ اپ	`Notification`
`support`	تیکت و گزارشِ باگ	`Ticket TicketMessage BugReport`
`campaigns`	پیامِ گروهی (ذخیره و هدف‌گیری؛ زمان‌بندیِ خودکار ندارد)	`MessageCampaign CampaignDelivery`
`statuslog`	ثبتِ عمومیِ «وضعیتِ فلان چیز از X به Y رفت»	`StatusChangeLog`
`config`	تنظیمات، مسیرهای اصلی، middleware، سلامت، مدیریتِ خطا	—

دو قاعده‌ی معماری که از کد پیدا نیست

۱. بینِ اپ‌ها کلیدِ خارجی (FK) نیست

اپِ `progress` کاربر را با یک عددِ ساده نگه می‌دارد، نه با اتصالِ دیتابیسی 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» سرویسِ جدا شد، زنجیره‌ای از وابستگی پاره نمی‌شود. داخلِ 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 تابعِ سرویسِ اپِ دیگر را صدا می‌زند، و هر اپ کارِ خودش را می‌کند. فایل‌های مربوط:

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

۴. ساختارِ فایل‌ها و نقشِ هر فایل

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

۴-۱ مخزنِ بک‌اند (`Tadrisino/MathApp`)

```
MathApp/
├── config/                  تنظیمات و سیم‌کشیِ کلِ پروژه
│   ├── settings.py          همه‌ی تنظیمات؛ خواندنِ .env. نرخ throttle ؛ مدل‌های AI ؛ کلیدِ امضا ؛
│   ├── urls.py              ریشه‌ی مسیرها: /admin، /api/docs، /api/health و include هر اپ
│   ├── middleware.py        سقفِ ورودِ ادمین، no-store، هر درخواست JSON لاگِ
│   ├── exceptions.py        هر خطا → پاسخِ JSON یکدست، بی‌جزئیاتِ داخلی
│   ├── health.py            GET /api/health: دیتابیس، migration، کش، محتوا، هشدارها
│   └── log_formatters.py    قالبِ JSON لاگ
├── accounts/  content/  progress/  ai_tutor/  rag/  subscription/  payment/
├── discount/  affiliate/  notifications/  support/  campaigns/  statuslog/
├── content/seed/            curriculum.json و questions.json — کلِ محتوا به‌صورتِ فایل در git
├── data/textbooks/          PDF کتاب‌ها (در git فقط؛ نیست README)
├── templates/tadrisino/     پوسته‌ی راست‌به‌چپِ Swagger و ReDoc
├── tools/fake_openai.py     جایگزینِ GapGPT برای تستِ بی‌هزینه
├── docs/                    operations، architecture، security-before-production، content-
authoring، …
├── deploy.sh                کلِ انتشار در یک اسکریپت (بخشِ ۱۳)
├── gunicorn.conf.py         worker، timeout ۱۲۰ نخ در هر ۸
├── requirements.txt         Django 5.2، DRF، drf-spectacular، openai، qdrant-client، pymupdf،
…
├── docker-compose.yml       فقط محلی: Loki + Alloy + Grafana برای دیدنِ لاگ
└── manage.py
```

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

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

فایل	چه چیزی در آن است	نمونه‌ی واقعی
`urls.py`	نگاشتِ آدرس به کلاسِ ویو	→ `questions/<int:question_id>/answer` `QuestionAnswerView`
`views.py`	منطقِ هر اندپوینت: چه کسی مجاز است (`permission_classes`)، چند بار (`throttle_scope`)، و کار اصلی	`QuestionAnswerView` ، `NodeQuizView` ، `NodeQuizCompleteView`
`serializers.py`	شکلِ ورودی و خروجی؛ اعتبارسنجی؛ همین فایل Swagger را می‌سازد	`AnswerRequestSerializer` (فقط `selectedOptionId`)
`services.py`	منطقِ مشترکی که چند ویو یا اپِ دیگر لازم دارند	درصدِ پیشرفت، جزیره‌ی بعدی
`models.py`	جدول‌های دیتابیس و قیدهای یکتایی	`UserAnswer` با قیدِ یکتا روی کلیدِ idempotency

فایل	چه چیزی در آن است	نمونه‌ی واقعی
`migrations/`	تاریخچه‌ی تغییرِ جدول‌ها؛ `migrate` روی سرور اعمالشان می‌کند	`0008`: id هر پایه = شماره‌ی پایه
`admin.py`	صفحه‌های پنلِ ادمین، اکشن‌ها، واردکردنِ فله‌ای	«وارد کردنِ سؤال‌ها از فایل»
`receivers.py`	واکنش به رویدادِ اپ‌های دیگر	باطل کردنِ کشِ نقشه وقتی سؤال عوض شد
`management/commands/`	دستورهای خطِ فرمان (`python manage.py` …)	`seed_content`، `generate_questions_from_chunks`
`tests*.py`	تست‌ها؛ هر باگِ پیداشده یک تست دارد	`tests_frontend_contract.py` — نگهبانِ قرارداد با فرانت

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

```
src/
├── main.tsx            نقطه‌ی شروع؛ App را سوار می‌کند
├── config.ts           نشانیِ API: dist/config.js ← VITE_API_BASE_URL ← /api
├── api/                نمی‌زند fetch تنها درِ ارتباط با سرور — هیچ جای دیگر
│   ├── http.ts         توکن، timeout، تکرارِ GET، refresh تکنسخه‌ای، SSE، Idempotency-Key
│   ├── index.ts        یک تابع برای هر اندپوینت: api.nodeQuiz، api.lesson.answer، …
│   └── types.ts        شکلِ پاسخ‌ها، از روی پاسخِ واقعیِ سرور
├── app/                App.tsx: جریانِ ورود · MainApp.tsx: تب‌ها و صفحه‌ها
├── nav/                گوشیِ back ناوبریِ پشته‌ای خودمان، هماهنگ با دکمه‌ی
├── state/              داده‌ی مشترک (پروفایل، نقشه) و گفتگوی دستیار
├── screens/            یک فایل برای هر صفحه: HomeScreen، QuizScreen، AskScreen،
ReportScreen، …
├── components/         اجزای مشترک: مسیرِ جزیره‌ها، برگه‌ی پایین، نوارِ پایین
├── mascots/            از کد، ۶ حالتِ چهره SVG — سیگما، دلتا، پی، تدی
├── diagram/            JSON شکلِ دستیار → SVG می‌شود؛ مختصات، ریشه، زاویه اینجا حساب
└── lib/                پیامِ خطای دوستانه، صدا، MathText، رقمِ فارسی، تاریخِ شمسی
```

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

۵. سفرِ یک درخواست، فایل‌به‌فایل

نمونه: ثبتِ پاسخِ یک دانش‌آموز به یک سؤال در آزمونِ جزیره. در ادامه مسیرِ کاملِ این درخواست از رابطِ کاربری تا دیتابیس و بازگشتِ پاسخ آمده است. سایرِ درخواست‌ها نیز همین مسیر را طی می‌کنند و تنها ویوِ پاسخ‌دهنده تفاوت دارد.

1. فرانت · صفحه — `src/screens/QuizScreen.tsx` — کاربر گزینه‌ای را انتخاب می‌کند و صفحه تابعِ API مربوط را فراخوانی می‌کند. صفحه نمی‌داند نشانیِ سرور چیست.

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

2. فرانت · api — `src/api/index.ts` — نامِ تابع را به نشانیِ API ترجمه می‌کند. برای هر جواب یک کلیدِ 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 })
```

3. فرانت · http — `src/api/http.ts` — توکن را از `localStorage` می‌خواند، هدرها را تنظیم می‌کند و درخواست را با محدودیتِ 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…      ← user_id:زمان:امضا
Idempotency-Key: 7f3c…                   {"selectedOptionId": 7290}
```

4. شبکه — اینترنت ← **nginx** ← **ArvanCloud** روی سرور ← سوکتِ یونیکس ← یک نخِ آزاد از **gunicorn** (`gunicorn.conf.py`).

5. بک‌اند · middleware — `config/settings.py → MIDDLEWARE` — لایه‌های عمومی به‌ترتیب: امنیت، **CORS** (آیا درخواست از `math.tadrisino2.ir` آمده؟)، فایل‌های استاتیک، سقفِ ورودِ ادمین، …، `APINoStoreMiddleware` (پاسخ در کشِ آروان نماند)، و در آخر `APIRequestLoggingMiddleware` که در پایان یک خطِ JSON لاگ می‌نویسد (`config/middleware.py`).

6. 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
```

7. بک‌اند · هویت — `accounts/authentication.py` — امضا و انقضای توکن را بی‌رجوع به دیتابیس می‌سنجد و `request.user` را می‌سازد. بی‌هدر یعنی «مهمانِ 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
```

8. بک‌اند · مجوز و سقف — `content/views.py → QuestionAnswerView` — فقط کاربرِ واردشده، حداکثر ۶۰ جواب در دقیقه (نرخ در `DEFAULT_THROTTLE_RATES` ِ settings).

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

9. بک‌اند · اعتبارسنجی — `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()
```

10. بک‌اند · دیتابیس — ثبتِ جواب و کم کردنِ 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()
```

قیدِ یکتای `(user_id, question, idempotency_key)` در دیتابیس آخرین خطِ دفاع است: درخواستِ تکراری نمی‌تواند ردیفِ دوم بسازد. بعد روزِ فعالِ زنجیره ثبت می‌شود (`progress/services.py`).

11. بک‌اند · پاسخ

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

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

12. فرانت · نمایش — `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 پی بپرس چرا غلط شد» را نشان می‌دهد. پیامِ خطا همیشه از `lib/errors.ts → friendlyError` می‌آید، نه متنِ خامِ سرور.

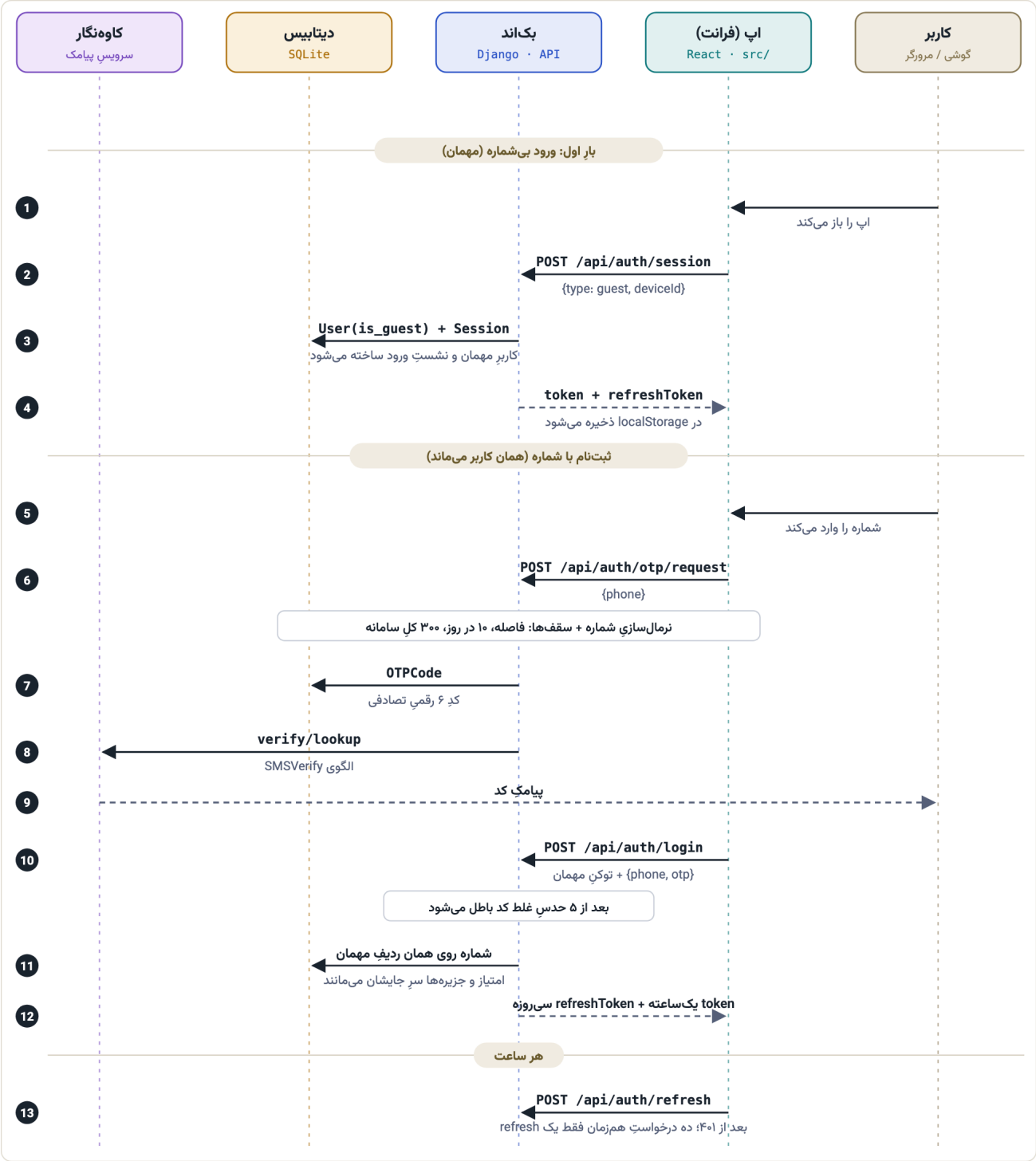
خلاصه‌ی مسیرِ درخواست

صفحه تابعِ API را صدا می‌زند؛ `http.ts` توکن را می‌گذارد و می‌فرستد. از آروان و nginx به gunicorn می‌رسد. جنگو از middlewareها رد می‌کند، `urls.py` ویو را پیدا می‌کند، توکن در `authentication.py` سنجیده می‌شود، ویو مجوز و سقف را چک می‌کند، serializer ورودی را، و منطق در یک تراکنشِ اتمیک در دیتابیس می‌نویسد. پاسخ JSON است و برای هر درخواست یک خطِ لاگِ JSON ثبت می‌شود.

۶. سناریوهای اصلی

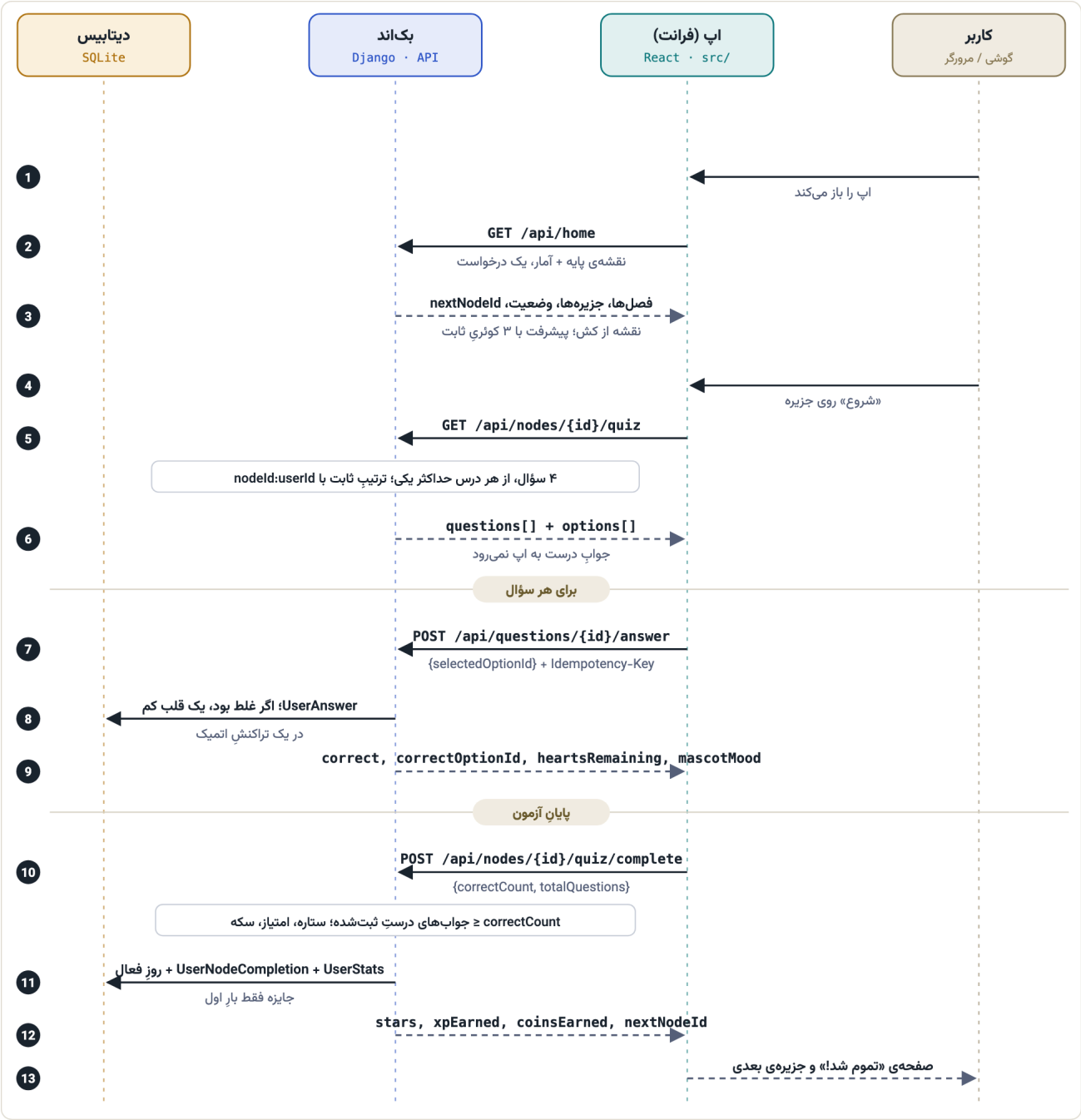
هر نمودار از راست به چپ خوانده می‌شود: ستون‌ها بازیگرها هستند، پیکان‌ها درخواست‌ها. پیکانِ خط‌چین یعنی پاسخ. شماره‌ها ترتیبِ زمانی‌اند.

۶-۱ ورود: مهمان، بعد ثبت‌نام با شماره



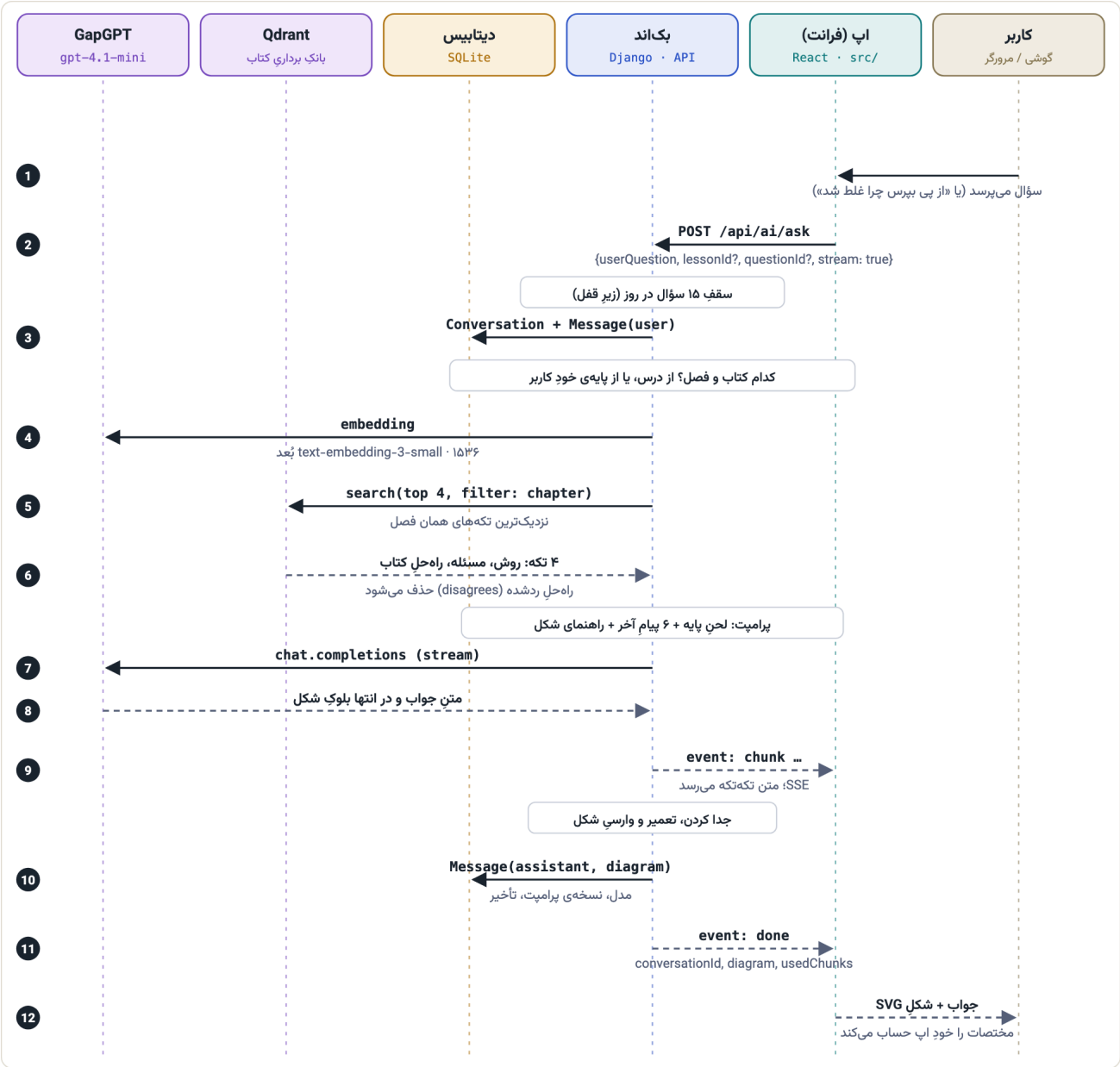
کد: `accounts/views.py` · `accounts/phone.py` *(نرمال‌سازیِ شماره)* · `accounts/services.py` *(ارسالِ پیامک)* · `accounts/authentication.py` *(توکن)* · *تست‌ها:* `accounts/tests_otp_limits.py`

۶-۲ بازیِ یک جزیره



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

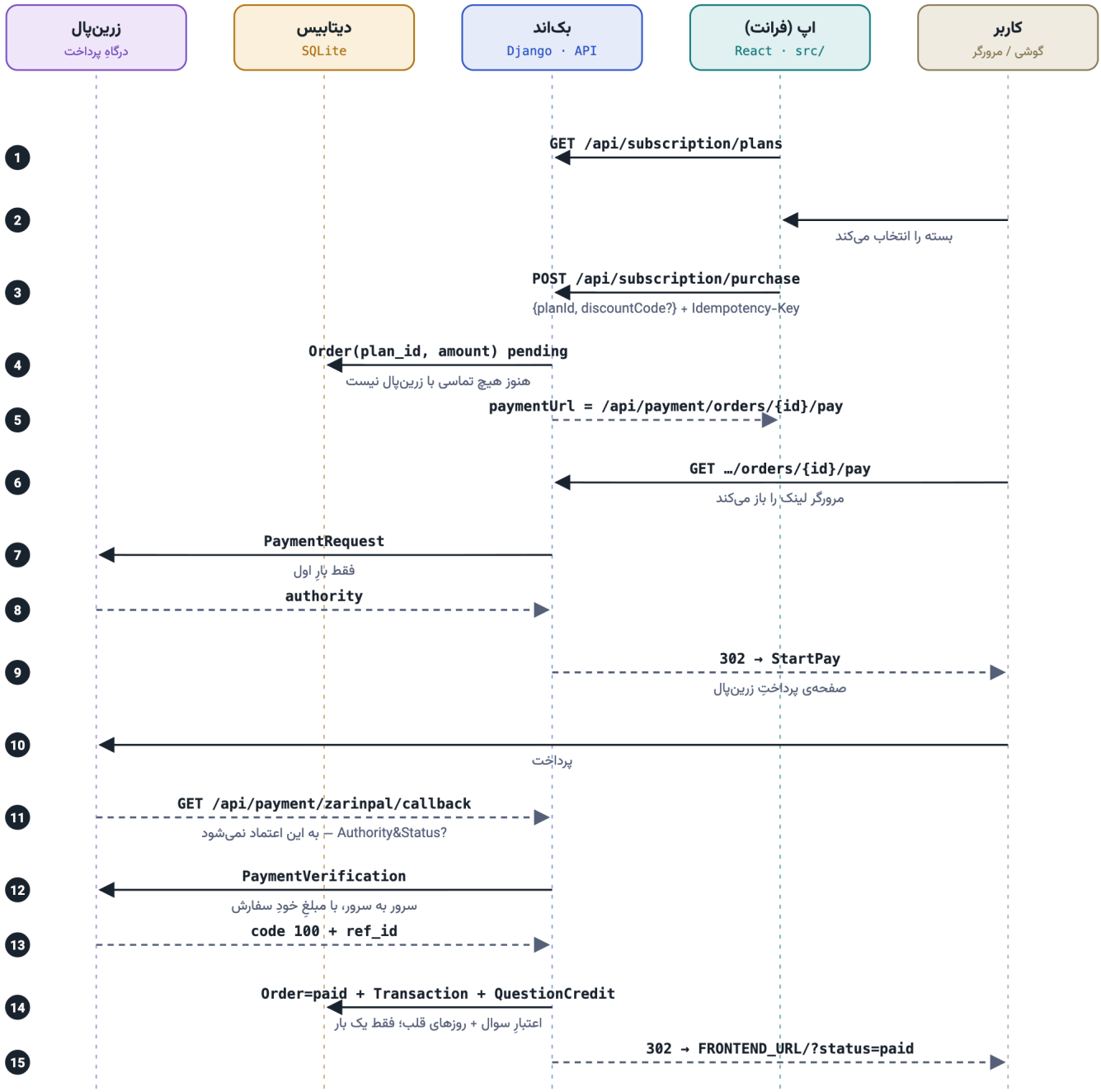
۶-۳ «از پی بپرس» — دستیار با کتاب



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

سه نکته‌ی طراحی: (۱) تماس با GapGPT **بیرون از تراکنشِ دیتابیس** است، وگرنه ۳۰ ثانیه انتظار دیتابیس را قفل می‌کرد؛ (۲) اگر GapGPT خطا داد، سهمیه‌ی رزروشده پس داده می‌شود و کاربر ۵۰۳ می‌گیرد، نه ۵۰۰؛ (۳) اولین پیامِ هر گفتگو کش می‌شود — سؤالِ تکراری از کش ۰٫۰۱ ثانیه‌ای برمی‌گردد در برابرِ ~۱۱ ثانیه.

۶-۴ خریدِ بسته‌ی سوال با زرین‌پال



کد: `subscription/views.py` · `payment/services.py` · `subscription/services.py → grant_pack` · `payment/views.py` (`PaymentRedirectView`، `ZarinpalCallbackView`) · `payment/zarinpal.py` · *تست‌ها:* `payment/tests_audit_fixes.py` · `subscription/tests_packs.py` · `payment/tests_refund.py`

از ۱۰ مهر ۱۴۰۵ آنچه فروخته می‌شود **بسته‌ی سوال** است، نه اشتراکِ ماهانه/سالانه: ۲۰ / ۶۰ / ۱۵۰ سوال از دستیار به ۴۵ / ۱۱۹ / ۲۵۹ هزار تومان، بی‌انقضا، با ۷ / ۳۰ / ۶۰ روز قلبِ نامحدودِ هدیه. پرداختِ تأییدشده یک ردیفِ مثبت در دفترِ `QuestionCredit` و روزهای `HeartsPass` می‌سازد و به اشتراکِ کسی دست نمی‌زند. پلن‌های اشتراکیِ قبلی پاک نشدند (`on_sale=False`) تا مشترکِ فعلی سقفش را نگه دارد.

چرا این‌قدر پیچیده؟

- **ساختِ سفارش به زرین‌پال وابسته نیست:** اگر زرین‌پال قطع باشد، سفارش ساخته می‌شود و فقط رفتن به درگاه عقب می‌افتد.
- **برگشتِ کاربر هیچ چیزی را ثابت نمی‌کند:** هر کس می‌تواند آدرسِ برگشت را با `Status=OK` بسازد. فقط تأییدِ سرور-به-سرور سفارش را «پرداخت‌شده» می‌کند.
- **پرداختِ جاافتاده بازسنجی می‌شود:** اگر زرین‌پال موقعِ برگشت جواب نداد، هر بار که اپ وضعیتِ خرید را می‌خواند سفارش دوباره سنجیده می‌شود (و دستورِ `payment_reconcile`).
- **پلن روی خودِ سفارش است** ؛ کدِ تخفیفِ ۱۰۰٪ بی‌درگاه تمام می‌کند؛ تأییدِ تکراری اعتبارِ دوباره نمی‌دهد (قیدِ یکتا روی سفارش).
- **کافه‌بازار:** سفارشِ بازار با `planId` ساخته می‌شود (`POST /api/payment/orders`)؛ تا ۱۰ مهر این فیلد نبود و خریدِ بسته از بازار هیچ اعتباری نمی‌داد (در بازبینی پیدا و با تست بسته شد).
- **لغو و بازپرداخت از پنلِ ادمین:** دو action روی سفارش‌ها؛ بازپرداخت پس از صفحه‌ی تأیید، اعتبارِ خرج‌نشده و روزهای قلب یا اشتراکِ همان سفارش را پس می‌گیرد. خودِ پول از پنلِ درگاه برگردانده می‌شود (`docs/operations.md`).

آزموده‌نشده زرین‌پال درخواستِ پرداخت را با کدِ ۱۰۰ می‌پذیرد، ولی یک پرداختِ واقعی تا انتها هنوز انجام نشده.

۶-۵ گزارشِ والدین

والد حسابِ جدا و کدِ دعوت ندارد (تا مهرِ ۱۴۰۵ داشت؛ به تصمیمِ پس از ارائه حذف شد). صفحه‌ی «گزارشِ والدین» در پروفایلِ خودِ دانش‌آموز است.

#	درخواست	چه می‌شود
۱	`GET /api/user/weekly-report`	برای هر یک از ۷ روزِ اخیر و ۷ روزِ قبلش: زمانِ تمرین، سوالِ حل‌شده و درست، امتیاز، درس و جزیره‌ی تمام‌شده، سوال از دستیار. فقط دیتای خودِ صاحبِ توکن.
۲	—	فرانت جمله‌ی مقایسه («آفرین! این هفته ۲۰ دقیقه بیشتر...») و دو نمودار می‌سازد: زمانِ هر روز در برابرِ همان روزِ هفته‌ی پیش، و سوال‌های درست/نادرستِ هر روز.
۳	`GET /api/progress/nodes`	پارامترِ `userId` برداشته شد؛ هر کاربر فقط پیشرفتِ خودش را می‌بیند.

تخمینی: «زمانِ تمرین» اندازه‌گیری نیست: از فاصله‌ی جواب‌ها و پرسش‌ها حساب می‌شود (فاصله‌ی بیش از ۱۰ دقیقه وقفه است). خواندنِ درس بدونِ جواب دادن در آن نیست، پس عدد معمولاً از زمانِ واقعی کمتر است.

۷. مدلِ داده

جدول‌های اصلی، دسته‌بندی‌شده بر اساسِ اپ. `→` یعنی کلیدِ خارجیِ واقعی (داخلِ همان اپ)، و `user_id` یعنی ارجاعِ سبک به اپِ دیگر.

accounts هویت

- `User phone, name, is_guest, device_id, grade_id, tokens_valid_after`
- `Profile → User · birth_year, parent_phone`
- `Session → User · refresh_token, expires_at, revoked`
- `OTPCode phone, code, expires_at, failed_attempts`

content محتوا

- `Grade id = (هشتم = ۸) شماره‌ی پایه`
- `Unit → Grade · title, blurb, order`
- `Node → Unit · order (جزیره)`
- `Lesson → Node · kind, example, steps`
- `Question → Lesson · prompt, type`
- `Choice → Question · value, is_correct`

content فعالیتِ کاربر

- `UserAnswer user_id, → Question, → Choice`
- `UserLessonCompletion user_id, → Lesson`
- `UserNodeCompletion user_id, → Node, stars, xp_earned, coins_earned`
- `NodeProgress user_id, → Node, status`
- `QuestionReport «این سؤال ایراد دارد»`

progress بازی

- `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`

ai_tutor · rag دستیار

- `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`

subscription · payment پول

- `Plan kind (pack/subscription), price, question_credits, hearts_days, on_sale, ai_daily_quota`
- `Subscription user_id, → Plan, status, expires_at`
- `QuestionCredit user_id, delta (+بازپرداخت / -سوال / -خرید), order_id, message_id`
- `HeartsPass user_id, expires_at قلبِ نامحدودِ بسته`
- `Order user_id, plan_id, amount, list_price, plan_title, status (… canceled, refunded), gateway_authority`
- `Transaction → Order · gateway_ref_id, status, paid_at`
- `BazaarCredential توکنِ اتصال به بازار`

discount · affiliate بازاریابی

- `DiscountCode code, type, value, max_uses`
- `Redemption یکتا: هر کاربر × هر کد`
- `Offer / UserOffer پیشنهادِ خودکار`
- `AffiliateAccount user_id, referral_code, commission_rate`
- `Conversion → AffiliateAccount · order_id, commission_amount`

support · notifications ارتباط

- `Ticket user_id, subject, status`
- `TicketMessage → Ticket · sender, text`
- `BugReport`
- `Notification user_id, title, body, is_read`

campaigns · statuslog

- `MessageCampaign, CampaignDelivery یکتا: کمپین × کاربر × کانال`
- `StatusChangeLog label, object_id, old_status → new_status`

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

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

قید	از چه جلوگیری می‌کند
یکتا: `UserAnswer(user_id, question, idempotency_key)`	ثبتِ دوباره‌ی یک جواب با تکرارِ درخواست
یکتا: `Redemption(کاربر، کد)`	استفاده‌ی دوباره از یک کدِ تخفیف
یکتا: `QuestionCredit(order_id)` برای ردیفِ خرید	اعتبارِ دوباره وقتی تأییدِ پرداخت دو بار برسد
یکتا: `CampaignDelivery(کمپین، کاربر، کانال)`	پیامِ تکراری وقتی کمپین دو بار هم‌زمان اجرا شود

قید	از چه جلوگیری می‌کند
یکتا: `LessonChunk(collection, qdrant_point_id)` و `IngestedPage(collection, page)`	تکه‌ی تکراری در Qdrant وقتی خواندنِ کتاب از سر گرفته می‌شود

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

۸. API: قواعد و فهرستِ کامل

مستندِ زنده و قابلِ امتحان: `https://mathapi.tadrisino2.ir/api/docs/` (Swagger). جدول‌های زیر از همان اسکیما استخراج شده‌اند.

قواعدِ مشترکِ همه‌ی اندپوینت‌ها

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

فهرستِ ۵۹ اندپوینت

عمومی بی‌ورود · ورود توکن لازم · داخلی با هدرِ کلیدِ داخلی (سرور-به-سرور)

روش	مسیر	دسترسی	کار
هویت و پروفایل — `accounts`			
POST	`/api/auth/session`	عمومی	ساختِ کاربرِ مهمان با `deviceId`
POST	`/api/auth/otp/request`	عمومی	فرستادنِ کدِ ورود
POST	`/api/auth/login`	عمومی	ورود با شماره و کد
POST	`/api/auth/refresh`	عمومی	توکنِ تازه با refresh token
POST	`/api/auth/logout`	ورود	خروج؛ توکن‌های قبلی باطل
GET · PATCH	`/api/user/profile`	ورود	خواندن و ویرایشِ پروفایل (تاریخِ تولدِ شمسی، شناسه‌ی نمایشی)

روش	مسیر	دسترسی	کار
POST	`/api/user/delete`	ورود	حذفِ حساب؛ شماره‌دار با کدِ پیامکیِ تازه، مهمان با تایپِ «حذف»
POST	`/api/user/grade`	ورود	ثبتِ پایه
محتوا و آزمون — `content`			
GET	`/api/grades`	عمومی	فهرستِ 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	آزمونِ ۴ سؤالیِ جزیره
POST	`/api/nodes/{id}/quiz/complete`	ورود	پایانِ 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	پایانِ درس
POST	`/api/questions/{id}/answer`	ورود	ثبتِ جواب
POST	`/api/questions/{id}/report`	ورود	گزارشِ ایرادِ سؤال
GET	`/api/progress/nodes`	ورود	وضعیتِ خودِ کاربر روی جزیره‌ها
پیشرفت و بازی — `progress`			
GET	`/api/user/summary`	ورود	پروفایل + آمار + زنجیره در یک درخواست
GET	`/api/user/hearts`	ورود	قلب‌ها و زمانِ پر شدن
GET	`/api/user/streak/calendar`	ورود	تقویمِ روزهای فعال
GET	`/api/leaderboard`	عمومی	جدولِ رتبه: هفتگی یا همه‌ی زمان‌ها
GET	`/api/characters`	ورود	همراه‌ها و باز/قفل بودنشان
POST	`/api/characters/select`	ورود	انتخابِ همراه (فقط بازشده)
دستیار — `ai_tutor` و `rag`			
POST	`/api/ai/ask`	ورود	پرسش؛ JSON یا استریم
GET	`/api/ai/quota`	ورود	سوال‌های رایگانِ باقیِ امروز و اعتبارِ بسته

روش	مسیر	دسترسی	کار
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	افزودنِ متن یا PDF به Qdrant
GET	`/api/rag/chunks`	داخلی	مرورِ تکه‌های ذخیره‌شده
بسته، پرداخت، تخفیف			
GET	`/api/subscription/plans`	عمومی	بسته‌های سوالِ در حالِ فروش
POST	`/api/subscription/purchase`	ورود	شروعِ خرید؛ سفارش و لینکِ پرداخت
GET	`/api/subscription/status`	ورود	اعتبارِ سوال، قلبِ نامحدود تا کِی، اشتراکِ قدیمی (و بازسنجیِ پرداختِ جاافتاده)
POST	`/api/payment/orders`	داخلی	ساختِ سفارش (سفارشِ بازار، با `planId` ِ بسته)
GET	`/api/payment/orders/{id}`	ورود	وضعیتِ سفارش
GET	`/api/payment/invoices`	ورود	پرداخت‌ها و فاکتورهای کاربر (شماره‌ی فاکتور، تخفیف، کدِ پیگیری)
GET	`/api/payment/orders/{id}/pay`	عمومی	رفتن به درگاهِ زرین‌پال
GET	`/api/payment/zarinpal/callback`	عمومی	برگشت از درگاه + تأییدِ سرور-به-سرور
POST	`/api/payment/bazaar/verify`	ورود	تأییدِ خریدِ کافه‌بازار
GET	`/api/discount/validate`	ورود	سنجشِ کدِ تخفیف
POST	`/api/discount/redeem`	ورود	ثبتِ مصرفِ کد
GET · POST	`/api/discount/offers/mine` · `/redeem`	ورود	پیشنهادهای کاربر
اجتماعی و پشتیبانی			
GET · POST	`/api/affiliate/me` · `/track` · `/conversions`	ورود	کدِ معرف، ثبتِ معرفی، پورسانت‌ها
GET	`/api/notifications`	ورود	اعلان‌ها
PATCH	`/api/notifications/{id}/read` · `/read-all`	ورود	خوانده‌شده
GET · POST	`/api/support/tickets`	ورود	تیکت‌های من / تیکتِ تازه
GET · POST	`/api/support/tickets/{id}` · `/messages`	ورود	جزئیات و پیامِ تازه
POST	`/api/support/bug-reports`	ورود	گزارشِ باگ

روش	مسیر	دسترسی	کار
GET	`/api/user/weekly-report`	ورود	«گزارشِ والدین»: ۷ روزِ اخیر در کنارِ ۷ روزِ قبلش
سیستم			
GET	`/api/health`	عمومی	سلامتِ سرویس (بخشِ ۱۳)

۹. دستیارِ هوشِ مصنوعی و داده‌ی Qdrant

دستیار «از حافظه‌ی مدل» جواب نمی‌دهد؛ اول تکه‌های مرتبطِ 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 «بانکِ برداری» است: هر تکه‌ی کتاب به یک بردار (۱۵۳۶ عدد) تبدیل می‌شود که «معنا»ی آن را نشان می‌دهد؛ سؤالِ 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 روشِ حل، صورتِ مسئله و راه‌حلِ کتاب.

collection (کتاب)	تکه	صفحه‌ی خوانده‌شده	چه کسی ساخت
ریاضی هشتم	۵۱۸	۱۵۰	ما — خواندنِ 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	۴۳۶ · ۴۳۷ · ۳۷۹	—	تیمِ دیگرِ شرکت؛ با `rag_import_collection` خوانده شد
علوم تجربی سوم · مطالعات اجتماعی ششم و چهارم	۷۰ · ۴۰ · ۲۷	—	تیمِ دیگر (در اپِ 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": ["مرکز","شعاع","کره"],
  "generalSolution": "۱. تعریفِ دایره را…",
  "problem": "…",  "solution": "…",
  "solution_verdict": "agrees",
  "source_page": 139, "chapter_number": 8 }
```

ساخته در `rag/services/ingest.py → _payload`

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

1. PDF کتاب /data/textbooks + نقشه‌ی فصل‌ها در textbooks.py
2. تصویرِ صفحه ۲۰۰ DPI pdf2image
3. مدلِ بینایی gpt-4.1-mini ~۰٫۲۲ سنت برای هر صفحه
4. JSON ساختاریافته نوع، کلیدواژه، روش، مسئله، راه‌حل
5. LessonChunk ردیف در دیتابیس (منبعِ اصلی، با بردار)
6. Qdrant upsert نقطه + payload IngestedPage برای ازسرگیری

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

۹-۳ کنترلِ کیفیت: راستی‌آزماییِ راه‌حل‌های کتاب

مدلِ بینایی گاهی عدد را بد می‌خواند یا راه‌حل را به مسئله‌ی کناری می‌چسباند. دستورِ `rag_verify_solutions` هر مسئله را مستقل دوباره حل می‌کند و رأیِ سه‌حالته می‌دهد:

رأی	تعداد	اثر
agrees راه‌حلِ کتاب درست است	۲,۴۶۶	فقط از این‌ها سؤال ساخته می‌شود
disagrees راه‌حلِ کتاب غلط است	۳۵۴	راه‌حلش به دستیار داده نمی‌شود؛ فقط روش و صورتِ مسئله
unjudgeable بی‌شکل قابلِ قضاوت نیست	۵۰۸	کنار می‌ماند
بررسی‌نشده	۱۸۹	علوم و مطالعاتِ تیمِ دیگر

متنِ اصلیِ کتاب هیچ‌وقت بازنویسی نمی‌شود؛ جوابِ مستقل جدا در `checked_answer` ذخیره می‌شود تا آدم بتواند داوری کند.

۹-۴ افزودن و پاک کردنِ داده

کار	روش	هزینه
افزودنِ کتابِ تازه	۱) PDF را در `data/textbooks/` بگذارید و فصل‌ها و `page_offset` را در `rag/services/textbooks.py` ثبت کنید. ۲) `python manage.py rag_ingest_book --grade نهم --plan` ← فقط نشان می‌دهد چه چیزی و با چه هزینه‌ای خوانده می‌شود. ۳) همان دستور بدونِ `--plan` (قابلِ قطع و ازسرگیری؛ صفحه‌ی خوانده‌شده دوباره خریده نمی‌شود). ۴) `rag_verify_solutions` ← ۵) `seed_curriculum_from_textbook` ← ۶) ساختِ سؤال (بخشِ ۱۰).	~۰٫۲۲ سنت/صفحه
افزودنِ یک متن یا PDF	`POST /api/rag/ingest/text` یا `/pdf` با هدرِ `X-Internal-Key`.	کم
آوردنِ collectionِ تیمِ دیگر	`rag_import_collection --collection "ریاضی ششم"` — فقط می‌خواند، چیزی در Qdrant نمی‌نویسد.	رایگان
بازخوانیِ صفحه‌های معیوب	`rag_ingest_book --grade نهم --refresh figures` یا `--refresh-pages 12,15,20-24`. هر صفحه اول پاک می‌شود (اول Qdrant، بعد دیتابیس — تابعِ `clear_page`) و دوباره خوانده می‌شود.	مثلِ خواندن
پاک کردن	دستورِ جداگانه‌ی «فقط پاک کن» نداریم. برای یک صفحه تابعِ `rag.services.ingest.clear_page(collection, page)` از `manage.py shell`. برای کلِ collection، `delete_collection` کلاینتِ Qdrant — **فقط با هماهنگیِ تیمِ دیگر**، چون Qdrant مشترک است.	—

دو احتیاط (از خواندنِ کد؛ آزموده نشده)

- پاک کردنِ یک `LessonChunk` از پنلِ ادمین نقطه‌یِ Qdrant آن را پاک نمی‌کند (هیچ شنونده‌ای برایش نیست)؛ نقطه بی‌صاحب می‌ماند و در جست‌وجو پیدا می‌شود. از `clear_page` استفاده کنید.
- ردیف‌های `LessonChunk` (نسخه‌ی ساختاریافته با بردار) در دیتابیسِ محلیِ توسعه‌اند، نه روی سرور؛ سرور فقط از Qdrant می‌خواند. از آن فایل بکاپ جدا نگه دارید — با آن، Qdrant بی‌هزینه‌ی embedding دوباره ساخته می‌شود.

۹-۵ 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 خودِ کاربر
جست‌وجو	`retrieval.py` · `qdrant_store.py` · `embedding.py`	متنِ جست‌وجو = سؤال + سؤالِ قبلیِ کاربر (برای «نفهمیدم»)

مرحله	فایل	نکته
پرامپت	`generation.py`	نسخه‌ی `tutor-v10-shape-anchor` ؛ لحنِ سه‌سطحی (دبستان/متوسطه‌ی اول/دبیرستان)؛ متنِ ساده بی‌LaTeX؛ حلِ مرحله‌به‌مرحله با وارسی و « ✅ جواب»؛ ۶ پیامِ آخرِ گفتگو
جوابِ درست/ غلطِ دانش‌آموز	`generation.py → question_focus`	با `questionId` دستیار صورتِ سؤال و آخرین جوابِ دانش‌آموز را می‌بیند و می‌گوید اشتباه کجا بود
تمیزکاری	`mathtext.py`	هرچه LaTeX از دستِ مدل در برود به یونیکد برمی‌گردد (`x²` ، `√3`)
شکل	`diagram.py` ← فرانت `src/diagram/`	مدل فقط توصیفِ JSON می‌نویسد؛ سرور جدا، تعمیر و با فهرستِ سفید وارسی می‌کند؛ اپ مختصات را حساب و SVG رسم می‌کند. ۱۵ نوع شکل — شرحِ کامل در کادرِ بنفشِ پایین.
ثبت	`ai_tutor/models.py → Message`	مدل، نسخه‌ی پرامپت، تعدادِ تکه، تأخیر و شکل برای هر جواب ذخیره می‌شود ← کیفیت قابلِ اندازه‌گیری است

مهم‌ترین ایده‌ی فنیِ دستیار

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

هوشِ مصنوعی ریاضی را **می‌فهمد** ولی در حسابِ دقیقِ مختصات اشتباه می‌کند؛ کامپیوتر برعکس، هیچ‌وقت در حساب اشتباه نمی‌کند ولی مسئله را نمی‌فهمد. پس کار بینِ این دو تقسیم شد: **مدل فقط توصیفِ شکل را به‌صورتِ داده (JSON) می‌نویسد**، سرور آن را وارسی می‌کند، و **کدِ اپ رأس‌ها، ریشه‌ها و زاویه‌ها را خودش حساب و رسم می‌کند.**

نامِ این الگو در مهندسیِ نرم‌افزار: «توصیفِ اعلانی + رسمِ قطعی» (Declarative Spec + Deterministic Rendering) — همان ایده‌ی پشتِ Vega-Lite و Mermaid، این‌جا مخصوصِ کتاب‌های ریاضیِ ایران، با ۱۵ نوع شکل.

گامِ ۱ · سرور → مدل

سرور به مدل می‌گوید اگر شکل لازم است، فقط JSON بنویسد — و فقط بخشِ مربوط از راهنمای شکل‌ها را کنارِ سؤال می‌گذارد (برای «۳۴۵ × ۱۲» هیچ). `rag/services/diagram.py`

متنِ واقعیِ قاعده (DIAGRAM_RULE): «در پایانِ پاسخ دقیقاً یک بلوکِ `<diagram>`…`</diagram>` بگذار که داخلش فقط یک JSON است، نه SVG و نه HTML. برای حسابِ عددی یا ساده‌کردنِ عبارتِ جبری که شکلی ندارد، شکل نگذار.»

گامِ ۲ · مدل

مدل حلِ قدم‌به‌قدم را می‌نویسد و در آخر فقط توصیفِ شکل (~۱۵۰ 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>
```

گامِ ۳ · سرور

سرور بلوک را از متن جدا، اشتباه‌های قالبیِ مدل را تعمیر، و با فهرستِ سفید وارسی می‌کند (رقمِ فارسی در عدد، `2*pi` ، کامنت، گیومه‌ی جاافتاده؛ نوع و رنگ و کلیدِ ناشناخته رد؛ عددها محدود). شکلِ خراب فقط حذف می‌شود و جوابِ متنی سرِ

جایش می‌ماند. نتیجه در `Message.diagram` ذخیره و در رویدادِ `done` فرستاده می‌شود. کارِ 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
```

گامِ ۴ · اپ (گوشی)

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

`src/diagram/shapes.ts` `src/diagram/expr.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, bisect where the sign flips
if (Math.sign(y) !== Math.sign(py)) { const r = bisect(f, px, x); if (Math.abs(f(r)) < 1e-6)
out.push(r); }
```

مثالِ ۱ · مثلثِ قائم‌الزاویه

مدل می‌نویسد

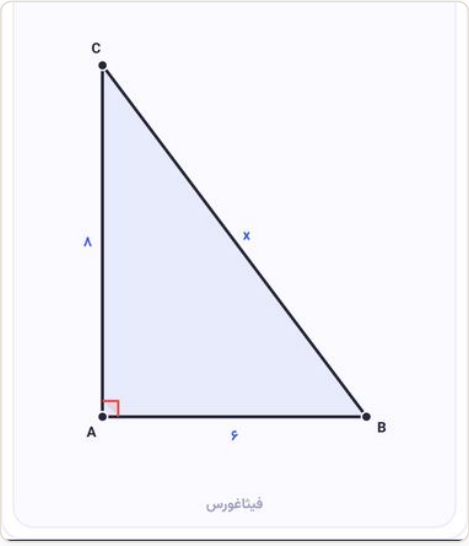
```
{"type":"triangle","names":["A","B","C"],
 "sides":{"AB":6,"AC":8},"angles":{"A":90},
 "labels":{"BC":"x"}}
```

اپ حساب می‌کند

قانونِ کسینوس‌ها: $BC^2 = 6^2 + 8^2 - 2\cdot6\cdot8\cdot\cos 90° = 100$

پس **BC = 10** و رأس‌ها A(0,0) · B(6,0) · C(0,8)

دانش‌آموز می‌بیند



مثالِ ۲ · ریشه و رأسِ سهمی

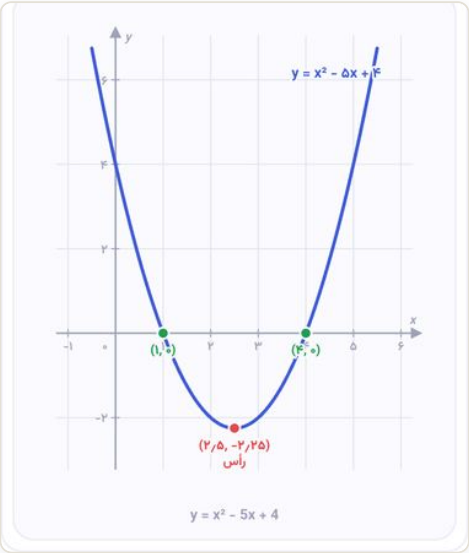
مدل می‌نویسد

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

اپ حساب می‌کند

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

دانش‌آموز می‌بیند



	روشِ رایج: مدل خودش SVG/HTML بنویسد	روشِ ما: مدل توصیف، اپ رسم
امنیت	کدِ مدل در صفحه اجرا می‌شود؛ متنِ مخرب در سؤال می‌تواند کد تزریق کند	هیچ کدی از مدل اجرا نمی‌شود؛ فقط JSON وارسی‌شده
دقت	مختصات را حدس می‌زند؛ پنج‌ضلعی کج، ریشه جابه‌جا	با فرمول حساب می‌شود
هزینه	چند برابر توکن برای هر شکل	~۱۵۰ توکن؛ برای سؤالِ بی‌شکل صفر (۰٫۰۰۰۳ تا ۰٫۰۰۰۸ دلار در هر جوابِ شکل‌دار)
اگر خراب شد	صفحه یا چت به‌هم می‌ریزد	فقط شکل نمی‌آید؛ جوابِ متنی سالم می‌ماند

۴) پس ۲۰ درصد از یک ساعت برابر است با ۱۲ دقیقه.

✅ جواب: ۱۲ دقیقه

در شکلِ پایین، ۲۰ خانه از ۱۰۰ خانه رنگ شده که نشان‌دهنده‌ی ۲۰ درصد است.

حالا خودت امتحان کن: ۱۵ درصد از ۸۰ دقیقه چند دقیقه می‌شود؟ 😊

۲۰ خانه از ۱۰۰ خانه، یعنی ٪۲۰

20 درصد

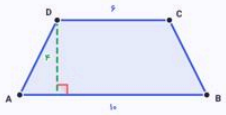
از پی بپرس
دستیار ریاضی، از روی کتاب خودت
گفتگوی تازه

۶) حالا تقسیم بر ۲:
۶۴ ÷ ۲ = ۳۲

✅ جواب: مساحت ذوزنقه برابر ۳۲ واحد مربع است.

در شکلِ پایین، ذوزنقه با قاعده‌های ۱۰ و ۶ و ارتفاع ۴ نشان داده شده است که به فهمیدن بهتر کمک می‌کند.

حالا خودت امتحان کن: مساحت ذوزنقه‌ای با قاعده‌های ۸ و ۵ و ارتفاع ۳ چقدر است؟



ذوزنقه با قاعده‌های 10 و 6 و ارتفاع 4

سوالت رو بنویس...
امروز ۱۴ سوالِ دیگه می‌تونی بپرسی
خانه نقشه دستیار اشتراک من

فایل‌ها — بک‌اند: `rag/services/diagram.py` (قاعده، تشخیص، تعمیر، وارسی) · `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` (انتخابِ نوع، محافظِ خطا) · `plane.tsx` · `shapes.ts` · `expr.ts`

۹-۶ کیفیت با عدد

سنجش	نتیجه
انتخابِ مدلِ جواب: ۱۰۲ مسئله‌ی تصحیح‌شده‌ی دستی	`gpt-4.1-mini` : ۹۵–۹۶ درست · `gpt-4o-mini` : ۸۵–۹۰
شکل‌ها: ۶۴ مسئله‌ی تصادفیِ کتاب، وارسیِ چشمیِ تک‌تک	۴۰ درست، ۱۲ قابلِ قبول، ۷ غلط، ۵ بی‌شکل — حدودِ یکی از پنج هنوز ایراد دارد؛ برای همین شکل همیشه کنارِ حلِ متنی می‌آید
سرعت	جوابِ کامل با شکل ~۳۰ ثانیه؛ اندازه‌گیری شد که در خودِ GapGPT است («فقط بگو سلام» = ۷ ثانیه)
سقف‌ها	۳ سؤالِ رایگان در روز، بعد از اعتبارِ بسته؛ ۲۰ در ساعت؛ مهمان‌ها بر اساسِ IP (تا با ساختِ حسابِ مهمانِ تازه دور زده نشود)

۹-۷ فقط ریاضی، و چه چیزی هزینه دارد

دستیار فقط درباره‌ی ریاضی حرف می‌زند (`rag/services/guardrails.py`). سه لایه دارد: (۱) صافیِ سرور پیش از مدل و بی‌هزینه، فقط برای ترفندهای آشکار («دستورهای قبلی را نادیده بگیر»، «پرامپتت را بگو»)؛ (۲) قاعده‌ی سخت در پرامپت که هر

چیزِ غیرِ ریاضی را با علامتی ثابت جواب می‌دهد؛ (۳) سرور آن علامت را با متنِ ثابتِ رد عوض می‌کند. جمله‌ای درباره‌ی آسیب به خود رد نمی‌شود و جوابی مهربان با شماره‌ی اورژانسِ اجتماعی (۱۲۳) و صدای مشاور (۱۴۸۰) می‌گیرد؛ از ۱۰ مهر هر شکلِ نوشتنش (با فاصله، بی‌نیم‌فاصله) گرفته می‌شود. آزمونِ زنده: ۷ از ۷ تلاشِ خروج از ریاضی رد شد.

قاعده‌ی خرج	
ترتیب	اول ۳ سوالِ رایگانِ همان روز، بعد اعتبارِ بسته (`Message.charge` : `free` / `credit` / `none`)
خرج می‌شود	هر سوالی که به مدل برسد، از جمله ردِ «غیرِ ریاضی» توسطِ مدل
خرج نمی‌شود	ردِ صافیِ سرور، جوابِ کش‌شده، جوابی که ساخته نشد، و تکرارِ درخواست با همان `Idempotency-Key`
هم‌زمانی	دو سوالِ هم‌زمان با یک اعتبار فقط یک جواب می‌گیرند؛ با دو thread واقعی آزموده می‌شود (`ai_tutor/tests_credit_concurrency.py`)
هزینه	هر سوال حدودِ ۰٫۲ سنت؛ قیمتِ اندازه‌گیری‌شده‌ی هر مدل و تاریخچه‌اش در `docs/model-prices.md`

۱۰. محتوای آموزشی و ساختِ سؤال

هدف: محتوا بدونِ برنامه‌نویس قابلِ ویرایش باشد، و یک سرورِ خالی با یک دستور پر شود.

محتوا کجا زندگی می‌کند

فایل	محتوا
`content/seed/curriculum.json`	۸ پایه، ۵۸ فصل، ۲۶۳ جزیره، ۲,۶۱۷ درس
`content/seed/questions.json`	۲,۵۴۵ سؤالِ چهارگزینه‌ای

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

کارهای روزمره از پنلِ ادمین

کار	کجا
افزودن یا ویرایشِ یک سؤال	`/admin/content/question/` ← «افزودن»
واردکردنِ فله‌ای (همه یا هیچ)	همان صفحه ← «وارد کردنِ سؤال‌ها از فایل»
خروجیِ سؤال یا برنامه‌ی درسی	انتخاب ← اکشنِ «دریافتِ فایل» (همان قالبی که ورودی می‌پذیرد)
پایه‌ی تازه	`/admin/content/grade/add/` — عنوانِ «پایه چهارم»، شناسه خودکار ۴
پلن، همراه، تیکت، کدِ تخفیف	صفحه‌ی اپِ مربوط در پنل
سؤال‌های گزارش‌شده	`QuestionReport` — دانش‌آموز «این سؤال ایراد دارد» زده

سؤال‌ها چطور ساخته شدند

1. تکه‌های agrees فقط راه‌حل‌های تأییدشده
2. export generate_questions_from_chunks --export رایگان
3. نویسنده‌ی سؤال subagentهای Claude (Sonnet) با همان قواعدِ پرامپت
4. import با اعتبارسنجی content/question_io.py
5. questions.json seed_content --export و commit

بازده حدودِ ۸۷–۹۲٪ بود؛ بقیه عمداً رد شدند (بیشتر مسئله‌هایی که بی‌شکلِ کتاب خودکفا نمی‌شوند). همین مسیر با API پولی هم ممکن است ولی اندازه‌گیری‌شده ~۰٫۷۷ سنت برای هر سؤال گران‌تر بود بی‌کیفیتِ بهتر. نویسنده‌های سؤال چند بار خطای حسابِ خودِ کتاب را هم گرفتند.

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

۱۱. گیمیفیکیشن و همراه‌ها

سازوکارهای شناخته‌شده‌ی اپ‌های آموزشیِ موفق (مانندِ دولینگو): مسیرِ 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

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

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

قاعده‌های بازی

سازوکار	قاعده	کجای کد
امتیاز (XP)	پایانِ درس یا جزیره: **۱۰ × جوابِ درست**، فقط بارِ اول. عددِ جوابِ درستی که اپ می‌فرستد به جواب‌های درستِ **ثبت‌شده** محدود می‌شود؛ صدا زدنِ مستقیمِ API امتیازِ مجانی نمی‌دهد.	`content/views.py` → `NodeQuizCompleteView`
ستاره	۹۰٪ به بالا ۳ · ۶۰٪ به بالا ۲ · بیشتر از صفر ۱	همان
سکه	**۵ × ستاره**، فقط بارِ اول	همان

سازوکار	قاعده	کجای کد
قلب	۵ قلب؛ هر جوابِ غلط یکی کم؛ هر **۵ ساعت** یکی برمی‌گردد و **۲۴ ساعت** بعد از اولین قلبِ ازدست‌رفته همه پر می‌شوند. با قلبِ صفر نه جوابی پذیرفته می‌شود نه پایانِ درس یا جزیره‌ی بی‌جواب (`403 no_hearts`). اشتراکِ فعال یا روزهای قلبِ بسته = قلبِ نامحدود.	`progress/models.py → UserHearts` · `content/views.py`
زنجیره (Streak)	جواب، پایانِ درس یا جزیره امروز را (به وقتِ تهران) روزِ فعال می‌کند. اگر آخرین روزِ فعال نه امروز باشد نه دیروز، صفر نشان داده می‌شود.	`progress/services.py`
جدولِ رتبه	هفتگی (امتیازِ ۷ روزِ اخیر) و همه‌ی زمان‌ها؛ زنده حساب و ۶۰ ثانیه کش می‌شود.	`progress/leaderboard.py`
پیشرفت روی نقشه	پایانِ آزمون، جزیره را با هر نمره‌ای تمام می‌کند و جزیره‌ی بعدی (`nextNodeId`) باز می‌شود؛ نمره فقط ستاره و امتیاز را تعیین می‌کند (تصمیمِ محصولی: ساده بماند).	`content/services.py`
اعلان	«جزیره را با ۳ ستاره تمام کردی و ۴۰ امتیاز گرفتی» — با رقمِ فارسی.	`notifications/services.py`

قلب واقعی است (`HEARTS_ENFORCED` پیش‌فرض روشن): قلبِ صفر بازی را متوقف می‌کند تا قلبِ بعدی برگردد. چکِ قلب داخلِ همان تراکنشی است که قلب را کم می‌کند، پس دو جوابِ هم‌زمان با یک قلبِ باقی‌مانده هر دو پذیرفته نمی‌شوند؛ و پایانِ آزمون بی‌جواب دادن هم بسته است. کسی که آخرین قلبش را روی آخرین سوال از دست داده، جزیره را تمام می‌کند. تست‌ها: `content/tests_hearts_gate.py` .

شکلِ ۱۱-۳ — صفحه‌ی «قلب‌هات تموم شد»: شمارشِ معکوسِ زنده تا قلبِ بعدی، قلبی که کم‌کم پر می‌شود، ساعتِ پر شدنِ همه، پیشنهادِ بسته‌ها و «تا اون موقع از پی بپرس» (پرسیدن قلب خرج نمی‌کند). وقتی وقتش رسید، صفحه خودش از سرور می‌پرسد و «ادامه بده» را نشان می‌دهد. فرانت: `src/components/NoHeartsScreen.tsx` .

۱۲. امنیت و احرازِ هویت

سه دور بازبینیِ بدبینانه روی کلِ بک‌اند انجام شد (مرداد، ۳ مهر و ۱۰ مهر)، و یک وارسیِ نهایی روی سایتِ زنده (۷ مهر) با این سؤال: «یک دانش‌آموزِ 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

1. کدِ پیامکی ۶ رقم، تصادفیِ رمزنگارانه؛ ۵ حدسِ غلط = باطل
2. access token user_id:زمان:امضا ۱ ساعت؛ در دیتابیس نیست
3. refresh token در جدولِ Session ؛ ۳۰ روز؛ قابلِ باطل کردن
4. خروج tokens_valid_after : هر توکنِ قدیمی‌تر رد می‌شود

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

۱۲-۲ آنچه رعایت شده

حوزه	چه کرده‌ایم	کجا
دسترسی به داده‌ی دیگران	شناسه‌ی کاربر همیشه از توکن خوانده می‌شود، هرگز از ورودی؛ هر کاربر فقط داده‌ی خودش را می‌بیند (گزارشِ والدین هم در حسابِ خودِ دانش‌آموز است).	همه‌ی `views.py` ها
سقفِ نرخ	روی همه‌ی نقاطِ حساس (ورود، دستیار، خرید، کدِ تخفیف، جواب…)؛ مهمان بر اساسِ IP واقعی (`NUM_PROXIES=2` : آروان + nginx).	`config/settings.py`
پیامک و هزینه‌اش	شماره به `09XXXXXXXXX` نرمال می‌شود؛ فاصله بینِ دو درخواست، ۱۰ در روز برای هر شماره، ۳۰۰ در روز برای کلِ سامانه (ضدِ SMS pumping) — شمارش در دیتابیس، نه حافظه‌ی موقت.	`accounts/phone.py` · `tests_otp_limits.py`
پرداخت	فقط تأییدِ سرور-به-سرور؛ برگشتِ جعلی اثری ندارد؛ webhook عمومیِ ناامن حذف شد؛ هر درگاهی بی‌کلید «بسته» می‌ماند (fail-closed)، نه موفقیتِ ساختگی. اعتبارِ بسته دفتری است و هر سفارش فقط یک بار اعتبار می‌دهد؛ وضعیتِ سفارش در ادمین فقط با action لغو/بازپرداخت عوض می‌شود.	`payment/` · `subscription/`
تقلبِ امتیاز	امتیاز فقط برای جوابِ ثبت‌شده؛ جایزه فقط بارِ اول.	`content/views.py`
درخواستِ تکراری و هم‌زمانی	Idempotency-Key در شش نقطه؛ `transaction.atomic` ؛ قیدِ یکتا در دیتابیس؛ SQLite در حالتِ `IMMEDIATE` تا قفل‌ها واقعاً قفل کنند.	`config/settings.py`
خطا و لاگ	هر خطا JSON یکدست بی‌جزئیاتِ داخلی؛ لاگ بدونِ بدنه، هدر، توکن یا رمز.	`config/exceptions.py` · `middleware.py`
محتوای مدل	HTML یا SVG تولیدِ مدل هرگز واردِ صفحه نمی‌شود؛ فقط JSON وارسی‌شده با فهرستِ سفید. دستیار فقط ریاضی (بخشِ ۹-۷).	`rag/services/diagram.py` · `guardrails.py`

حوزه	چه کرده‌ایم	کجا
حذفِ حساب	شماره‌دار با کدِ پیامکیِ تازه (همان سقفِ حدسِ ورود)، مهمان با تایپِ «حذف»؛ حسابِ نمایشی حذف نمی‌شود؛ سفارش‌ها برای حسابداری می‌مانند؛ توکنِ قبلی بعد از حذف کار نمی‌کند.	`accounts/views.py` → `DeleteAccountView`
مرورگر	CORS فقط برای دامنه‌ی فرانت؛ پاسخِ API با `no-store` در کشِ آروان نمی‌ماند؛ نامِ نمایشی از شماره و لینک پاک می‌شود؛ سقفِ حدسِ رمزِ ادمین برای هر IP.	`config/middleware.py`
کلیدها	همه در `.env` ِ سرور؛ هیچ کلیدی در git یا در پاسخِ API نیست؛ `/api/health` فقط «بله/خیر» می‌گوید.	`.env.example`

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

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

#	مورد	خطر	راهِ بستن
۱	کلیدِ امضای توکن روی سرور همان کلیدِ پیش‌فرضِ داخلِ مخزن است؛ هر کس مخزن را ببیند می‌تواند توکن جعل کند.	بحرانی	حذف از `.env` تا تنظیمات خودش کلیدِ تصادفی بسازد (کد آماده است)؛ همه یک بار دوباره وارد می‌شوند.
۲	۷ شماره‌ی دمو با کدِ ثابت روی سرورِ واقعی	بالا	پاک کردنِ `TEST_LOGIN_PHONES`
۳	داده و حساب‌های دمو و آزمایشی در دیتابیسِ واقعی	بالا	پاک‌سازی پس از بکاپ
۴	دسترسیِ سرور: حسابِ مشترک با رمز و ِsudo کامل	بالا	کلیدِ SSH و حسابِ محدود — کارِ ادمینِ سرور
۵	`/api/health`، Swagger و جدولِ رتبه بی‌ورود عمومی‌اند؛ پنلِ ادمین روی دامنه‌ی عمومی	متوسط	محدود کردن با ورود یا IP
۶	سرور بی‌آروان هم روی پورتِ ۸۰ جواب می‌دهد (امکانِ جعلِ IP برای دور زدنِ سقف)	متوسط	بستنِ پورت جز برای آروان — ادمینِ سرور
۷	کشِ مشترک (Redis) تنظیم نشده؛ سقف‌های نرخ برای هر worker جداست	متوسط	`CACHE_URL` روی ِRedis موجودِ سرور
۸	کدِ پیامکی و refresh token بی‌هش در دیتابیس؛ توکن در `localStorage` ِ مرورگر	کم	هش کردن؛ کوکیِ httpOnly
۹	آزمونِ جزیره قفل بودنِ جزیره را نمی‌سنجد (می‌شود جزیره‌ی جلوتر را با API تمام کرد)	کم	بررسیِ جزیره‌ی قبلی در `NodeQuizCompleteView`

جمع‌بندیِ وضعیتِ امنیت

لایه‌ی برنامه بازبینی و تست شده است: دسترسی، سقفِ نرخ، پرداخت، تکرارِ درخواست و تقلبِ امتیاز بسته‌اند. چیزهایی که باز مانده عمدی و مستند است — مهم‌ترینِ آن‌ها کلیدِ امضای سرور است که کدِ رفعش آماده است و اعمالش به زمانی موکول شده که خروجِ همه‌ی کاربرها از حساب مشکلی ایجاد نکند. بخشی از موارد نیز به زیرساخت مربوط است و بر عهده‌ی ادمینِ سرور است.

۱۳. استقرار (Deploy) و نگهداری

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

۱۳-۱ انتشارِ بک‌اند

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

1. ۱ · fetch از Gitea
2. ۲ · reset تغییرِ دستی ← patch در .deploy-backups/
3. ۳ · pip install
4. ۴ · migrate
5. ۵ · seed_content محتوا
6. ۶ · collectstatic
7. ۷ · restart + اثباتِ ری‌استارت
8. ۸ · health poll تا سالم

- **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:** زمانِ شروعِ سرویس را قبل و بعد مقایسه می‌کند. یک بار سرور دو روز کدِ قدیمی سرو کرد درحالی‌که همه‌ی فایل‌ها تازه بودند — از آن به بعد «فایل روی دیسک» را «منتشرشده» حساب نمی‌کنیم.
- ری‌استارت جداست چون `sudo` داخلِ اسکریپت نمی‌تواند رمز بپرسد.

۱۳-۲ انتشارِ فرانت

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

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

۱۳-۳ پیکربندی (`.env` ِ سرور)

متغیر	اگر نباشد
`PUBLIC_BASE_URL` · `FRONTEND_URL`	دامنه‌های فعلی. همه‌ی تنظیماتِ دامنه (CORS، callback، زرین‌پال، …) از همین دو ساخته می‌شوند؛ جابه‌جاییِ دامنه یعنی عوض کردنِ همین دو.
`DJANGO_SECRET_KEY`	یک کلیدِ تصادفی ساخته و در `.django_secret_key` نگه داشته می‌شود (حالتِ مطلوب).
`OPENAI_API_KEY` · `QDRANT_API_KEY`	دستیار جوابِ ثابتِ نمایشی می‌دهد.

متغیر	اگر نباشد
`KAVENEGAR_API_KEY` · `KAVENEGAR_OTP_TEMPLATE`	درخواستِ کد ۵۰۳ صریح می‌گیرد. الگو باید `SMSVerify` بماند.
`ZARINPAL_MERCHANT_ID` · `BAZAAR_*`	پرداخت بسته می‌ماند (۵۰۳).
`CACHE_URL`	کشِ جدا برای هر worker؛ `/api/health` هشدار می‌دهد.
`HEARTS_ENFORCED` · `TEST_LOGIN_PHONES`	روشن (قلب واقعی؛ `0` خاموشش می‌کند) · هیچ حسابِ آزمایشی.
`AI_DAILY_QUOTA_FREE`	۳ سوالِ رایگان در روز.
`DJANGO_DEBUG`	خاموش — و **هرگز روی سرور روشن نشود** (کدِ ورودِ همگانیِ `0000` را فعال می‌کند).

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

۱۳-۴ پایش، بکاپ و قواعدِ سرورِ مشترک

`GET /api/health`

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

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.

سرور مالِ ما نیست؛ ما یکی از مستأجرهاییم

- مالِ ما فقط `/home/administrator/mathapp-v2` و سرویسِ `zareian-mathapp` است. `gunicorn` ِ 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 حدودِ 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
۱۴ اپ با مرزِ 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؛ `tests_frontend_contract.py` تغییرِ ناخواسته را می‌گیرد.
۷۷۹ تستِ خودکار	تازه‌وارد بی‌ترس تغییر می‌دهد: اگر چیزی را شکست، تست قبل از سرور خبر می‌دهد.
قواعدِ یک‌جا	نرخ‌ها، دامنه‌ها و تنظیمات فقط در `config/settings.py` ؛ پیامِ خطا، لاگ و هویت یک مسیرِ مشترک دارند.
مستندات در مخزن	`docs/` + همین سند + Swagger + توضیحِ «چرا» کنارِ کدِ حساس.

۱۴-۲ دستورالعمل: افزودنِ یک اندپوینتِ تازه

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

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

وضعِ فعلی: ۲۴ درخواستِ هم‌زمان؛ درخواستِ عادی چند ده میلی‌ثانیه، جوابِ دستیار ده‌ها ثانیه. توکن stateless است، پس هر تعداد نسخه‌ی دیگر از بک‌اند پشتِ nginx بی‌تغییرِ کد کار می‌کند — به شرطِ دیتابیس و کشِ مشترک.

#	گلوگاه	راه‌حلِ ساده‌ی بعدی	نیاز
۱	SQLite فقط یک نویسنده دارد	Postgres (همین حالا روی سرور هست)؛ تغییرِ تنظیمات + انتقالِ داده؛ حالتِ `IMMEDIATE` برداشته می‌شود و قفلِ ردیف واقعی می‌شود.	ادمینِ سرور
۲	کش و سقفِ نرخِ جدا برای هر worker	`CACHE_URL` روی Redis موجود (db1 آزاد است).	یک خط در `.env`
۳	جوابِ دستیار نخ را نگه می‌دارد	صفِ کار (مثلاً Celery، که RabbitMQ و Redisاش روی سرور هست) یا ویوی async برای `/api/ai/ask` .	کد
۴	یک سرور برای همه	چند نسخه‌ی gunicorn پشتِ nginx؛ فرانت از CDN آروان.	زیرساخت
۵	هزینه‌ی دستیار با کاربرِ زیاد	کش، سقفِ روزانه (هست)، و سنجشِ مدلِ ارزان‌تر روی همان ابزارِ `rag_model_benchmark` .	سنجش

بهینه‌سازی‌های انجام‌شده: رفعِ N+1 در صفحه‌ی اصلی (از N+۱ کوئری به ۳ کوئریِ ثابت)، کشِ نقشه، کشِ جدولِ رتبه (۶۰ ثانیه)، کشِ جوابِ دستیار، ادغامِ چند درخواست در یکی (`/api/home` ، `/api/user/summary` ، `/api/nodes/{id}/quiz`)، و حجمِ بارِ اولِ فرانت از ۵۰۵ به ۸۵ کیلوبایت.

۱۵. تست و کیفیت

چه	چطور
تستِ خودکارِ بک‌اند	**۷۷۹ تست**، همه سبز (`python manage.py test`). روی فایلِ دیتابیسِ واقعی (`.test-db.sqlite3`) اجرا می‌شوند تا رفتارِ قفل‌ها مثلِ سرور باشد. هر باگِ پیداشده در بازبینی‌ها یک تستِ اختصاصی دارد (`tests_audit_fixes.py` ، `tests_otp_limits.py` ، `tests_hearts_gate.py` ، …). هم‌زمانی با دو thread واقعی آزموده می‌شود (اعتبارِ دستیار، آخرین قلب).
قرارداد با فرانت	`content/tests_frontend_contract.py` و build با TypeScript ِstrict در فرانت.
تستِ روی سایتِ زنده	هر صفحه با حساب‌های نمایشی روی سرورِ واقعی؛ مرورگرِ خودکار (puppeteer) برای وارسیِ صفحه‌ها.
کیفیتِ دستیار	`rag_model_benchmark` (مسئله‌های واقعیِ کتاب از مسیرِ واقعی، با هزینه‌ی واقعی)، `ai_feedback_report` (👍/👎 به تفکیکِ نسخه‌ی پرامپت)، وارسیِ چشمیِ نمونه پیش از اعتماد به هر اجرای انبوه.
بی‌هزینه آزمودن	`tools/fake_openai.py` جای GapGPT می‌نشیند.
پایش	لاگِ JSON هر درخواست + `/api/health` ؛ محلی با Grafana/Loki (`docs/docker.md`).

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 (۱۰ مهر ۱۴۰۵، روی سرورِ زنده)

بخش	وضعیت	توضیح
اپ و API	زنده	هر دو روی آخرین نسخه؛ همه‌ی صفحه‌ها با حساب‌های نمایشی وارسی شد
محتوا	کامل	۸ پایه؛ هیچ جزیره‌ای زیرِ ۴ سؤال نیست
پیامکِ ورود	کار می‌کند	کدِ واقعی به گوشی رسید (الگوی SMSVerify)؛ هر پیامک هزینه دارد
دستیار + شکل	کار می‌کند	کُند (~۳۰ ثانیه، از سمتِ GapGPT)؛ ~یک شکل از پنج ایراد دارد
زرین‌پال	نیمه	درخواستِ پرداخت پذیرفته می‌شود؛ پرداختِ واقعی تا انتها آزموده نشده
کافه‌بازار	منتظر	اتصالِ API برقرار؛ منتظرِ تعریفِ بسته‌ها در پنلِ بازار (محصولِ مصرفی) و اپِ اندرویدی
بسته‌ی سوال و قلب	زنده	سه بسته روی سایت؛ قلبِ صفر بازی را متوقف می‌کند
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	حساب‌های آزمایشیِ شماره‌دار ۱۰ مهر پاک شدند (با بکاپ)؛ ۷ شماره‌ی نمایشی و حساب‌هایشان عمداً مانده‌اند.
قیمتِ بسته‌ها و سهمیه‌ی رایگان	بعد از دو هفته مصرفِ واقعی بازبینی شود؛ سود در `docs/ai-cost-and-guardrails.md` حساب شده.
سیاستِ بازپرداخت	الان فقط اعتبارِ خرج‌نشده پس گرفته می‌شود؛ پول از پنلِ درگاه.
Redis و Postgres	هماهنگی با ادمینِ سرور؛ هر دو روی همان ماشین موجودند.
پورتِ ۸۰ و دسترسیِ SSH	کارِ ادمینِ سرور.
پیشرفت با هر نمره	در حالِ 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 واقعیِ کامل با زرین‌پال.
4. `CACHE_URL` (Redis) و بعد Postgres.

5. صفِ کار برای دستیار؛ سنجشِ شکل روی نمونه‌ی بزرگ‌تر.
6. تعریفِ سه بسته در پنلِ کافه‌بازار و اپِ اندرویدی با Poolakey.

پیوست

الف · واژه‌نامه

واژه	معنی در این پروژه
API / اندپوینت	نشانی‌ای که اپ برای گرفتن یا فرستادنِ داده صدا می‌زند، مثلِ `GET /api/home` .
اپِ جنگو	یک پوشه‌ی مستقل با مدل، ویو و تستِ خودش؛ یک مسئولیت.
مدل / migration	تعریفِ یک جدول در پایتون / اسکریپتی که تغییرِ جدول را روی دیتابیس اعمال می‌کند.
ویو / serializer	کدی که درخواست را جواب می‌دهد / کدی که شکلِ ورودی و خروجی را تعریف و وارسی می‌کند.
middleware	لایه‌ای که همه‌ی درخواست‌ها پیش و پس از ویو از آن رد می‌شوند.
توکن	«کارتِ شناسایی» امضاشده که اپ با هر درخواست می‌فرستد.
throttle	سقفِ تعدادِ درخواست در زمان.
Idempotency-Key	کلیدی که باعث می‌شود تکرارِ یک درخواست دو بار اثر نکند.
تراکنشِ اتمیک	چند نوشتن که یا همه انجام می‌شوند یا هیچ‌کدام.
RAG	اول پیدا کردنِ متنِ مرتبط از کتاب، بعد نوشتنِ جواب بر پایه‌ی آن.
embedding / بردار	تبدیلِ متن به عددهایی که معنا را نشان می‌دهند تا متن‌های هم‌معنا کنارِ هم بیفتند.
chunk / collection	یک تکه‌ی کتاب (یک تیپ‌مسئله) / همه‌ی تکه‌های یک کتاب در Qdrant.
SSE (استریم)	فرستادنِ جواب تکه‌تکه، تا کاربر پیش از تمام شدن چیزی ببیند.
seed	پر کردنِ دیتابیس از فایل‌های محتوای داخلِ 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           # کلِ تست‌ها
python manage.py spectacular --file schema.yaml   # قراردادِ API

# محتوا و دستیار
python manage.py seed_content --export --only questions
python manage.py rag_catalog                      # collectionهای Qdrant (فقط خواندن)
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 مستنداتِ موجود

سند	برای
`README.md` (هر دو مخزن)	درِ ورودی و راه‌اندازی
`docs/architecture.md`	تصمیم‌های معماری و دلیلشان، با جزئیات
`docs/operations.md`	انتشار، متغیرها، بکاپ، جابه‌جاییِ دامنه، ورودِ تیم
`docs/security-before-production.md`	۱۸ موردِ امنیتیِ باز با خطر و راهِ بستن
`docs/content-authoring.md`	ویرایشِ محتوا و ساختِ انبوهِ سؤال
`docs/extraction-playbook.md`	خواندنِ کتاب با مدلِ بینایی: هزینه و درس‌ها
`docs/ai-cost-and-guardrails.md`	هزینه‌ی هر سوالِ دستیار، بسته‌ها و قاعده‌ی خرج، محافظِ «فقط ریاضی»
`docs/model-prices.md`	دفترِ قیمتِ اندازه‌گیری‌شده‌ی مدل‌ها، با تاریخ؛ اگر قیمت عوض شد ردیف اضافه شود
`data/textbooks/README.md`	افزودنِ کتابِ تازه
`/api/docs/`	قراردادِ زنده و قابلِ امتحانِ API

تدریسینو · مستندِ فنی · علیرضا زارعیان · مهرِ ۱۴۰۵

عددها از کد، اسکیمای API، دیتابیس و سنجش‌های ثبت‌شده در مخزن استخراج شده‌اند.