هفت‌خان توسعه بک اند
خان دوم و خان سوم: عمق و استحکام


رضا قمی‌الاصل
دانشجوی مهندسی کامپیوتر
دانشکدگان فارابی دانشگاه تهران
rezaqomyalasl@gmail.com

۱۸ دقیقه

توی قسمت قبلی، با مفاهیم پایه‌ی بک‌اند و ابزارهایی آشنا شدیم که برای شروع کار بهشون نیاز داریم. یکی از اون‌ها دیتابیس بود. بیشتر هم سراغ دیتابیس‌های رابطه‌ای رفتیم، SQL رو شناختیم و دیدیم ORM[1] و Migration چه کمکی به ما می‌کنن.

ولی بلد بودن SQL تازه شروع کار با داده‌ست.

وقتی پروژه بزرگ‌تر می‌شه، دیگه فقط مهم نیست که کوئری[2] ما جواب درست برگردونه. باید بدونیم اطلاعات رو چطور طراحی کنیم، دیتابیس کوئری‌ها رو چطور اجرا می‌کنه و وقتی چند درخواست هم‌زمان روی یه داده کار می‌کنن چه اتفاقی می‌افته.

از طرف دیگه، همه‌ی کارهایی که با داده انجام می‌دیم شبیه هم نیستن. ثبت یه سفارش، جستجو بین چند میلیون محصول و گرفتن گزارش فروش چند سال گذشته، سه مسئله متفاوتن و لزوماً یه ابزار بهترین انتخاب برای هر سه نیست.

توی خان دوم می‌خوایم نقشه‌ی این بخش از مسیر رو کامل‌تر کنیم. بعد از اون هم می‌ریم سراغ امنیت؛ جایی که باید یاد بگیریم چه کسی داره با سیستم کار می‌کنه، چه کاری اجازه داره انجام بده و چقدر می‌تونیم به چیزی که از بیرون وارد برنامه می‌شه اعتماد کنیم.

خان دوم: غواصی در اقیانوس داده‌ها

پایگاه‌های داده، NoSQL و Caching

یک قدم عمیق‌تر در دیتابیس‌های رابطه‌ای

توی قسمت قبلی با دیتابیس‌های رابطه‌ای مثل PostgreSQL و MySQL آشنا شدیم. برای شروع، ساخت جدول و یاد گرفتن SQL کار ما رو راه می‌اندازه؛ ولی اگه بخواید جدی‌تر وارد بک‌اند بشید، باید کم‌کم از نوشتن کوئری فاصله بگیرید و یه مقدار از رفتار خود دیتابیس هم سر دربیارید.

طراحی داده

یکی از اولین چیزهایی که باید جدی‌تر یاد بگیرید مدل‌سازی داده[3] هست.

فرض کنید یه فروشگاه داریم و قیمت یه محصول امروز ۵۰۰ هزار تومنه. یه کاربر محصول رو می‌خره و چند ماه بعد قیمتش به ۸۰۰ هزار تومان می‌رسه. سفارش قدیمی کاربر هنوز باید همون قیمت زمان خرید رو نشون بده.

همین مثال ساده نشون می‌ده طراحی داده فقط این نیست که برای User و Product و Order چندتا جدول درست کنیم. باید بدونیم کدوم اطلاعات تغییر می‌کنن، چه چیزهایی باید سابقه‌ی خودشون رو حفظ کنن و رابطه‌ی بین قسمت‌های مختلف سیستم چیه.

اینجا رابطه‌های یک‌به‌یک[4]، یک‌به‌چند[5] و چندبه‌چند[6] رو بیشتر می‌بینید و بعد به Normalization و Denormalization می‌رسید. مهم‌تر اینه که بفهمید چرا بعضی اطلاعات رو از هم جدا نگه می‌داریم و چرا بعضی وقت‌ها هم عمداً بخشی از داده رو تکرار می‌کنیم.

برای عمیق‌تر شدن توی این بخش، Cardinality، Schema Design، Normalization و Data Lifecycle موضوعات خوبی هستن.

وقتی تعداد داده‌ها زیاد می‌شه

یه کوئری ممکنه روی ده هزار رکورد خیلی سریع باشه، ولی روی چند میلیون رکورد دیگه اون عملکرد قبلی رو نداشته باشه.

اینجا یکی از اولین موضوعاتی که باید سراغش برید ایندکس‌گذاری[7] هست. ایندکس می‌تونه پیدا کردن بعضی اطلاعات رو سریع‌تر کنه، ولی خودش هم هزینه داره و قرار نیست روی هر ستونی Index بسازیم.

بعد از اون باید با کوئری پلن[8] آشنا بشید. دیتابیس برای اجرای یه کوئری تصمیم می‌گیره از چه مسیری به داده برسه و ابزارهایی مثل EXPLAIN و EXPLAIN ANALYZE کمک می‌کنن این مسیر رو ببینیم.

۱. Object–Relational Mapping
۲. Query
۳. Data Modeling
۴. One-to-One
۵. One-to-Many
۶. Many-to-Many
۷. Indexing
۸. Query Plan

لازم نیست وارد جزئیات موتور PostgreSQL بشید. چیزی که مهمه اینه که وقتی یه کوئری کند شد، بدونید بررسی رو از کجا شروع کنید. B-Tree، Composite Index و Query Planner ادامه‌ی طبیعی این بخش هستن.

وقتی چند درخواست با یک داده کار می‌کنن

یه مسئله مهم دیگه زمانی پیش میاد که چند تغییر به هم وابسته باشن یا چند درخواست[۹] هم‌زمان روی یه داده کار کنن.

مثلاً توی انتقال پول، کم شدن موجودی یه حساب و زیاد شدن موجودی حساب دیگه باید با هم انجام بشن. یا ممکنه دو نفر تقریباً در یه لحظه بخوان آخرین موجودی یه محصول رو بخرن.

اینجاست که تراکنش[۱۰] و هم‌زمانی[۱۱] اهمیت پیدا می‌کنن.

برای شروع، ACID[۱۲] رو بخونید و بعد سراغ Lock، Isolation Level، Deadlock و MVCC[۱۳] برید. لازم نیست این مفاهیم رو همین‌جا کامل یاد بگیرید، ولی اگه با پرداخت، موجودی یا هر داده‌ی حساسی کار می‌کنید، این بخش رو جدی بگیرید.

همه استفاده‌ها از داده شبیه هم نیستن

OLTP[۱۴] و OLAP[۱۵]

فرض کنید یه فروشگاه چند سال فعالیت کرده و میلیون‌ها سفارش داخل سیستم داره.

دیتابیس اصلی هر لحظه درگیر ثبت سفارش، پرداخت و تغییر موجودیه. در کنارش، تیم کسب‌وکار هم می‌خواد فروش چند سال گذشته رو تحلیل کنه.

هر دو کار با داده سروکار دارن، ولی جنس کارشون یکی نیست.

عملیات‌هایی مثل ثبت سفارش و پرداخت معمولاً در دسته‌ی OLTP قرار می‌گیرن. اینجا تعداد زیادی عملیات نسبتاً کوچک داریم که باید سریع و درست انجام بشن.

در مقابل، وقتی حجم زیادی از داده‌های قبلی رو برای گزارش و تحلیل می‌خونیم، وارد فضای OLAP می‌شیم.

برای پروژه‌های کوچیک ممکنه همون PostgreSQL هر دو کار رو انجام بده. ولی با زیاد شدن حجم اطلاعات، کوئری‌های تحلیلی سنگین می‌تونن به دیتابیس اصلی فشار بیارن. اینجاست که ابزارهایی مثل ClickHouse رو می‌بینید.

۹. Request
۱۰. Transaction
۱۱. Concurrency
۱۲. Atomicity, Consistency, Isolation, Durability
۱۳. Multi-Version Concurrency Control
۱۴. Online Transaction Processing
۱۵. Online Analytical Processing

اگه خواستید این مسیر رو ادامه بدید، Column-Oriented Database، Data Warehouse و ETL/ELT اصطلاح‌های مهم بعدی هستن.

یه نگاه به دنیای بزرگ‌تر دیتابیس‌ها

دیتابیس رابطه‌ای تنها مدل موجود نیست.

بسته به نوع اطلاعات و کاری که می‌خوایم باهاش انجام بدیم، خانواده‌های مختلفی از دیتابیس‌ها وجود دارن. بخشی از اون‌ها رو معمولاً زیر عنوان NoSQL می‌بینید.

دیتابیس‌های سندمحوری[۱۶] مثل مونگودی‌بی[۱۷]، Key-Value Storeها، دیتابیس‌های ستون‌گسترده[۱۸] مثل Cassandra و ScyllaDB و دیتابیس‌های گرافی[۱۹] چند نمونه از این خانواده‌ها هستن. در کنار این‌ها، دیتابیس‌های سری زمانی[۲۰]، موتورهای جستجو و دیتابیس‌های تحلیلی هم وجود دارن.

قرار نیست همه این‌ها رو یاد بگیرید. هدف اینه که بدونید PostgreSQL جواب همه مسئله‌ها نیست و از اون طرف هم هر ابزار جدیدی ارزش وارد شدن به معماری پروژه رو نداره.

مونگودی‌بی و دیتابیس‌های سندمحور

مونگودی‌بی داده رو به شکل سندهایی[۲۱] شبیه JSON نگه می‌داره و برای بعضی مدل‌های داده انعطاف بیشتری می‌ده.

اگه وارد این مسیر شدید، فقط دستورات مونگو رو یاد نگیرید. Embedding و Referencing و نحوه‌ی طراحی Schema توی دیتابیس‌های سندمحور مهم‌ترن.

انعطاف‌پذیری[۲۲] ساختار هم به معنی بی‌نیازی از طراحی داده نیست. اگه هر سند یه شکل داشته باشه، فقط آشفتگی رو از دیتابیس به کد منتقل کردید.

۱۶. Document Database
۱۷. MongoDB
۱۸. Wide-Column Database
۱۹. Graph Database
۲۰. Time-Series Database
۲۱. Document
۲۲. Flexibility

Cassandra و دیتابیس‌های توزیع‌شده

در سیستم‌هایی که حجم داده و میزان نوشتن خیلی بالاست[۲۳] و اطلاعات روی چند سرور پخش شدن، ممکنه اسم‌هایی مثل Cassandra یا ScyllaDB رو ببینید.

اینجا مدل‌کردن داده با PostgreSQL فرق داره و کوئری‌هایی که قراره اجرا بشن می‌تونن روی شکل ذخیره شدن اطلاعات اثر مستقیم داشته باشن.

اگه یه روز وارد چنین پروژه‌ای شدید، Replication، Partition Key و Consistency Level موضوعات مهم بعدی هستن. بخش عمیق‌تر این داستان به Distributed Systems می‌رسه که جلوتر دوباره سراغش می‌ریم.

جستجو هم مسئله‌ی خودش رو داره

Elasticsearch و OpenSearch

فرض کنید چند میلیون محصول داریم. کاربر چیزی رو جستجو می‌کنه، ولی شاید اسم دقیق محصول رو ندونه یا یه کلمه رو اشتباه تایپ کرده باشه. در عین حال انتظار داره نتیجه‌های مرتبط‌تر بالاتر نمایش داده بشن.

با SQL می‌شه سیستم جستجو ساخت، اما وقتی جستجو بخش مهمی از محصول می‌شه، ابزارهایی مثل Elasticsearch و OpenSearch امکانات بیشتری در اختیارمون می‌ذارن.

برای شروع، Full-Text Search و Inverted Index رو بشناسید. بعد از اون Analyzer، Tokenization و Relevance وارد مسیر می‌شن. اگه بیشتر جلو برید، Vector Search، BM25 و Hybrid Search هم موضوعات مهمی هستن.

معمولاً اطلاعات اصلی همچنان داخل دیتابیس اصلی می‌مونن و موتور جستجو نسخه‌ای مناسب برای جستجو از اون‌ها رو نگه می‌داره. همین‌جا یه مسئله‌ی جدید هم به‌وجود میاد: هماهنگ نگه‌داشتن این دو نسخه از داده.

۲۳. Write

Caching؛ وقتی لازم نیست هر بار از اول شروع کنیم

بعضی اطلاعات خیلی زیاد خونده می‌شن، ولی زیاد تغییر نمی‌کنن. اگه برای هر درخواست دوباره همون کوئری رو اجرا کنیم، داریم یه کار تکراری رو بارها انجام می‌دیم.

اینجاست که کَش به کار میاد.

در نگاه اول کَش ساده‌ست: نتیجه‌ای رو برای مدتی نگه می‌داریم تا دفعه بعد سریع‌تر بهش برسیم. بخش سخت وقتی شروع می‌شه که داده‌ی اصلی تغییر کنه و نسخه‌ی قدیمی هنوز داخل کَش مونده باشه.

برای همین باید با Cache Invalidation، Cache-Aside، Cache Hit/Miss و TTL[۲۴] آشنا بشید. بعدتر Read-Through، Write-Through و موضوعاتی مثل Cache Stampede و Eviction Policy رو هم دنبال کنید.

یکی از ابزارهای معروف این بخش ردیس[۲۵] هست. ردیس فقط برای کَش استفاده نمی‌شه و ممکنه توی Counter، Rate Limiting، Session و چند مسئله دیگه هم ببینیدش.

مثل همیشه، هدف این نیست که دستورات ردیس رو حفظ کنید؛ باید بفهمید چه داده‌ای رو چرا داخل ردیس می‌ذارید.

چند موضوعی که نباید از نقشه حذف بشن

کار با دیتابیس فقط به کوئری و انتخاب نوع دیتابیس ختم نمی‌شه.

Connection Pooling رو بشناسید و بفهمید چرا برنامه برای هر درخواست یه کانکشن تازه باز نمی‌کنه.

پشتیبان‌گیری[۲۶] و بازیابی[۲۷] رو جدی بگیرید. بک‌آپی که هیچ‌وقت بازیابی‌کردنش رو امتحان نکردید، خیلی قابل‌اعتماد نیست.

بعدتر هم با Partitioning، Replication و Sharding آشنا بشید. جزئیات Scaling دیتابیس رو برای خان‌های بعدی می‌ذاریم، ولی اسم و کاربرد کلی این مفاهیم باید توی نقشه‌تون باشه.

پایان خان دوم

هدف این خان یاد گرفتن چند دیتابیس جدید نبود.

بعد از SQL، باید مدل‌سازی داده رو جدی‌تر یاد بگیرید، ایندکس و کوئری پلن رو بشناسید و تراکنش و همزمانی رو بفهمید.

بعد از اون، خانواده‌های مختلف دیتابیس، Analytics، Search و Caching کم‌کم وارد مسیرتون می‌شن.

۲۴. Time to Live
۲۵. Redis
۲۶. Backup
۲۷. Restore

لازم نیست همه این‌ها رو با هم یاد بگیرید. اینجا فقط نقشه رو جلوی شما گذاشتیم تا وقتی با هرکدوم توی یه پروژه روبه‌رو شدید، بدونید از کجا باید شروع کنید.

خان سوم: نگهبانان و قفل‌های آهنین

امنیت، احراز هویت و طراحی APIهای استاندارد

توی خان صفر گفتیم یکی از وظایف مهم بک‌اند اینه که مطمئن بشه هر کسی فقط به اطلاعاتی دسترسی داره که اجازه‌ش رو داره.

حالا می‌خوایم یه مقدار بیشتر وارد همین موضوع بشیم.

امنیت بک‌اند فقط لاگین کردن کاربر یا گذاشتن رمز عبور روی یه صفحه نیست. تقریباً هر چیزی که از بیرون وارد برنامه می‌شه، می‌تونه روی امنیت سیستم اثر بذاره.

کاربر اطلاعات می‌فرسته، فایل آپلود می‌کنه، شناسه یه سفارش رو توی URL قرار می‌ده یا یه توکن همراه درخواست می‌فرسته.

نکته‌ی مهم اینه که بک‌اند نباید فرض کنه چون فرانت‌اند خودش این درخواست رو ساخته، پس حتماً همه‌چیز درسته. هر کسی می‌تونه مستقیماً با API ما ارتباط برقرار کنه و درخواست خودش رو بسازه.

برای همین یکی از عادت‌های مهم یه توسعه‌دهنده‌ی بک‌اند اینه که به اطلاعاتی که از کلاینت میاد، قبل از بررسی اعتماد نکنه.

اول باید بدونیم با چه کسی طرفیم

Authentication و Authorization

اولین چیزی که باید توی امنیت بک‌اند خوب از هم جدا کنید، Authentication و Authorization هست.

Authentication یعنی بفهمیم کاربر واقعاً کیه.

Authorization یه سؤال متفاوت داره: حالا که فهمیدیم این کاربر کیه، اجازه داره چه کاری انجام بده؟

مثلاً ممکنه کاربر وارد شده باشه و سفارش خودش رو از آدرس زیر ببینه:

/orders/120

ولی اگه شماره سفارش رو به ۱۲۱ تغییر داد، بک‌اند باید بررسی کنه که این سفارش هم متعلق به همون کاربر هست یا نه.

پس لاگین بودن کاربر به این معنی نیست که اجازه دسترسی به هر چیزی رو داره.

این تفاوت ساده، پایه‌ی بخش بزرگی از امنیت بک‌انده.

بعد از اینکه این دو مفهوم رو فهمیدید، باید سراغ راه‌هایی برید که هویت کاربر بین درخواست‌های مختلف نگه داشته می‌شه.

اینجا معمولاً با سشن (Session)، کوکی (Cookie) و JWT[۲۸] روبه‌رو می‌شید.

در سشن معمولاً یه شناسه داخل کوکی نگه داشته می‌شه و اطلاعات مربوط به سشن سمت سرور قرار داره. اگه از کوکی استفاده می‌کنید، تنظیماتی مثل Secure، HttpOnly و SameSite رو هم بخونید؛ چون روی امنیت اون تأثیر دارن.

JWT روش دیگه‌ایه که مخصوصاً توی APIها زیاد می‌بینید.

ولی JWT رو فقط به ساختن یه توکن و اعتبارسنجی‌کردنش[۲۹] محدود نکنید. باید بدونید اطلاعات داخل JWT معمولاً قابل خوندنه و امضاشدن توکن به معنی رمزشدنش نیست.

بعدتر هم می‌تونید Refresh Token، Access Token، Expiration و Token Revocation رو دنبال کنید.

هدف این نیست که بین سشن و JWT یکی رو به‌عنوان روش «بهتر» انتخاب کنید. باید بفهمید هرکدوم چطور کار می‌کنن و چه دردسرهایی دارن.

ورود با Google و سرویس‌های دیگه

بعد از Authentication معمولی، احتمالاً یه جایی با «ورود با Google» یا «ورود با GitHub» روبه‌رو می‌شید.

اینجا اسم OAuth 2.0 و OpenID Connect رو زیاد می‌شنوید.

لازم نیست اول کار تمام جریان‌های OAuth[۳۰] رو حفظ کنید.

فعلاً همین رو بدونید که قرار نیست سایت مقصد رمز حساب Google شما رو بگیره. کاربر به یه سرویس اجازه می‌ده تا به‌شکل مشخصی از اطلاعات یا هویت اون استفاده کنه.

۲۸. JSON Web Token
۲۹. Verify
۳۰. Flow

وقتی خواستید این بخش رو عمیق‌تر یاد بگیرید، تفاوت OAuth و OpenID Connect رو بخونید و بعد سراغ Access Token، Scope، Authorization Code Flow و Refresh Token برید.

رمز عبور رو چطور نگه داریم؟

یه قسمت مهم دیگه از امنیت، نگهداری اطلاعات حساسه.

واضح‌ترین نمونه هم رمز کاربره.

رمز نباید به‌شکل متن ساده داخل دیتابیس ذخیره بشه. حتی استفاده از هر هشی[۳۱] هم برای این کار مناسب نیست.

برای هش‌کردن رمز عبور الگوریتم‌هایی مثل Argon2 و bcrypt وجود دارن که مخصوص همین کار طراحی شدن.

توی همین مسیر با مفاهیمی مثل Salt هم آشنا می‌شید.

یه تفاوت مهم دیگه هم هست که باید از همون اول درست یاد بگیرید: Hashing، Encoding و Encryption یه چیز نیستن.

Encoding فقط شکل نمایش اطلاعات رو عوض می‌کنه. مثلاً Base64 امنیت خاصی به داده اضافه نمی‌کنه.

Hashing معمولاً یه مسیر یک‌طرفه‌ست و رمز عبور یکی از جاهاییه که ازش استفاده می‌کنیم.

Encryption با یک کلید[۳۲] انجام می‌شه و اطلاعات در صورت داشتن کلید مناسب قابل بازیابی هستن.

برای یه توسعه‌دهنده بک‌اند همین درک پایه خیلی مهم‌تر از اینه که وارد جزئیات الگوریتم‌های رمزنگاری بشه. بعداً که بهش نیاز پیدا کردید، می‌تونید کلید عمومی و خصوصی[۳۳]، امضای دیجیتال[۳۴]، گواهی[۳۵] و TLS رو عمیق‌تر بخونید.

۳۱. Hash
۳۲. Key
۳۳. Public/Private Key
۳۴. Digital Signature
۳۵. Certificate

چیزی که کاربر می‌فرسته، لزوماً همون چیزیه که انتظار داریم

حالا برگردیم به درخواست‌هایی که وارد بک‌اند می‌شن.

فرض کنید یه Endpoint انتظار داره یه عدد به‌عنوان user_id بگیره.

فرانت‌اند شما ممکنه همیشه عدد درست بفرسته، ولی هیچ چیزی جلوی یه نفر رو نمی‌گیره که خودش درخواست بسازه و مقدار دیگه‌ای ارسال کنه.

برای همین Input Validation یکی از پایه‌های کار بک‌انده.

نوع، اندازه و ساختار اطلاعاتی که وارد سیستم می‌شن باید بررسی بشن.

اما موضوع فقط Validation نیست.

گاهی ورودی کاربر وارد قسمت حساسی از برنامه می‌شه.

مثلاً اگه ورودی کاربر رو مستقیم وارد یه کوئری SQL کنیم، ممکنه زمینه‌ی SQL Injection رو فراهم کنیم. برای همین کوئری‌ها باید به‌شکل درست و با پارامترها ساخته بشن.

همین الگو جاهای دیگه هم وجود داره.

ورودی‌ای که به‌عنوان دستور به سیستم‌عامل داده می‌شه می‌تونه Command Injection ایجاد کنه. دستکاری مسیر فایل ممکنه به Path Traversal برسه و Request زدن بک‌اند به آدرسی که کاربر کنترلش می‌کنه می‌تونه زمینه‌ی SSRF[۳۶] رو ایجاد کنه.

لازم نیست اسم تمام حمله‌ها رو حفظ کنید.

چیزی که باید توی ذهنتون بمونه این سؤاله:

این داده از کجا اومده و قراره کجا استفاده بشه؟

اگه این سؤال تبدیل به عادت بشه، بعداً فهمیدن خیلی از مسائل امنیتی راحت‌تر می‌شه.

برای عمیق‌تر شدن توی این بخش هم OWASP Top 10 و OWASP API Security Top 10 دو نقطه‌ی شروع خیلی خوب هستن.

وقتی Client مرورگره

بخشی از امنیت وقتی مهم‌تر می‌شه که بک‌اند ما با مرورگر کار می‌کنه.

سه اسمی که اینجا زیاد می‌بینید CORS[۳۷]، CSRF[۳۸] و XSS[۳۹] هستن.

لازم نیست توی این مقاله وارد جزئیاتشون بشیم، ولی حداقل باید بدونید هرکدوم تقریباً مربوط به چه مسئله‌ایه.

۳۶. Server-Side Request Forgery
۳۷. Cross-Origin Resource Sharing
۳۸. Cross-Site Request Forgery
۳۹. Cross-Site Scripting

CORS مشخص می‌کنه کد داخل مرورگر تحت چه شرایطی می‌تونه به یه Origin دیگه درخواست بده. پس CORS جای Authentication یا Authorization رو نمی‌گیره.

CSRF بیشتر وقتی اهمیت پیدا می‌کنه که مرورگر اطلاعات احراز هویت، مثل کوکی، رو به شکل خودکار همراه درخواست می‌فرسته.

XSS هم بیشتر توی مرورگر خودش رو نشون می‌ده، ولی بک‌اند می‌تونه با ذخیره و برگردوندن محتوای ناامن، بخشی از این مشکل باشه.

پس حتی اگه فقط بک‌اند کار می‌کنید، لازمه رفتار امنیتی مرورگر رو هم در حد خوبی بشناسید.

چند عادت امنیتی مهم

در کنار این مفاهیم، چند چیز هم هست که بهتره از همون اول تبدیل به عادت بشن.

API نباید اجازه بده یه کلاینت بدون محدودیت درخواست بفرسته یا منابع سیستم رو مصرف کنه. برای همین Rate Limiting و محدود کردن چیزهایی مثل حجم آپلود یا تعداد آیتم‌های یه درخواست اهمیت دارن.

اطلاعاتی مثل رمز دیتابیس، کلید API، توکن و کلید خصوصی هم نباید مستقیم داخل Source Code یا Git قرار بگیرن. این‌ها Secret هستن و باید جدا از کد نگهداری بشن.

سطح دسترسی هم تا جای ممکن باید محدود باشه. اگه یه سرویس فقط نیاز داره از دیتابیس بخونه، دلیل خاصی نداره دسترسی حذف اطلاعات هم داشته باشه.

همین چند عادت ساده جلوی تعداد زیادی از اشتباه‌های امنیتی رو می‌گیرن.

API خوب فقط APIای نیست که جواب بده

توی قسمت قبلی با متدهای HTTP و کدهای وضعیت[۴۰] آشنا شدیم. حالا باید کم‌کم اون‌ها رو توی طراحی API هم درست استفاده کنیم.

مسیرهای API باید قابل‌فهم و قابل‌پیش‌بینی باشن.

خطاها بهتره ساختار مشخصی داشته باشن.

۴۰. Status Codes

برای لیست‌های بزرگ، صفحه‌بندی[۴۱] داشته باشیم و اگه فیلتر یا مرتب‌سازی داریم، شکل استفاده از اون‌ها توی Endpointهای مختلف بی‌دلیل عوض نشه.

یه مفهوم مهم دیگه هم خنثی‌بودن[۴۲] هست.

مثلاً اگه کلاینت به خاطر قطع شدن اینترنت ندونه درخواست پرداختش انجام شده یا نه و دوباره همون درخواست رو بفرسته، نباید ناخواسته دوبار پرداخت انجام بشه.

این نوع مسائل نشون می‌دن طراحی API فقط انتخاب اسمِ اندپوینت نیست.

بعدتر می‌تونید نسخه‌گذاری API رو هم دنبال کنید و با OpenAPI و Swagger قرارداد API رو مستند نگه دارید.

پایان خان سوم

قرار نیست با خوندن این خان متخصص امنیت بشید.

چیزی که باید ازش با خودتون ببرید چند اصل ساده‌ست:

باید بدونید کاربر کیه و چه اجازه‌ای داره.

نباید به ورودی کلاینت اعتماد کنید.

رمزها و اطلاعات حساس باید درست نگهداری بشن.

و امنیت چیزی نیست که بعد از تمام شدن پروژه بهش اضافه کنیم.

وقتی وارد پروژه‌های واقعی‌تر شدید، هرکدوم از این بخش‌ها دنیای بزرگ‌تری جلوتون باز می‌کنن. OWASP هم می‌تونه نقشه‌ی خوبی برای ادامه‌ی این مسیر باشه.

توی قسمت بعدی می‌ریم سراغ ساختار خود نرم‌افزار؛ جایی که با بزرگ‌تر شدن پروژه، معماری، اصول SOLID و Design Patternها اهمیت بیشتری پیدا می‌کنن.

۴۱. Pagination

۴۲. Idempotency