System Architecture Document

1. هدف سند معماری سیستم

هدف این سند صرفاً توصیف اجزای فنی نیست، بلکه نمایش بلوغ فکری تیم در طراحی سیستم قابل رشد، امن و قابل نگهداری است. این سند باید به‌وضوح نشان دهد که تیم فقط «کد ننوشته»، بلکه درباره مقیاس، امنیت، هزینه، عملیات، ریسک و آینده سیستم فکر کرده است.

System Architecture Documentپاسخ می‌دهد به این پرسش‌های کلیدی:

سیستم چگونه کار می‌کند؟

چرا این معماری انتخاب شده؟

اگر کاربر ۱۰ برابر شود چه اتفاقی می‌افتد؟

اگر یکی از اجزا از کار بیفتد، چه می‌شود؟

هزینه فنی این تصمیم‌ها چیست؟

چه بدهی فنی آگاهانه پذیرفته شده است؟

2.اصول راهنمای معماری(Architecture Principles)

معماری این سیستم بر اساس مجموعه‌ای از اصول طراحی شده که در تمام تصمیم‌های فنی لحاظ شده‌اند. این اصول نه شعاری، بلکه عملیاتی هستند.

2.1سادگی قابل رشد(Scalable Simplicity)

سیستم در فازهای اولیه عمداً ساده نگه داشته شده، اما به‌گونه‌ای که مسیر رشد مسدود نشود. از over-engineering پرهیز شده، اما نقاط گلوگاه بالقوه از ابتدا شناسایی شده‌اند.

2.2جداسازی مسئولیت‌ها(Separation of Concerns)

هر جزء سیستم یک مسئولیت مشخص دارد. این اصل باعث کاهشcoupling ، افزایش تست‌پذیری و ساده‌تر شدن توسعه تیمی شده است.

2.3طراحی برای خطا(Design for Failure)

فرض معماری این است که خطا اجتناب‌ناپذیر است. بنابراین سیستم باید بتواند خطا را تشخیص دهد، محدود کند و بازیابی شود.

.3نمای کلی معماری سیستم(High-Level Architecture Overview)

در سطح کلان، سیستم از چند لایه اصلی تشکیل شده است:

لایه(Client (Frontend

لایه API وBackend Services

لایه Data وStorage

لایه Integration با سرویس‌های خارجی

لایه Observability وMonitoring

لایه Security وIdentity

این تفکیک باعث می‌شود هر لایه مستقل توسعه و مقیاس‌پذیر باشد و تغییر در یک لایه کمترین اثر را بر سایر لایه‌ها داشته باشد.

4.معماریFrontend

Frontendبه‌عنوان نقطه تماس مستقیم کاربر با سیستم، نقش حیاتی در تجربه محصول دارد. طراحی این لایه نه‌تنها از منظرUI ، بلکه از منظر معماری نیز اهمیت دارد.

4.1مسئولیت‌هایFrontend

نمایش داده به کاربر به‌صورت شفاف و قابل فهم

مدیریت وضعیت‌های موقت(loading) ، error، (empty

اعتبارسنجی اولیه ورودی‌ها

تعامل امن باAPI ها

4.2تفکیک لایه‌هایFrontend

کد Frontend به لایه‌های مشخص تقسیم شده:

Presentation Layer

State Management

API Client Layer

Error Handling Layer

این تفکیک باعث می‌شود تغییر در API یا منطق کسب‌وکار، کمترین اثر را بر UI داشته باشد.

5.معماری Backend و سرویس‌ها

Backendقلب سیستم است و بیشترین پیچیدگی فنی در این بخش قرار دارد. معماری Backend به‌صورت Service-Oriented طراحی شده، با این امکان که در آینده به Microservice تکامل یابد.

5.1دلیل عدم Microservice کامل در فاز اول

انتخاب معماری کاملاً Microservice در فاز MVP عمداً رد شده است، زیرا:

پیچیدگی عملیاتی بالا

هزینه DevOps زیاد

نیاز به تیم بزرگ‌تر

در عوض، Modular Monolithانتخاب شده که مزایای سادگی و مسیر مهاجرت را همزمان دارد.

6.جریان داده(Data Flow)

داده در سیستم از لحظه ورود تا ذخیره و پردازش، مسیر مشخصی دارد. هیچ داده‌ای بدون اعتبارسنجی، لاگ و کنترل وارد سیستم نمی‌شود.

6.1مراحل جریان داده

دریافت درخواست ازClient

اعتبارسنجی ورودی

احراز هویت و مجوز

اجرای منطق کسب‌وکار

ذخیره یا بازیابی داده

لاگ‌گذاری و متریک

پاسخ بهClient

این شفافیت جریان داده برای امنیت، اشکال‌زدایی و مقیاس‌پذیری حیاتی است.

7.طراحیAPI ها

APIها قرارداد رسمی بینFrontend ، Backendو سرویس‌های خارجی هستند. طراحیAPI ها به‌صورت Contract-First انجام شده است.

7.1اصول طراحیAPI

نسخه‌بندی شفاف

پاسخ‌های قابل پیش‌بینی

Error Codeهای استاندارد

عدم افشای اطلاعات داخلی

7.2مدیریت تغییراتAPI

هر تغییر در API باید:

Backward Compatibleباشد

مستندسازی شود

در محیط staging تست شود

8.امنیت(Security Architecture)

امنیت به‌عنوان لایه تزئینی در نظر گرفته نشده، بلکه بخشی از معماری است.

8.1احراز هویت(Authentication)

سیستم از مکانیزم احراز هویت مبتنی بر Token استفاده می‌کند که:

قابل ابطال است

تاریخ انقضا دارد

قابل مانیتور است

8.2مجوزدهی(Authorization)

سطوح دسترسی به‌صورت Role-Based تعریف شده و در تمام مسیرهای حساس اعمال می‌شود.

9.مدیریت داده و دیتابیس

انتخاب دیتابیس بر اساس الگوی دسترسی داده انجام شده، نه صرفاً محبوبیت تکنولوژی.

9.1دلایل انتخاب

حجم تراکنش

نیاز بهconsistency

هزینه نگهداری

امکان scale در آینده

9.2استراتژیMigration

تغییر ساختار داده بدون downtime طراحی شده و rollback در نظر گرفته شده است.

10. Observability و Monitoring

سیستم بدون observability ، یک جعبه سیاه است. بنابراین از ابتدا ابزارهای زیر در نظر گرفته شده‌اند:

Logging ساختاریافته

Metrics کلیدی

Alerting هوشمند

11. مدیریت خطا و Incident

خطاها دسته‌بندی شده‌اند:

خطای کاربر

خطای سیستم

خطای وابستگی خارجی

برای هر دسته، رفتار سیستم مشخص شده است.

12.مقیاس‌پذیری(Scalability Strategy)

سیستم طوری طراحی شده که:

در فاز اول با حداقل هزینه کار کند

در فاز رشد بدون بازنویسی کامل مقیاس بگیرد

13.تصمیم‌های معماری وADR ها(Architectural Decision Records)

تمام تصمیم‌های مهم معماری مستند شده‌اند:

گزینه‌های بررسی‌شده

دلیل انتخاب

پیامدهای مثبت و منفی

این بخش نشان می‌دهد تیم تصمیم‌ها را تصادفی نگرفته است.

14.بدهی فنی (Technical Debt) آگاهانه

برخی تصمیم‌ها عمداً بدهی فنی ایجاد می‌کنند، اما:

ثبت شده‌اند

هزینه‌شان معلوم است

زمان پرداخت مشخص دارد

15.ریسک‌های معماری

ریسک‌هایی مثل:

افزایش ناگهانی ترافیک

وابستگی به سرویس خارجی

خطای انسانی درdeploy

شناسایی و برایشان برنامه کاهش تعریف شده است.

16.جمع‌بندی معماری

این معماری نشان می‌دهد سیستم:

قابل رشد است

قابل نگهداری است

وابسته به یک فرد نیست

و آماده عبور از MVP به محصول واقعی است

System Architecture Document

17. استراتژی مقیاس‌پذیری عددی (10x / 50x / 100x Growth)

یکی از اشتباهات رایج در طراحی معماری این است که تیم یا بیش‌ازحد برای مقیاس آینده طراحی می‌کند (Over-Engineering) یا اصلاً به آن فکر نمی‌کند. در این پروژه، مقیاس‌پذیری به‌صورت سناریومحور و عددی تحلیل شده است.

17.1 سناریوی رشد ۱۰ برابر

در سناریوی رشد ۱۰ برابر، فرض بر این است که:

تعداد کاربران فعال روزانه ۱۰ برابر می‌شود

تعداد درخواست‌های API به‌صورت خطی رشد می‌کند

الگوی رفتاری کاربران تغییر بنیادین نمی‌کند

در این سطح، معماری فعلی با:

scale افقی سرویس‌ها

بهینه‌سازی query ها

caching در لایه‌های مشخص

قادر به پاسخ‌گویی است، بدون نیاز به تغییر بنیادین در ساختار.

17.2سناریوی رشد ۵۰ تا ۱۰۰ برابر

در سناریوی رشد شدید، فرضیات سخت‌گیرانه‌تر می‌شوند:

افزایش هم‌زمان کاربران

رشد ناهمگون در برخیendpoint ها

فشار بالا روی دیتابیس

در این حالت، معماری از ابتدا نقاط مهاجرت را پیش‌بینی کرده است:

استخراج سرویس‌های پرمصرف

جداسازیread/write

معرفی message queue برای عملیات غیرهمزمان

نکته کلیدی این است که این مهاجرت‌ها قابل انجام بدون توقف سرویس طراحی شده‌اند.

18.معماری داده پیشرفته(Advanced Data Architecture)

18.1تفکیک داده‌های عملیاتی و تحلیلی

برای جلوگیری از فشار تحلیلی روی دیتابیس عملیاتی، داده‌ها به دو مسیر تفکیک می‌شوند:

داده‌هایTransactional

داده‌هایAnalytical

این تفکیک اجازه می‌دهد گزارش‌گیری و تحلیل بدون آسیب به عملکرد سیستم اصلی انجام شود.

18.2سیاست نگهداری داده(Data Retention Policy)

همه داده‌ها به‌صورت نامحدود نگهداری نمی‌شوند. برای هر نوع داده:

دوره نگهداری

سطح حساسیت

سیاست حذف یا آرشیو

تعریف شده است.

این تصمیم هم از نظر هزینه و هم از نظر انطباق قانونی حیاتی است.

19. امنیت پیشرفته و Threat Modeling

امنیت در این سیستم به‌صورت لایه‌ای (Defense in Depth) طراحی شده است.

19.1 شناسایی تهدیدها

تهدیدها به چند دسته تقسیم شده‌اند:

تهدیدهای خارجی (حمله، نفوذ، سوءاستفاده)

تهدیدهای داخلی (دسترسی بیش‌ازحد، خطای انسانی)

تهدیدهای سیستمی(dependency compromise)

برای هر دسته، سناریوهای محتمل بررسی شده‌اند.

19.2پاسخ به تهدید(Incident Response)

در صورت وقوع حادثه امنیتی:

مسیر تشخیص

ایزوله‌سازی

بازیابی

اطلاع‌رسانی

از قبل تعریف شده است.

این بخش نشان می‌دهد امنیت فقط پیشگیری نیست، بلکه آمادگی برای بدترین حالت است.

20.مدیریت وابستگی‌ها و ریسکThird-party

سیستم به سرویس‌های خارجی وابسته است، اما این وابستگی‌ها کورکورانه نیستند.

20.1سیاست انتخابVendor

هر سرویس خارجی بر اساس:

پایداری

سابقه

SLA

هزینه خروج(Exit Cost)

ارزیابی شده است.

20.2 Failoverو Plan B

برای سرویس‌های حیاتی:

سناریوی قطع

رفتار سیستم

تجربه کاربر در حالتdegraded

مشخص شده است.

21.استراتژی استقرار(Deployment Strategy)

21.1محیط‌ها(Environments)

سیستم حداقل شامل:

Development

Staging

Production

است و هر محیط نقش مشخصی دارد.

21.2فرآیندDeploy

Deployبه‌صورت:

قابل تکرار

قابلrollback

با حداقلdowntime

طراحی شده است.

هدف این است که deploy یک رویداد پرریسک نباشد.

22. CI/CDو کنترل کیفیت فنی

Pipelineتوسعه به‌گونه‌ای طراحی شده که:

خطای انسانی را کم کند

بازخورد سریع بدهد

کیفیت را enforce کند

تست‌ها، lintingو بررسی امنیتی بخشی از pipeline هستند، نه مرحله‌ای اختیاری.

.23مدیریت پیکربندی وSecrets

هیچ داده حساسی در کد ذخیره نمی‌شود:Secrets .

ایزوله‌اند

قابل rotation هستند

دسترسی‌شان محدود است

این تصمیم از بروز فجایع امنیتی رایج جلوگیری می‌کند.

24. Disaster Recovery & Business Continuity

24.1سناریوهای فاجعه

سناریوهایی مثل:

از دست رفتن دیتابیس

قطع کامل زیرساخت

خطای انسانی گسترده

تحلیل شده‌اند.

24.2اهداف بازیابی

برای هر سناریو:

) RTOزمان بازیابی(

) RPOمیزان داده قابل از دست رفتن(

تعریف شده است.

.25هزینه زیرساخت وTrade-off های مالی

معماری فقط فنی نیست، اقتصادی هم هست. برای هر تصمیم:

هزینه ماهانه

هزینه رشد

نقطه بهینه

بررسی شده است.

هدف این است که هزینه زیرساخت متناسب با رشد درآمد افزایش یابد، نه جلوتر از آن.

26. Observability پیشرفته و تصمیم‌گیری داده‌محور

متریک‌ها فقط برای مانیتور نیستند، بلکه برای تصمیم‌گیری استفاده می‌شوند:

چه چیزی باید scale شود؟

کجا bottleneck است؟

چه فیچری هزینه بیشتری ایجاد کرده؟

27. معماری تیمی و تأثیر آن بر سیستم

معماری سیستم با معماری تیم هم‌راستا طراحی شده است Conway’s Law .به‌صورت آگاهانه در نظر گرفته شده تا:

ownership واضح باشد

وابستگی تیم‌ها حداقل شود

.28برنامه تکامل معماری در ۱۸–۲۴ ماه

این معماری ایستا نیست. مسیر تکامل آن مشخص شده:

چه زمانی monolith شکسته می‌شود

چه زمانی سرویس‌های جدید اضافه می‌شوند

چه زمانی ابزارها تغییر می‌کنند

.29معیار بلوغ معماری

معماری زمانی موفق تلقی می‌شود که:

توسعه feature سریع‌تر شود

خطاها زودتر کشف شوند

وابستگی به افراد کاهش یابد

.30جمع‌بندی نهایی بخش

این سند نشان می‌دهد که:

معماری انتخاب‌شده تصادفی نیست

تیم به مقیاس، امنیت، هزینه و عملیات فکر کرده

سیستم برای MVP ساخته نشده، برای رشد واقعی آماده شده است