نحوه پاسخگویی به نیازمندی های مهم معماری

# زمان پاسخگویی به درخواست ها

با توجه به پیشرفت تکنولوژی‌ها و شناخت تیم به ابزارها و تجربه در ایجاد نرم‌افزارهای مختلف، این مسئله نگران‌کننده نیست و می‌توان با ابزارهای امروزی پاسخ مطلوب داد. قسمت نگران‌کننده برای این بخش، در زمان scale کردن سیستم و رشد آن است که بار زیاد روی سیستم می‌تواند آن را در پاسخ‌دهی به درخواست‌ها دچار مشکل کند. برای رفع آن، از تکنیک‌هایی مانند load balancing استفاده می‌کنیم تا بار زیاد را بر روی مجموعه‌ای از سرورها توزیع کنیم. استفاده از این تکنیک و تکنیک‌های دسته‌ی Redundant Spare که رویکرد replication دارند باعث می‌شود بتوان در زیر لود زیاد تحمل کرد. اگر این مسئله بسیار جدی باشد، می توان به سمت Graceful degradation نیز رفت. تیم تصمیم دارد این مراحل را به تدریج و با توجه به پیش‌بینی‌ها و داده‌های به دست آمده از میزان بار سیستم انجام دهد و به یکباره این موارد را لحاظ نمی‌کند تا در هزینه اولیه ایجاد سیستم صرفه‌جویی کند.

# مدت زمان uptime

مدت زمان آپ بودن سیستم از جمله مسائل مهمی است که می‌تواند در مسائل کسب‌وکار نیز تاثیرگذار باشد و به نوعی اعتبار سیستم و تجربه کاربری کاربران سیستم را تحت تاثیر قرار می‌دهد. بنابراین تیم تصمیم گرفته است که میزان آپ بودن سیستم را 99درصد قرار دهد. این عدد در حالت کلی و معمول برقرار است. مواردی مانند تغییرات زمان نگهداری، ریلیزهای آینده، حملات مختلف و ... آن را تحت تاثیر قرار دهند. برای رفع این مشکل از تکنیک‌هایی مانند Redundant Spare در آینده و از مکانیزم های Rollback یا Exception Handling از ابتدای ایجاد نرم‌افزار و رویکرد Retry برای اکشن‌هایی که ممکن است به دلیل خطاهای جزئی رخ دهند استفاده می‌شود. در بار زیاد نیز ممکن است سیستم از تکنیک Graceful degradation برای رفع نیازهای خود استفاده کند.

# مستقل از پلتفرم

رویکرد platform independent امروز توسط عمده زبان‌های برنامه نویسی ساپورت می‌شود و تیم مشکلی از نظر ایجاد نرم‌افزار ندارد. برای کلاینت‌ها نیز چون سیستم قرار است تحت وب باشد، بنابراین این مشکل نیز قابل حل است و تیم نگرانی از این بابت در گام فعلی ندارد.

# مقیاس پذیری

مطابق پیش بینی اولیه از مقیاس سیستم، سیستم فعلی به گونه‌ای طراحی شده است که بتواند نیازمندی‌ها را پاسخ دهد. معماری سیستم از یک ساختار کامپوننتی پیروی می‌کند که تا حد ممکن از هم تفکیک شده‌اند و ارتباطات بین آنها تا حد خوبی سطح بالاست. این رویکرد بدین جهت بوده که در صورت لود بالا، تیم بتواند به شکل راحت تری ماژول‌ها را به صورت سرویس هایی تفکیک کند و بتواند خود را اسکیل کند و مشکلات مربوطه را حل کند. همچنین رویکردهایی مانند horizontal scaling یا vertical scaling در اولویت بالاتری قرار دارند. چرا که هزینه تغییر معماری زیاد است و تا جای ممکن از آن جلوگیری می‌شود. اما این مکانیزم و شیوه طراحی را تیم برای مشکلات جدی آینده از ابتدا لحاظ کرده است.

# امنیت

مسئله امنیت سیستم از درجه اهمیت بالایی برخوردار است. برای حفظ محرمانگی داده‌ها(با توجه به اینکه سیستم شامل محتواهای پولی است که نباید به رایگان در دسترس قرار بگیرند) از مکانیزم‌ها امنیتی لازم مانند Authenticate Actors و Authorize Actorsو Encrypt Data و Validate Input استفاده می‌کند. این مکانیزم‌ها به ما اجازه می دهند پیغام های سیستم به شکل محرمانه به کاربرهای شناخته شده و مورد تایید برسد. همچنین تیم محدودیت های لازم را برای جلوگیری از سواستفاده‌ی کاربران مورد تایید از محتواهای در اختیار قرارداده شده اعمال می شود. همچنین برای جلوگیری از دسترسی های غیرمجاز، تکنیک Firewall کارآمد و کم هزینه است.

# نگهداری داده

برای حفظ داده‌ها و تراکنش‌های مالی و قابلیت اطمینان محصول در حوزه مالی، پکیج مالی در معماری به صورت جدا توسعه داده می‌شود تا بتوان مکانیزم‌های لازم را برای اصالت داده‌‌ها، برقراری ارتباط امن با درگاه پرداخت، تراکنش‌های کاربر و موجودی‌های کیف پول کاربران انجام داد. امن کردن دیتابیس به وسیله مکانیزم‌های امنیتی و پشتیبان‌گیری از داده‌های لازم و حیاتی از دیگر روش‌های کارآمد برای این مسئله هستند.