دانشگاه صنعتی خواجه نصیرالدین طوسی

عنوان:

طراحی معماری سامانه شطرنج

دانشجویان:

یونس شفیعی، مهدی خوشمرام‌زاده

استاد راهنما پروژه:

دکتر محمدمهدی اثنی عشری

۱۵ بهمن ۱۴۰۱

بسم اللہ رحمان رحیم

فهرست مطالب

[۱] *performance*
[۲] *Security*

فصل ۱

مقدمه

در سامانه‌های بازی‌های چندنفره آنچه نیاز است اصل حفظ اعتماد و امانت، کارایی و قابل اتکا بودن سامانه می‌باشد. بر این اساس صفاتی چون امنیت و کارایی در صدر اهم قرار داده شدند. برای طراحی انجام شده، تصمیم برای معماری یک تصمیم پسین بوده است. بازی شطرنج دارای قواعد محدود است. شطرنج نیازمند آن است که کنترل و پردازش زمان قابل قبولی داشته باشد. همینطور باید اشاره کرد که دیتامدل پیاده‌سازی خود بازی دارای سادگی است. اگر صفحه شطرنج مانند شکل زیر باشد، حرکت مهره تنها باید مقصد آن ذخیره و پردازش شود. برای مثال اگر $Rb2$ نوشته شود به این معنی است که $rook$ در هر نقطه از صفحه باید به خانه‌ی $b2$ برود. پس کار ساده‌ای به لحاظ حفظ ترتیب بازی و ذخیره‌سازی روند بازی بر عهده‌ی طراح و توسعه‌دهنده است.

شکل ۱.۱:

فصل ۲

سناریوهای کیفی

۱.۲ کارایی[۱]

در بازی شطرنج علاوه بر *checkmate* کردن حریف میتوان با غلبه زمانی پیروز شد. به دلیل سوددهی و اعلام پیروزی طرفین بر اساس زمان -صفر شدن زمان حریف-، کارایی و پردازش قابل قبول بازی دارای اهمیت بسیار زیادی است. تصور کنید اگر حرکتی در یک بازی ۶۰ *sec* در زمانی بیشتر از ۰.۵ ثانیه اعمال شود. در چنین شرایطی طرفین ناکارآمدی این فرمت بازی را احساس میکنند و طراحی و محصول با شکست رو به رو می‌شود. میشود در زمانهای مختلف تعداد کاربران با افزایش مشخصی رو به رو شود و ما نیازمند آن باشیم که بتوانیم واحدهای پردازشی بازی را افزایش دهیم. کارفرمایان و سرمایه‌گذاران برای هر بخش سامانه زمان مشخصی را برای *deadline* مشخص میکنند و برآورده کردن آن دارای اهمیت می‌باشد. در معماری باید توانایی لازم جهت حفظ این زمان لحاظ شود. آنچه در بالا گفته شد مصادیق سناریوهای مختلف برای خصیصه و صفت کارایی بوده است و در زیر به صورت عام و خلاصه، جدولی قرار داده شده است.

Source	Internal condition - User
Stimulus	Internal incidents - User request
Artifact	System (Specifically game service)
Environment	Normal mode - Peak load
Response	Load balance – Service initialization
Response Measure	Latency – Miss rate - Deadline

شکل ۱.۲: جدول سناربو کیفی برای کارایی

[۱] *performance*

۲.۲ امنیت[۲]

در مسابقات شطرنج آنچه دارای اهمیت می‌باشد حفظ حرکات بازیکن‌های بازی می‌باشد. تصور کنید دو بازیکن مطرح شطرنج، فینال یک مسابقه را در بستر سامانه ما بازی میکنند. حال حین بازی حرکتی برای یک سمت بازی بر روی صفحه ظاهر شود، یا با وجود گذشت زمان یکی از طرفین، حرکات کاربر به دلیل مداخله بیرونی صورت نپذیرد. چنین اتفاقاتی ضرر هنگفت به اعتبار و هزینه‌ی سامانه وارد میکند و انجام مسابقات چند هزار دلاری را در سامانه ناممکن می‌سازد. علاوه بر چنین سناریوهایی، حفظ اطلاعات و حریم شخصی کاربران بر عهده سامانه می‌باشد. باید بتوان معماری مطلوبی برای چنین امری طراحی کرد. برای مقابله با حملات گسترده نیز باید راهکار و مقابله‌ای صورت پذیرد. به دلیل همان مسابقات دارای اهمیت، امکان حملات DOS وجود دارد. یک معماری مطلوب برای چنین محصولی باید بتواند تا حد امکان با چنین خطراتی مقابله داشته باشد.

سناریوهای بالا مصادیق مشخصی برای خصیصه‌ی امنیت بودند و در زیر جدولی اجمالا به صورت کلی آنها را گردآوری کرده است:

Source	Human or systematical attack
Stimulus	Unauthorized attempt is made to display data, change or delete data, access system services, change the system's behavior, or reduce availability.
Artifact	Services and data within the system
Environment	During providing Service by the system
Response	Service concealing – Prevent unauthorized data access
Response Measure	For web server: how much time passed before an attack was detected, how many attacks were resisted – Recover time

شکل ۲.۲: جدول سناربو کیفی برای امنیت

۳.۲ سایر صفات کیفی

برای این محصول صفات کیفی دیگری هم دارای اهمیت می‌توانند باشند. از جمله مقیاس‌پذیری[۳] و قابل اتکا بودن[۴] از صفات محرز دیگر است. در ادامه به آنها در موقعیت تحلیل معماری خواهیم پرداخت.

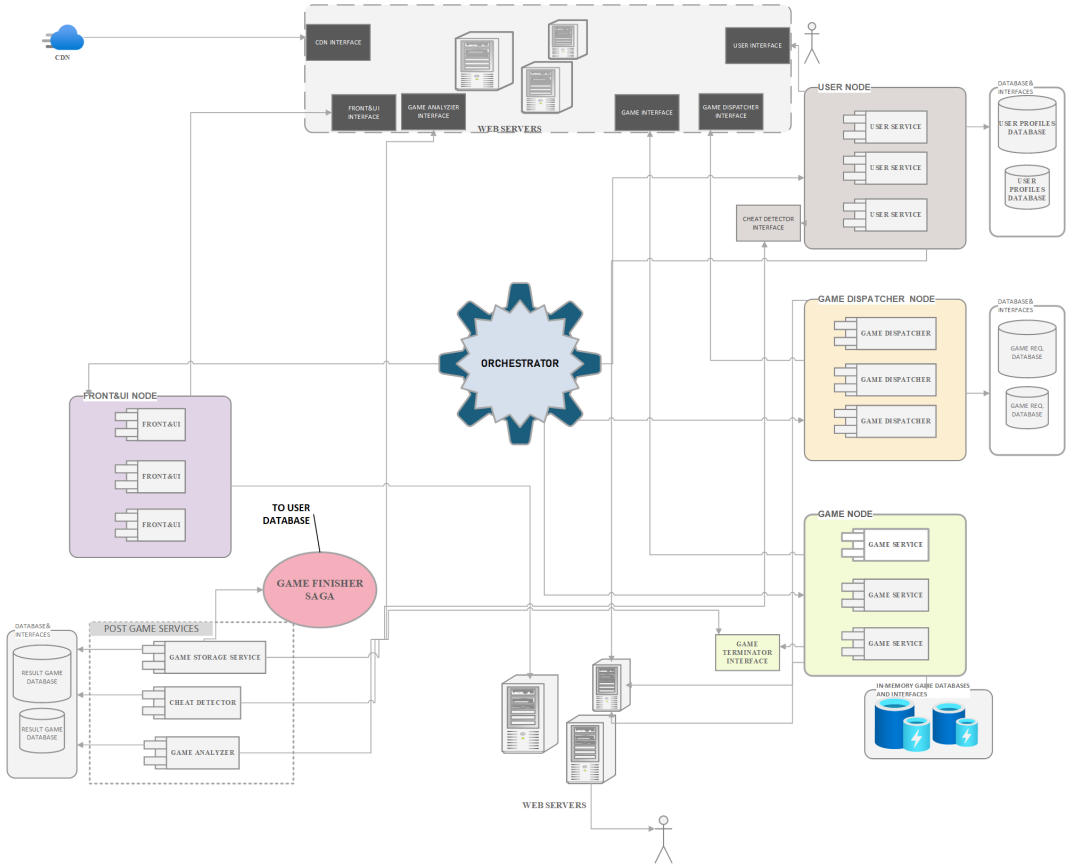
---

[۲] *Security*
[۳] *Scalability*
[۴] *Reliability*

فصل ۳

معماری سامانه شطرنج

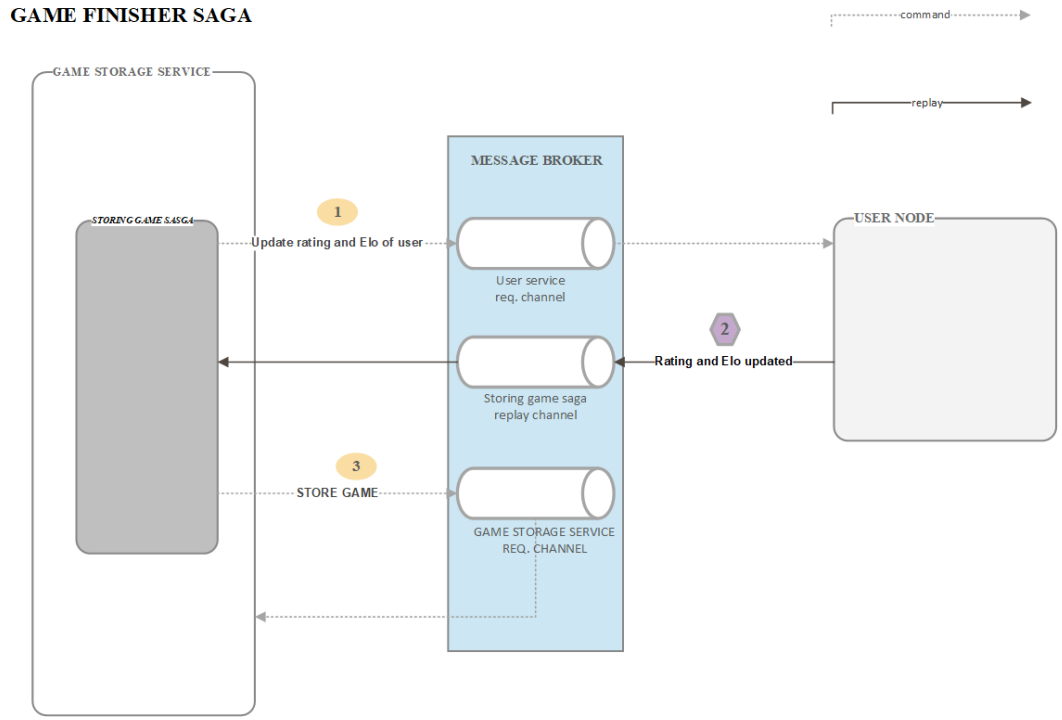
آیا برای طراحی معماری سامانه به الگوی مشخصی رجوع شده است؟ هم بله و هم خیر. بر اساس نیازها و صفات کیفی، سعی بر آن بوده است که بتوان *Component* ها را به فرم مناسب ایجاد و ارتباط لازم را بینشان شکل داد. این هدف‌گذاری سبب شد که به الگوی میکروسرویس نزدیکتر شده و از دانش پیشین برای طراحی چنین معماری‌ای استفاده گردد. توجه بشود خصلت‌های مناسب و مفید الگوی میکروسرویس از جمله مقیاس‌پذیری و امنیت و البته کارایی از دلایل عمده‌ی بهره بردن بیشتر از الگوساختارهای میکروسرویسی بوده است. در ادامه نمودار *C&C* با رعایت جهت وابستگی خواهیم دید.



شکل ۱.۳: نمودار *C&C* با رعایت جهت وابستگی

ارتباط کاربر و سامانه تنها از طریق *API Gateway* صورت می‌پذیرد. نام مشخص‌تر و عملیاتی‌تر *web server* در نمودار نمایانگر *API Gateway* می‌باشد. درخواست و پاسخ کاربران در این نقطه پردازش و مسیریابی می‌شوند. این لایه و *component*

قادر است در مقابله با حملات احتمالی سیاست‌های مقابله‌ی مناسبی را اتخاذ کند. میکروسرویس‌های سامانه بر اساس کارکرد و هدف‌گذاری مناسب، بخش‌بندی شده‌اند. سرویس‌هایی مثل سرویس *user*، سرویس *dispatcher*، سرویس *game*، سرویس‌های پسین بازی، سرویس *front-end* در سامانه حضور دارند. این سرویس‌ها همگی در خدمت سرویس با اهمیت *game* هستند. این سرویس باید کارایی بالایی داشته باشد. در این سامانه بازی‌ها به شکل‌های مختلفی میتواند ایجاد شوند. یک)کاربری بدون ثبت‌نام. دو)کاربری با ثبت نام با کاربران حاضر برخط. سه)کاربری با ثبت نام با کاربری مشخص. در حالت اول، سامانه توسط سرویس *dispatcher* یک حریف برخط برای این کاربر در رنج امتیاز مختلف به صورت اتفاقی می‌یابد. سپس توسط سرویس *game* اجرا و در نهایت پس از پایان بازی داده‌های بازی به صورت موقت توسط سرویس *game storage* در پایگاه داده‌ی *result game* ذخیره میشود. این بازی‌ها بعد مدتی از پایگاه داده پاک خواهند شد. در حالت دوم، پس از درخواست کاربر برای بازی، اطلاعات امتیازی آن کاربر در سرویس *dispatcher* جهت یافتن حریف در همان رنج امتیاز استفاده میشود. سپس پس از یافتن حریف بازی در سرویس *game* مراحل بعدی را مانند حالت اول طی میکند. لازم به ذکر است اطلاعات بازی به صورت دائمی ذخیره می‌شوند و توسط *game finisher saga* (شکل ۳) ذخیره سازی در سرویسهای توزیع شده به صورت کامل صورت میگیرد. نیاز است امتیاز کاربر در پایگاه داده *user* به روز شود. بازی انجام شده در پایگاه داده سرویس *game storage* ذخیره می‌شود. حالت سوم



شکل ۲.۳: معماری *saga*

مانند حالت دوم است به جز آنکه کاربر آیدی کاربر مشخصی را به *dispatcher* میدهد. این بازی بین این دو باید اجرا شود. سپس مراحل تکراری نهایی‌سازی ذخیره‌ی بازی صورت پذیرد.

سرویس‌هایی مانند سرویس *cheat detector* و یا *game analyzer* سرویس‌های پسینی هستند که بار مشخصی بر بخش‌های مختلف سامانه نباید به وجود آورند.

در این معماری ما میتوانیم از هر سرویس *replica* های مختلف در فرمت مختلف داشته باشیم و چنین هم خواهد بود. نیاز است در مورد بخش نهایی، یعنی *orchestrator* توضیح کافی داده شود.

درواقع قلب این معماری *orchestrator* میباشد. تمامی شبکه‌سازی داخلی در این معماری توسط *orchestrator* انجام می‌شود. *orchestrator* قابلیت اجرا و یا پایان دادن به یک سرویس را دارد. *orchestrator* میتواند میزان بار بر روی سرویس‌ها را تشخیص دهد. *orchestrator* در این سامانه هر مجموعه سرویس هم‌نوع را در یک خوشه یا *node* قرار میدهد. این عملیات به جهت کارکرد و نوع پیاده‌سازی مدنظر معمار صورت می‌پذیرد. تمامی قابلیت‌های بالا برای *orchestrator* با اتکا بر همین خوشه‌بندی رخ میدهد.

با تشکر