بررسي كنيد آيا Class Diagram ارائه شده توسط گروه شما در فاز Structural Modelling

نيازمند بهينه سازي مي باشد. براي پاسخ خود دليل بياوريد) مهمترين كار اين است كه خصوصيات

Couplingو Cohesion را در كلاسهاي طراحي شده بررسي كنيد.

پاسخ :

بله تنها یک کلاس دیگر به عنوان کلاس قرار داد های بین مربی و ورزشکار باید اضافه می شد تا اصل Couplingو Cohesionبیشتر رعایت شود.

کلاس های ما با توجه به این دو اصل بنا شده اند. البته به صورت مطلق از آنها پیروی نمی کنند.

بر اساس اصل coupling کلاس ها می بایست کم ترین نیازمندی ارتباطی را به یک دیگر داشته باشند. در حالت ایده آل بدین شکل است که کلاس ها اصلا به همدیگر وابسته نباشند. اصل cohesion میزان انسجام درونی اشیا کلاس ها را بیان می کند. و می توان گفت از این منظر کلاس های ما کاملا بر این اصل منطبق هستند. هر کلاس وظیفه ی انجام یک عملیات خاص را دارد و پیوستگی اشیا برای انجام یک عملیات خاص کاملا مشهود است.

از چه فرمتي براي مديريت داده ها استفاده مي كنيد. دليل خود را بيان و تحليل كنيد. براي دلايل خود نيازمندي هاي Nonfunctional را هم در نظر بگيريد.

پاسخ :

فرمتی که برای ذخیره اطلاعات کاربران راجع به برنامه های ورزشی و اطلاعات تاریخچه ورزشی و همچنین منطق های برنامه از ان استفاده شده است ، پایگاه داده رابطه ای (Relational Database) و برای گرفتن اطلاعات از SQLاستفاده شده است. برای مثال ما در پروژه از پایگاه داده Postgre استفاده کرده ایم که آپشن های بسیاری در جهت مدیریت داده های ما به ما می دهد.

فقط برای قسمت ارتباط با مربی که همان ویژگی چت برنامه می باشد از NoSQLها و پایگاه داده های غیر رابطه ای استفاده شده است. برای مثال ما در پروژه برای این بخش ها و کلا بخش هایی که تعداد داده ها یا همان حجم داده یا گوناگونی و سرعت تغییر ان زیاد می شود و به نوعی با Big Data سر و کار داریم از Mongo DBاستفاده کرده ایم.

دلیلی که برای استفاده از این تکنولوژی ها داریم این است که:

۱ – به دلیل اینکه ساختار برنامه ما زیاد پیچیده نیست و همچنین بخش مخصوصی از جامعه را مورد بررسی قرار دادیم که همان ورزشکار ها و مربیان می باشند و اینکه اطلاعات آنها به خوبی می تواند در جداول مربوطه جای بگیرد از پایگاه داده رابطه ای برای اینها استفاده کرده ایم. همچنین در رابطه با تاثیر نیازمندی های غیر عملکردی Performanceمانند سرعت (Speed) بسیار تاثیر گذار است و می تواند حجم خوبی از داده هارا مدیریت کند (Capacity). همچنین با Deployکردن این پایگاه داده بر روی یک سرور شخصی یا فضای ابری می توانیم دسترسی پذیر بودن (Availability & Reliability)آن را نیز تضمین کنیم. همچنین از لحاظ نیازمندی های امنیتی مثل سطوح دسترسی (Access Control) هم با توجه به اینکه در Postgreاین امکان را داریم که Roleهای متفاوت برای دسترسی به سیستم طراحی کنیم نیز فراهم می شود. همچنین قابلیت زمر نگاری نیز توسط این نوع موتور پایگاه داده به ما داده می شود(Encryption).

برای چت ها نیز نیازمندی سرعت بسیار تاثیر گذار است چون حجم داده ها بالا می رود و سرعت تغییر نیز در آن ها بسیار زیاد است برای همین پایگاه داده رابطه ای زیاد جوابگو نیاز های ما در بخش چت نیست پس از NoSQL ها استفاده می کنیم.

همچنین ما برای دسترسی به دیتابیس های مختلفمان از ORM نیز استفاده کرده ایم که مفاهیم شی گرایی را به مفاهیم پایگاه داده پیوند می زند و دسترسی ما را راحت تر می کند.

به نظر شما از چه نوع معماري بايستي براي طراحي سيستم خود استفاده مي كنيد. دليل خود را بيان و تحليل كنيد.براي دلايل خود نيازمندي هاي Nonfunctional را هم در نظر بگيريد.

پاسخ :

معماری کلاینت سرور بهترین گزینه برای پیاده سازی نرم افزار ما می باشد.

باتوجه این امر که برخی از ویژگی های نرم افزار ما برای هر کاربر منحصر به فرد می باشد و همچنین نیازمند پردازش هایی است که لازم است از سوی کاربر صورت گیرد معماری کلاینت سرور پیشنهاد می شود.

معماری کلاینت سرور مزایای زیر را برای پروژه به ارمغان می آورد:

۱ - هزینه ی پیاده سازی این معماری نسبتا پایین بوده و به راحتی می توان زیر ساخت آنرا ارتقا داد.

۲ - با توجه به نیاز به کنترل بهیه ی نرم افزار و حفظ امنیت آن معماری کلاینت سرور می تواند این نیاز را پوشش دهد.

۳ - مدل کلاینت سرور برای گسترش GUI و امکانات ظاهری مفید واقع می شود و از آنجا که کاربر با سیستم کار می کند باید ظاهر مناسبی به او ارائه گردد.

۴ - مدل کلاینت سرور برای Maintainability می تواند مفید باشد. چون تغییرات در برنامه سرور منجر به تغییرات کلان در برای کلاینت نخواهد شد.

آیا در طراحي، از Design pattern خاصي استفاده نمودهايد؟ نام آن چيست و چرا؟

پاسخ :

بله !

در قسمت معماری نرم افزار سمت سرور از دیزاین پترن معماری MVC یا همان Model View Controllerاستفاده کرده ایم. زیرا اولا روش خوبی برای جدا کردن منطق های اجرایی یا همان Controller برنامه از بخش Presentationیا همان بخشی که کاربر می بیند که به آن در این دیزاین پترن Viewمی گویند همچنین بخش مربوط به دیتا های ما و دیتابیس ما نیز جدا شده با اینکه ارتباط خودر را با بقیه بخش ها حفظ می کند و به ان بخش Modelمی گویند. از طرفی این معماری که اصطلاحا به آن Monolithنیز گفته می شود بدلیل اینکه برنامه سمت سرور تحت این دیزاین پترن بصورت یکپارچه کار می کند از سرعت بسیار خوبی نیز برخوردار است.

در بخش UI برای ذخیره سازی اطلاعات محلی نظیر Tokenو ... از پایگاه داده SQFLite استفاده شده که در طراحی آن از دیزاین پترن Singletonاستفاده کرده ایم.

در بخش های دسترسی به داده ها و کار کردن با api پایگاه داده از دیزاین پترن Proxy استفاده کرده ایم که بوسیله یکسری کلاس خاص و بوسیله واسط های ان با پایگاه داده ارتباط برقرار می کنیم.

همچنین در نمایش درست اطلاعات در سمت مشتری ، از دیزاین پترن Adapter استفاده کرده ایم به این صورت که اطلاعات را طبق قوانین RESTful به صورت JSON

دریافت کرده و آن هارا به آبجکت های مورد نظر خودمان تبدیل می کنیم که این کار به لطف کلاس هایی که از دیزاین پترن Adapterاستفاده می کنند انجام شده است.