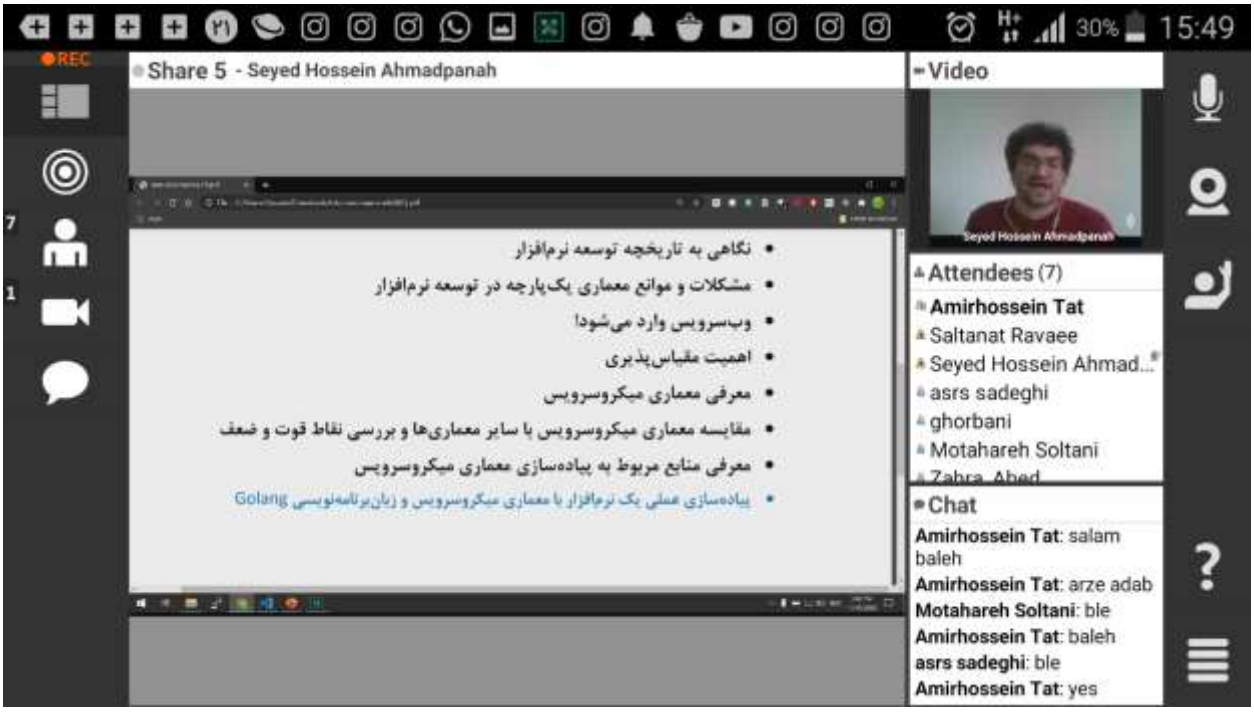
گزارش برگزاری همایش برخط معماری میکروسرویس: دروازه جدید توسعه نرم افزار

این برنامه با توجه به برنامه ریزی های قبلی و لحاظ نمودن شرایط بیماری کرونا باهماهنگی گروه برق والکترونیک در روزچهارشنبه 99/8/21با حضور معاونت محترم پژوهش و فناوری دانشکده سرکار خانم روایی ومدیر محترم گروه سرکار خانم مهندس زمان زاده در بستر نرم افزار ADOBE CONNECT باحضور دانشجویان رشته برق و کامپیوتر برگزار گردید .

میکروسرویس، همان طور که از نام آن مشخص می‌شود، اساساً به سرویس‌های نرم‌افزاری مستقلی گفته می‌شود که کارکردهای تجاری خاصی را در یک اپلیکیشن نرم‌افزاری ارائه می‌کنند. این سرویس‌ها می‌توانند به صورت مستقل از هم نگهداری، نظارت و توزیع شوند. معماری میکروسرویس (Microservices Architecture) به مرور در حال کسب محبوبیت فزاینده‌ای است و امروزه تقریباً در همه پروژه‌های نرم‌افزاری عمده از آن استفاده می‌شود. دلیل اصلی این مسئله ناشی از مزیت‌های آن و مسائلی است که حل می‌کند.برهمین اساس یکی از مباحث مهمی است که دانشجویان رشته های برق و کامپیوتر در دانشگاه فنی و حرفه ای باید فرا بگیرند .



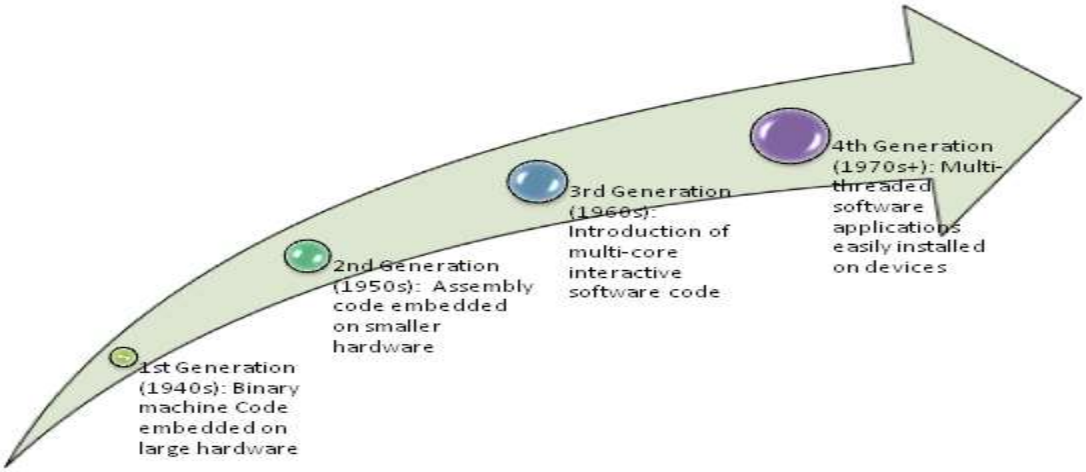
درابتدا استاد احمد پناه به سابقه تکامل برنامه‌نویسی نرم‌افزاراشاره فرمودند

اپلیکیشن‌های نرم‌افزاری همواره باید نگهداری شده و بهبود یابند. همواره این نیاز برای پیاده‌سازی الزامات جدید و بهبود ویژگی‌های موجود وجود خواهد داشت. به علاوه اپلیکیشن‌های نرم‌افزاری به طور موردی نیازمند اجرای سریع‌تر هستند. برخی اوقات ویژگی‌های موجود باید منسوخ شوند.

در نتیجه بهبود و نگهداری اپلیکیشن‌های نرم‌افزاری یک جزء ضروری از چرخه عمر نرم‌افزار است. خلاصه همه بحث‌های فوق این است که ما همواره به نگهداری و بهبود نرم‌افزار نیاز داریم. از این رو باید این فرایند را ساده‌تر کنیم.

تکامل رایانه‌ها

ابتدا باید با فرایند تکامل رایانه‌ها در طی زمان آشنا شویم. در تصویر زیر این تکامل سیستم‌های رایانه‌ای به صورت خلاصه ارائه شده است:



زمانی که رایانه‌ها برای اولین بار در دهه ۱۹۴۰ میلادی ارائه شدند، نرم‌افزار به صورت پانچ شده درون سخت‌افزارهای بزرگ و گران‌قیمت مانند کارت‌های پانچ و نوارهای پانچ جاسازی شده بود. دستورالعمل‌ها به زبان ماشین باینری نوشته شده بودند. متعاقباً اگر لازم می‌شد تغییری پیاده‌سازی شود، یک کارشناس زبان باینری ماشین باید دستورالعمل‌های جدید را در اختیار ماشین قرار می‌داد. این امر یک فرایند بسیار گران‌قیمت بود.

سپس نسل دوم رایانه‌ها در دهه ۱۹۵۰ میلادی به عنوان نسخه بهبود یافته نسل اول رایانه‌ها معرفی شدند. این رایانه‌ها به برنامه‌هایی با زبان اسمبلی نیاز داشتند که در تراشه‌های سخت‌افزاری کوچک‌تری نوشته می‌شد. فلسفه طراحی این رایانه‌ها پیرامون این واقعیت شکل گرفته بود که یادگیری زبان نمادین اسمبلی بسیار راحت‌تر از آموختن کدهای باینری است. در نتیجه اسمبلرها معرفی شدند که کد ماشین را به زبان نمادین اسمبلی تبدیل می‌کردند.

در این مورد نیز هر زمان که تغییری مورد نیاز بود، فرد باید دستورالعمل‌ها را در سخت‌افزار وارد می‌کرد. در نتیجه نرم‌افزار و سخت‌افزار باید از همدیگر جدا می‌شدند.

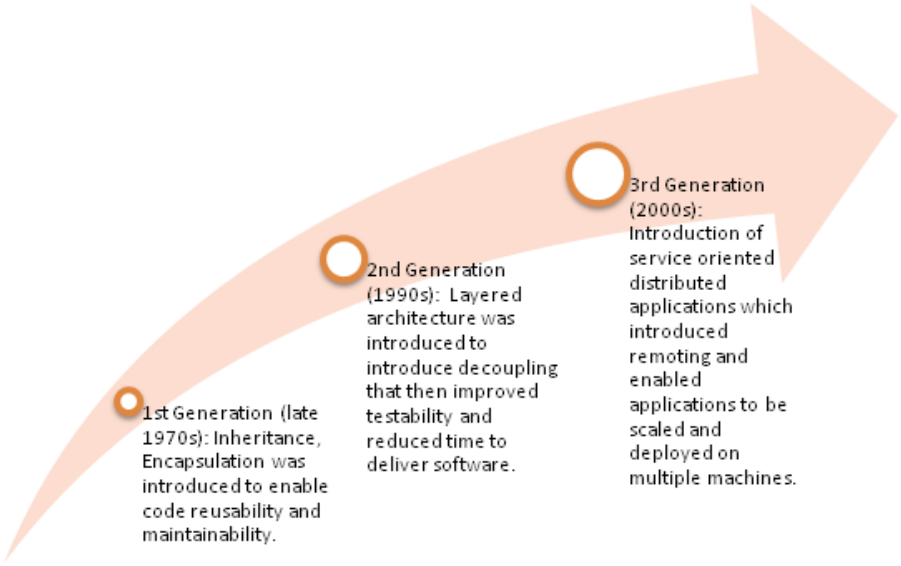
و سپس نسل سوم رایانه‌ها در دهه ۱۹۶۰ میلادی عرضه شدند. این رایانه‌ها به کامپایلر و مفسر برای ترجمه زبان قابل فهم از سوی انسان به کد ماشین نیاز داشتند. این رایانه‌ها کوچک‌تر بودند و می‌بایست نرم‌افزارهای قابل تعامل از سوی کاربر روی آن‌ها نصب می‌شد. نسل سوم رایانه‌ها می‌توانستند چندین اپلیکیشن را همزمان اجرا کنند. زبان‌های برنامه‌نویسی مختلفی معرفی شدند که روی این رایانه‌ها نصب می‌شدند. سخت‌افزار همچنان گران‌قیمت بود؛ اما انعطاف‌پذیری حاصل از جداسازی نرم‌افزار از سخت‌افزار موجب ایجاد فرصت‌های بی‌شماری برای بهبود کارکرد آن‌ها شد.

در نهایت نسل چهارم رایانه‌ها در دهه ۱۹۷۰ میلادی معرفی شدند. این رایانه‌ها دستگاه‌های دستی بودند که می‌توانستند در کف دست جای بگیرند. در این مورد نیز اپلیکیشن‌های جدید معرفی شدند که می‌توانستند روی دستگاه‌ها بدون نیاز به خرید سخت‌افزار جدید نصب شوند. سهولت نصب و قابلیت نگهداری موجب شد که بسیاری از شرکت‌ها بتوانند ایده‌های فناورانه جدیدی را ابداع کنند.

یک طرح مشترک وجود دارد. رایانه‌ها به این دلیل تکامل یافتند که همواره نیاز به نگهداری و بهبود اپلیکیشن‌های نصب شده روی رایانه‌ها وجود داشته است.

تکامل نرم‌افزار

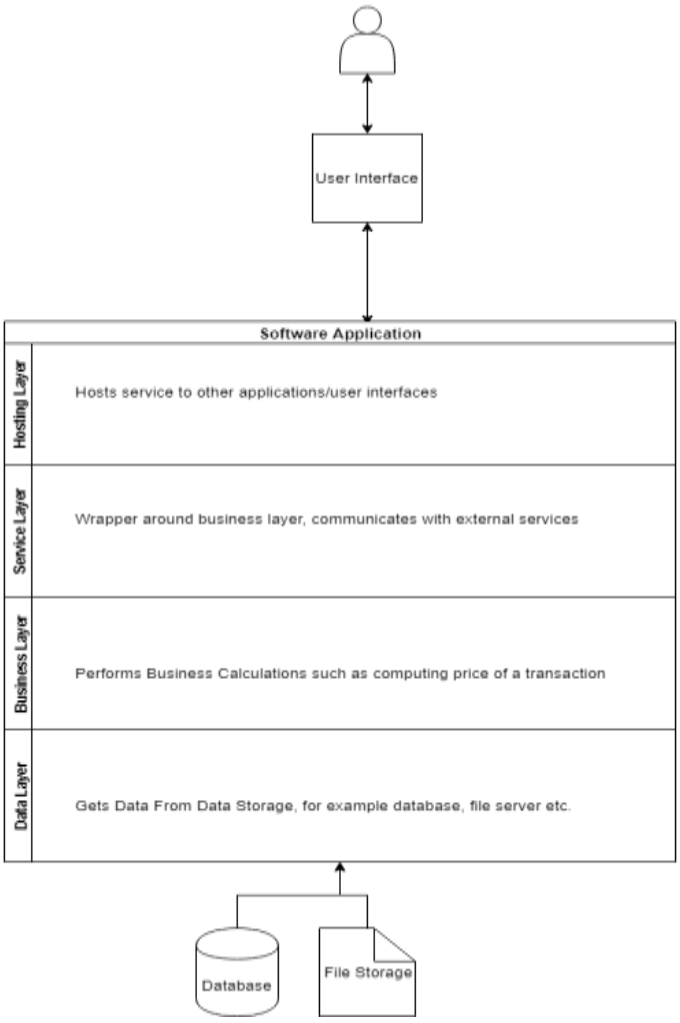
تکامل نرم‌افزار نیز چرخه مشابه را طی کرده است. تصور زیر فرایند تکامل نرم‌افزار را نشان می‌دهد:



- **نسل اول (اواخر دهه ۱۹۷۰ میلادی :(**مفاهیم شیءگرایی مانند وراثت، کپسوله‌سازی، و چندریختی معرفی شدند تا قابلیت استفاده مجدد از کد و قابلیت نگهداری فراهم شوند.
- **نسل دوم (دهه ۱۹۹۰ میلادی :(**اپلیکیشن‌ها با استفاده از معماری لایه‌بندی شده طراحی و پیاده‌سازی شدند. معماری لایه‌بندی شده برای کاهش در هم تنیدگی زیاد بین اجزای مختلف اپلیکیشن‌های نرم‌افزاری معرفی شد. در نتیجه تست نرم‌افزار و زمان برای تحویل نرم‌افزار کاهش یافت. اپلیکیشن‌ها همچنان یکپارچه بودند، یعنی یک واحد منفرد که همه کارکردهای اپلیکیشن وجود داشت در آن کپسوله‌سازی شده بود.
- **نسل سوم (دهه ۲۰۰۰ میلادی :(**در این زمان اپلیکیشن‌های با توزیع مبتنی بر سرویس معرفی شدند. روش طراحی مورد استفاده در این نسل شامل تعریف سرویس ریموت بود که به اپلیکیشن‌ها امکان مقیاس‌بندی و توزیع روی ماشین‌های چندگانه را می‌داد.

معماری اپلیکیشن‌های نرم‌افزاری در بازه‌های زمانی اخیر و مشکلات این معماری‌ها

- در سال‌های اخیر اپلیکیشن‌های نرم‌افزاری در معماری‌های چندلایه پیاده‌سازی می‌شدند. به عنوان نمونه در تصویر زیر یک معماری معمول برای اپلیکیشن نرم‌افزاری سازمانی ارائه شده است:



هر لایه در کد نرم‌افزاری پیاده‌سازی می‌شد و از چندین کلاس و اینترفیس تشکیل یافته بود:

لایه داده

این لایه برای ذخیره‌سازی داده‌ها در پایگاه داده و فایل‌ها پیاده‌سازی شده است. تنها مسئولیت این لایه ارائه داده‌ها از منابع داده‌ای مختلف است.

لایه تجاری

مسئولیت لایه تجاری، بازیابی داده‌ها از لایه داده و اجرای محاسبات است. لایه تجاری نمی‌داند که داده‌ها در فایل یا پایگاه داده یا در موارد دیگر هستند. لایه تجاری به لایه داده‌ها وابسته است.

لایه سرویس

لایه سرویس روی لایه تجاری قرار می‌گیرد. این لایه یک پوشش برای لایه تجاری ایجاد می‌کند که شامل امنیت/گزارش‌گیری/وساطت برای فراخوانی کارکردها است. لایه سرویس به لایه تجاری وابسته است.

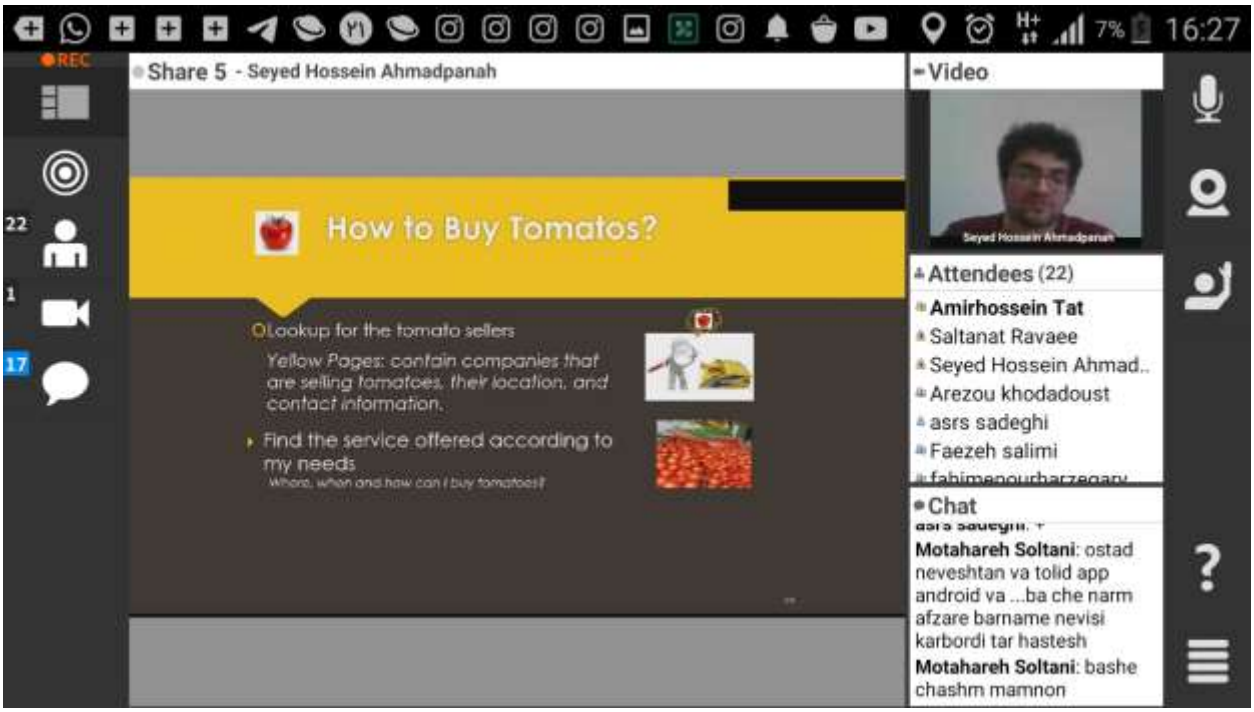
لایه میزبانی

سرویس‌ها از طریق این لایه میزبانی می‌شوند. لایه میزبانی از فناوری‌های مبتنی بر سرویس مانند WCF یا REST APIبرای میزبانی با استفاده از پروتکل‌های مختلف مانند http ، https، tcp، named pipesو غیره بهره می‌گیرد. کل اپلیکیشن به صورت یک پردازش نرم‌افزاری برای نمونه سرویس ویندوز یا وب‌سرویس IIS اجرا می‌شود. اپلیکیشن‌ها از طریق URL مانند https://example.com/myapplication در دسترس هستند.

لایه رابط کاربری

لایه اینترفیس یا رابط کاربری شامل کدهایی است که در لایه میزبانی مورد ارجاع قرار می‌گیرند و به منظور ایجاد امکان تعامل با اپلیکیشن برای کاربران طراحی شده است.

این طراحی به توسعه‌دهندگان امکان می‌دهد که روی یک کارکرد خاص متمرکز شوند، ویژگی را تست کنند، و یا اپلیکیشن را با استفاده از کنترل معکوس از طریق تزریق وابستگی‌ها و میزبانی یک اپلیکیشن روی ماشین‌های متعدد تجزیه کنند.

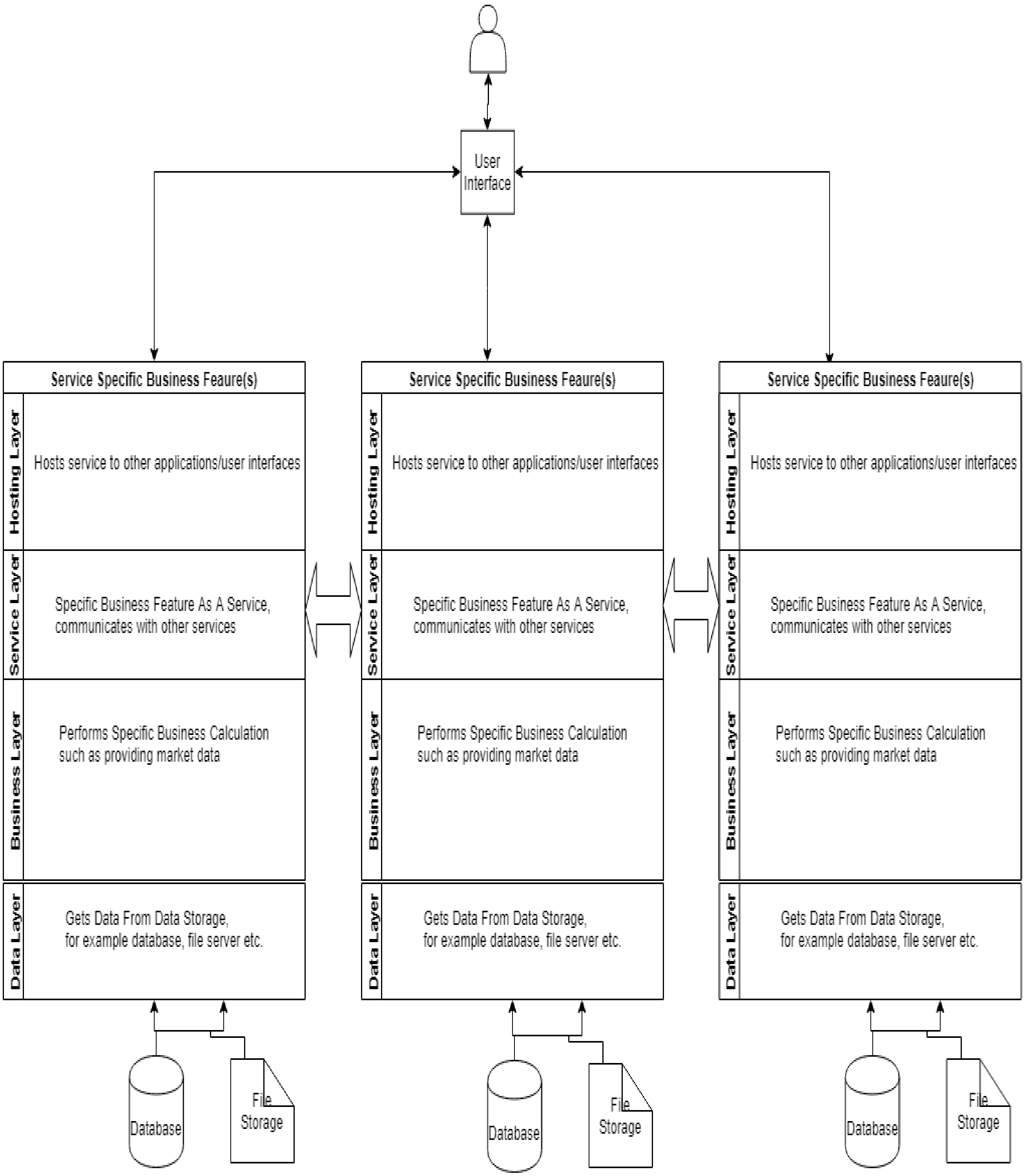


معماری میکروسرویس چگونه است؟

به طور خلاصه یک مفهوم وجود دارد و آن این است که ما همواره ملزم هستیم نرم‌افزار را نگهداری و به‌روزرسانی کنیم. باید فرایند بهبود اپلیکیشن‌ها را آسان‌تر و مقدار زمان مورد نیاز برای ارائه نسخه‌های جدید اپلیکیشن‌ها را کوتاه‌تر کنیم.

میکروسرویس‌ها برای حل مسائل اشاره شده فوق معرفی شده‌اند. معماری میکروسرویس یک بهینه‌سازی در زمینه معماری فوق‌الذکر محسوب می‌شود. در این معماری هر کارکرد تجاری به صورت یک سرویس ارائه می‌شود. هر سرویس می‌تواند به صورت مستقل از سرویس‌های دیگر میزبانی و توزیع شود.

معماری میکروسرویس

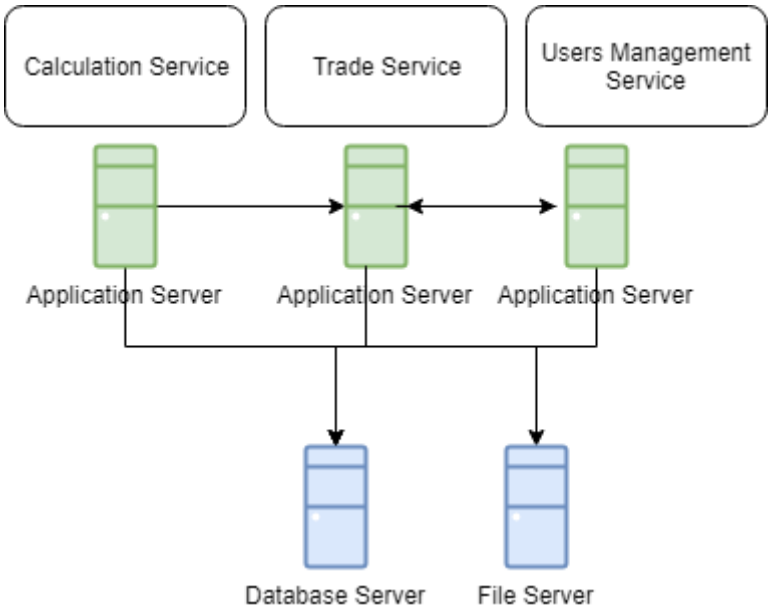


هر سرویس را می‌توان یک اپلیکیشن کوچک تصور کرد.

- همه سرویس‌ها می‌توانند حتی زمانی که سرویس‌ها روی ماشین‌های مختلف هستند، با همدیگر ارتباط داشته باشند. این وضعیت در ادامه امکان پیاده‌سازی کارکردهای جدید در سرویس‌ها را فراهم می‌سازد.
- میکروسرویس‌ها، سازمان‌ها را تشویق می‌کنند که از فرایند توزیع و تحویل پیوسته خودکار پیروی کنند.
- اپلیکیشن‌ها در نهایت بسیار پایدارتر می‌شوند، چون هر ویژگی می‌تواند به صورت مستقلانه تست و توزیع شود.
- از آنجا که هر سرویس روی پردازش مجزایی میزبانی می‌شود، اگر یک سرویس به نقطه تنگنای اپلیکیشن تبدیل شود و به منابع زیادی نیاز داشته باشد، در این صورت می‌توان آن را بدون هیچ گونه تأثیر سوء روی سرویس‌های دیگر، به ماشین دیگری انتقال داد.
- زمانی که کاربران بیشتری شروع به استفاده از یک ویژگی اپلیکیشن بکنند، آن سرویس می‌تواند با توزیع روی رایانه‌های قدرتمندتر یا از طریق استفاده از کش بدون این که روی سرویس‌های دیگر هیچ تأثیری بگذارد، مقیاس‌بندی شود.
- این معماری پایداری اپلیکیشن را نیز افزایش می‌دهد، چون هر سرویس می‌تواند به صورت مستقلانه ساخته، تست، توزیع و استفاده شود.
- کد اپلیکیشن می‌تواند به سادگی نگهداری شود و پردازش‌ها می‌توانند به صورت مجزا تحت نظارت قرار بگیرند.
- توسعه‌دهندگان اختصاصی می‌توانند سرویس‌ها را به صورت مستقل از هم پیاده‌سازی کرده و این سرویس‌ها را بدون تأثیرگذاری روی سرویس‌های دیگر انتشار دهند.
- بدین ترتیب نقطه شکست منفرد نیز از بین می‌رود، زیرا یک سرویس می‌تواند بدون تأثیرگذاری روی ویژگی‌های دیگری که اپلیکیشن نرم‌افزاری ارائه می‌کند، متوقف شود.
- در نتیجه این طراحی زمان مورد نیاز برای تحویل نسخه‌های جدید را کاهش می‌دهد و بنابراین هزینه را در طی زمان کاهش می‌دهد.
- قابلیت استفاده مجدد از کد افزایش می‌یابد، زیرا یک ویژگی به صورت سرویس میزبانی شده است و امکان استفاده چند سرویس از یک ویژگی به جای پیاده‌سازی مجدد کد در هر مورد وجود دارد.

- معماری مبتنی بر سرویس امکان استفاده از مجموعه متنوعی از فناوری برای رفع نیازها وجود دارد. به عنوان نمونه بسته‌های تحلیل داده زبان R یا پایتون می‌توانند به صورت مجزا توزیع و میزبانی شوند و همزمان می‌توان از C#.Net برای پیاده‌سازی سرویس‌ها استفاده کرد. به علاوه می‌توان از NodeJS در سمت سرور استفاده کرد و AngularJs و ReactJs نیز برای پیاده‌سازی رابط کاربری مورد استفاده قرار گیرند. هر ویژگی تجاری می‌تواند با استفاده از مجموعه مختلفی و از طریق تیم متفاوتی، مستقل از ویژگی‌های دیگر پیاده‌سازی شود.

در تصویر زیر نشان داده‌ایم که سرویس‌های متفاوت چگونه می‌توانند روی سرورهای مختلف اپلیکیشن توزیع شوند.



دروازه API می‌تواند به صورت نقطه ورودی برای کلاینت‌ها معرفی شود. این دروازه پاسخ را از چندین سرویس تجمیع کرده و بازگشت می‌دهد.

عیب معماری میکروسرویس چیست؟

معماری میکروسرویس نیز برای خود نقایصی دارد که در ادامه فهرستی از آن‌ها را ارائه کرده‌ایم.

۱. برای ساختن هر میکروسرویس به یک گردش کاری متفاوت برای ایجاد، توزیع و انتشار نیاز داریم. از این رو مطمئن شدن از این که گردش کاری توسعه خودکار است کاملاً حائز اهمیت است، چون در غیر این صورت، بار کاری تیم‌های عملیات IT افزایش می‌یابد.

۲. از آنجا که هر سرویس به صورت پردازش مجزایی پیاده‌سازی شده است، ابزارهای نظارت و نگهداری برای هر پردازش به صورت مستقل مورد نیاز هستند. این فرایند یک فرایند یک‌باره محسوب می‌شود؛ اما این گزارش‌ها و ابزارها نیز خود باید مورد نظارت قرار بگیرند. مجموعه ELS یا Prometheus به این منظور مناسب هستند.

۳. میکروسرویس‌ها بار کار را روی سیستم پیکربندی افزایش می‌دهد. پیکربندی سرویس‌های بیرونی معمولاً از طریق سرویس‌های مشترک صورت می‌گیرند و می‌توانند به وظیفه‌ای زمان‌بر تبدیل شوند.

۴. لغو یک وظیفه در حال اجرا زمانی که سرویس به صورت اپلیکیشن مستقلی توزیع یافته است، دشوارتر می‌شود.

۵. این طراحی می‌تواند روی عملکرد تأثیر منفی بگذارد، زیرا سربار شبکه برای هر فراخوانی به یک سرویس وجود دارد. در این موارد معمولاً از کَش کردن و همزمانی برای بهبود عملکرد استفاده می‌شود.

۶. میکروسرویس‌ها مشکلاتی در ارتباط با محاسبات توزیع یافته مانند امنیت، تراکنش‌ها، همزمانی و غیره ایجاد می‌کنند.

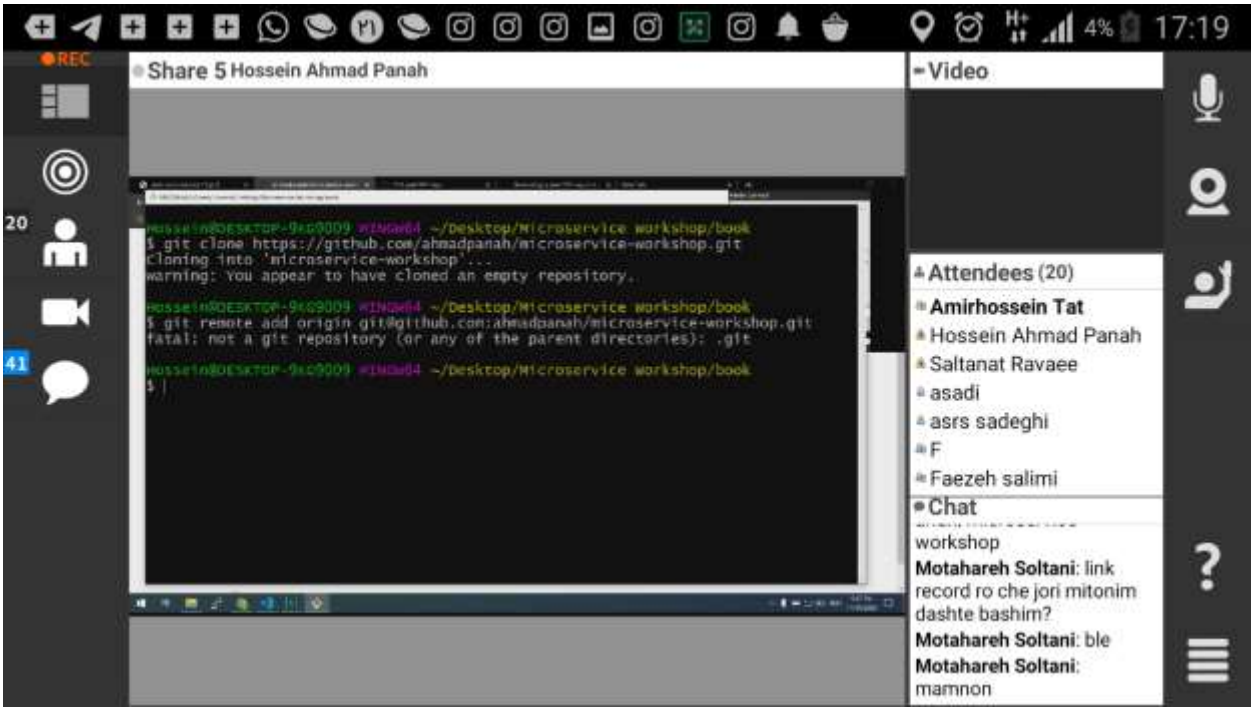
۷. تست‌های پایداری در سطح اپلیکیشن جهت تضمین این که ویژگی‌های جدید، کارکرد سرویس‌های موجود را از کار نمی‌اندازند، ضروری است.

۸. این معماری می‌تواند موجب افزایش کار مستندسازی برای هر سرویس بشود، چون هر سرویس نسخه، برنامه انتشار و چرخه انتشار خاص خود را دارد.

۹. زمانی که تعداد اپلیکیشن‌ها و کد آن افزایش می‌یابد، نگهداری و مدیریت صحیح آن به وظیفه دشواری تبدیل می‌شود.

کارگاه عملی

پس از آن استاد با تعریف پروژهای کار عملی نوشتن یک آپ کاربردی کتابخانه را به صورت مرحله به مرحله کد نویسی و اجرا نمودند .



درپایان سرکار خانم دکتر روایی از استاد محترم و مدیر محترم گروه و دانشجویان شرکت کننده قدردانی نمودند این برنامه در ساعت ۱۸پایان پذیرفت .