دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی

طراحی و نمونه سازی سامانه تشخیص بیماری و ارجاع به متخصص با استفاده از معماری میکروسرویس

نگارنده:

پویان حسابی

استاد راهنما:

دکتر امیر کلباسی

مهر ۱۴۰۲


بسم الله الرحمن الرحيم

به نام خدا

تعهدنامه اصالت اثر

تاریخ: ۱۴۰۱/۱۱/۱۵

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

اینجانب پویان حسابی متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

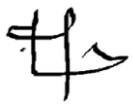
در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان‌نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است.

نقل مطالب با ذکر مآخذ بلامانع است.

پویان حسابی

امضا



سپاس‌گزاری

بدینوسیله از زحمات و تلاش بی دریغ استاد محترم جناب آقای دکتر امیر کلباسی که در تهیه این مجموعه با این جانب همکاری داشته و راهنمایی کرده‌اند، تشکر و مراتب سپاس قلبی خود را اعلام نموده و موفقیت ایشان را از خداوند متعال خواهانم.

همچنین از استاد گرانمایه، جناب آقای دکتر علیرضا باقری که زحمت داوری این پایان نامه را برعهده داشتند نهایت تشکر را دارم.

پویان حسابی

مهر ۱۴۰۲

چکیده

امروزه برنامه های کاربردی در زمینه پزشکی، متناسب با نیاز های کاربر پیشرفت کرده و قابلیت های آنها افزایش یافته است. تشخیص دقیق بیماری با روش های مبتنی بر هوش مصنوعی از مسائل به‌روز این سامانه‌ها است. به غیر از تشخیص بیماری، در بسیاری موارد بیمار نمی‌داند باید به چه پزشک با چه تخصصی مراجعه کند. این سردرگمی ممکن است منجر به مراجعه به متخصص نامربوط شود و زمان زیادی هدر رود.

همواره معماری نرم افزار جزو مسائل مهم در طراحی و توسعه بوده است. معماری میکروسرویس، به عنوان یک الگوی شاخص از معماری نرم افزار که با فلسفه جداسازی اجزای بزرگ سامانه به سرویس های کوچکتر به وجود آمده، راه حل های زیادی برای طراحی و پیاده سازی ارائه می‌دهد. در این پروژه، سامانه پزشکی تشخیص بیماری و ارجاع به متخصص با استفاده از معماری میکروسرویس طراحی و نمونه سازی شده است. سامانه به سرویس‌های کوچک و مستقل تجزیه شده به نوعی که مقیاس پذیر بوده و در آینده قابل توسعه و نگهداری باشد. در این پروژه کاربر می‌تواند با استفاده از رابط کاربری با برنامه تعامل کند و با اتصال به سرویس‌ها نیازمندی‌های خود را برطرف کند. سرویس‌های پیاده سازی شده عبارتند از: سرویس ورود کاربر، سرویس تشخیص بیماری، سرویس تشخیص تخصص از بیماری، سرویس جستجوی متخصص. هر کدام از سرویس‌ها قابلیت‌های مختلفی دارند که در ادامه به آنها پرداخته می‌شود.

در انتها با ابزارها و روش‌های ارزیابی و تطابق آنها با نیازمندی‌های سامانه، با تکیه بر اصول مهندسی نرم‌افزار، مشاهده می‌شود که معماری میکروسرویس و فناوری‌های استفاده شده گزینه مناسبی برای طراحی سامانه مورد نظر می‌باشد.

واژه‌های کلیدی:

معماری نرم افزار، معماری میکروسرویس، سامانه پزشکی، تشخیص بیماری

فهرست مطالب

فهرست شکل‌ها

فصل اول: مقدمه

در این بخش مقدمه پایان‌نامه ارائه می‌گردد. ابتدا مسأله شرح داده می‌شود، سپس ضرورت و اهداف و انگیزه پروژه قید می‌شود. در انتها ساختار گزارش و سرفصل های آینده توضیح داده می‌شود.

۱-۱- شرح مسأله

با توجه به روند رو به رشد فناوری و استفاده از سامانه‌های هوشمند در بخش پزشکی، طراحی و پیاده‌سازی یک سامانه هوشمند و خودکار برای تشخیص بیماری و ارجاع به متخصصین مربوطه، اهمیت بسیاری دارد. با استفاده از سامانه تشخیص بیماری و ارجاع به متخصص، بیماران می‌توانند به صورت سریع‌تر و دقیق‌تر روند تشخیص بیماری خود را طی کنند. همچنین، با استفاده از این سامانه، بیماران قادر به مشاوره با متخصصان مرتبط و دریافت نظرات آنان و خواندن مقالات علمی در مورد بیماری خود هستند. از دیگر مزایای این سامانه می‌توان به کاهش هزینه‌های درمانی و بهبود کیفیت خدمات پزشکی اشاره کرد. با تشخیص سریع‌تر و دقیق‌تر بیماری، احتمال درمان موفق‌تر بیشتر می‌شود و در نتیجه هزینه‌های درمانی کاهش می‌یابد. همچنین با استفاده از این سامانه، زمان بیشتری برای متخصصان پزشکی فراهم می‌شود تا بتوانند به بیماران خود بهترین خدمات را ارائه کنند.

امروزه با افزایش کاربران اینترنت، پروژه‌های نرم‌افزاری در مقیاس بزرگ ارائه می‌شوند. بزرگ بودن سامانه چالش هایی ایجاد می‌کند؛ برای مثال یک سرور با منابع محدود نمی‌تواند پاسخگوی تعداد درخواست بالا باشد. تاکنون راه حل های زیادی برای رفع این چالش ها ارائه شده است. یکی از مسائل مهم که باید در تولید این سامانه ها لحاظ کرد معماری نرم افزار[1] می‌باشد. معماری نرم افزار تاثیر بسزایی در طراحی سامانه دارد که در فصل های آینده به تفصیل در مورد آن صحبت می‌شود.

در این پروژه از معماری میکروسرویس[2] جهت طراحی و پیاده سازی سامانه استفاده می‌شود. در معماری میکروسرویس برنامه به چند سرویس مستقل، سبک و قابل مدیریت تقسیم می‌شود. این نوع سرویس‌ها به منظور مدیریت کردن یک وظیفه خاص طراحی می‌شوند. به طور مثال، یک سرویس صرفاً وظیفه مدیریت کاربران را دارا است و سرویس دیگر فقط برای بخش جستجوی سایت کاربرد دارد و با توجه به اینکه میکروسرویس‌ها مجزا و مستقل از یکدیگر هستند، به راحتی قادر خواهیم بود تا آن‌ها را با زبان‌های برنامه‌نویسی مختلفی نوشته و برای ذخیره‌سازی داده‌های مرتبط با آن‌ها نیز از سیستم‌های مدیریت پایگاه داده مختلفی استفاده کنیم. دلیل استفاده از این معماری برای سامانه، مزیت های میکروسرویس می‌باشد. همانطور که گفته شد سرویس ها مستقل هستند و یک وظیفه مشخص دارند برای همین پیاده سازی آن

[1] Software Architecture
[2] Microservice Architecture

راحت تر است و مفاهیمی مثل اصول سالید[1] در آن به آسانی تحقق می‌یابد، توضیحات بیشتر مربوط به میکروسرویس در فصل های ارائه می‌شود. همچنین برای پیاده سازی قسمت تشخیص بیماری ها و یافتن تخصص مربوطه از روش های یادگیری ماشین[2] با نظارت[3] و درخت تصمیم[4] استفاده می‌شود که در ادامه جزئیات آن توضیح داده می‌شود.

۱-۲- ضرورت و اهداف پروژه

سامانه های پزشکی زیادی جهت یافتن مراکز درمانی و متخصصین پزشکی، نوبت دهی اینترنتی، کسب اطلاعات پزشکی و مجله سلامت و ... وجود دارد. یکی از مشکلات آنها این است که فرض می‌شود کاربر عادی درک خوبی از انوع بیماری و ارتباط آن با تخصص های مختلف دارد. به طور مثال کسی که لرزش دست دارد نمی‌داند که به متخصص مغز و اعصاب، پزشک عمومی، یا متخصص غدد مراجعه کند، یا ممکن است کسی که ساعات زیادی را به صفحه نمایش کامپیوتر نگاه می‌کند و از آن سردرد می‌گیرد، منشا سردرد خود را نداند. از این رو باید زمان زیادی را جهت مطالعه در مورد آن یا مراجعه به پزشکی که آن تخصص را ندارد صرف کند. می‌دانیم با توجه به گستردگی بیماری ها و انواع مختلف علائم آنها نیاز است که کاربر درک خوبی از هر حوزه تخصص پزشکی داشته باشد که امروزه توقع آن زیاد بوده و بهتر است سامانه ای در راستای پیدا کردن بیماری و حوزه تخصص پزشکی جهت مراجعه طراحی شود.

از مزایا و اهمیت این پروژه و سامانه می‌توان به موارد زیر اشاره کرد:

۱- بهبود خدمات بهداشتی: توسعه این سامانه به بهبود و افزایش کیفیت خدمات بهداشتی و درمانی منجر خواهد شد. افراد به سرعت و با دقت بالا به تشخیص بیماری‌ها دسترسی خواهند داشت.

۲- کاهش هزینه‌های درمانی: با تشخیص زودرس بیماری‌ها و ارجاع به متخصصان در مراحل ابتدایی بیماری، هزینه‌های درمانی کاهش خواهد یافت. این سیستم کمک به بهبود مدیریت منابع درمانی و کاهش اصطکاک‌های مالی مرتبط با تأخیر در تشخیص خواهد کرد.

۳- زندگی سالم‌تر: این پروژه به افراد کمک می‌کند تا زودتر به تشخیص بیماری‌ها دسترسی پیدا کنند. این در نتیجه منجر به افزایش شانس‌های بهبود کامل و زندگی سالم‌تر خواهد شد.

۴- ارتقاء ارتباط بین متخصصین پزشکی: این سامانه به تعامل بیشتر بین متخصصان پزشکی و ارجاع دهندگان،

[1] SOLID: Single responsibility, Open-Closed, Liskov, Interface segregation, and Dependency Inversion
[2] Machine Learning
[3] Supervised Learning
[4] Decision Tree

پزشکان عمومی، کمک خواهد کرد. ارتباط سریع‌تر و دقیق‌تر بین این گروه‌ها، منجر به ارتقاء کیفیت مراقبت از بیماران خواهد شد.

۵- بهره‌وری افزایشی در سامانه پزشکی: توسعه این سامانه به بهره‌وری در سامانه پزشکی کمک خواهد کرد. با افزایش دقت تشخیص و کاهش تأخیر در ارجاع، منابع بهداشتی به بهترین نحو مدیریت می‌شوند.

۶- پیشرفت فناوری و نرم‌افزار: توسعه این پروژه منجر به پیشرفت تکنولوژی در حوزه بهداشت و نرم‌افزار خواهد شد. این سامانه به عنوان یک مثال نمونه برای بهره‌گیری از فناوری‌های مدرن در بهبود خدمات بهداشتی خواهد ایفا کرد.

۷- مساعدت به جوامع آسیب‌پذیر: این سامانه می‌تواند به جوامع آسیب‌پذیری که به سختی به خدمات بهداشتی دسترسی دارند کمک کند.

این اهداف و مزیت‌ها نشان می‌دهند که این پروژه نه تنها به بهبود خدمات بهداشتی و درمانی کمک می‌کند، بلکه نقش مهمی در بهره‌وری، کاهش هزینه‌ها، و بهبود کیفیت زندگی افراد ایفا می‌کند.

هدف از انجام این پروژه آن است که سامانه ای طراحی و پیاده سازی شود تا بیماران پس از اظهار علائم و نشانه های بیماری، نوع بیماری که به صورت تخمینی است و حوزه تخصص مربوطه را بیابند و متخصصانی جهت ویزیت به آنها معرفی شود. امروزه معماری نرم افزار اهمیتی زیادی پیدا کرده است، از آنجایی که نرم افزارها در حال تغییر هستند، معماری باید به گونه ای باشد که بتوانیم تغییرات را اعمال کرده و پروژه را توسعه دهیم.

۱-۳- ساختار گزارش

در فصل ابتدایی این گزارش، مقدمه‌ای بر روی مسأله مورد نظر این پروژه ارائه شد. در فصل دوم، مروری بر کارها و سامانه‌های پیشین با بررسی مزایا و معایب آنها انجام می‌گردد. در فصل سوم به معماری میکروسرویس و مفاهیم اولیه نظری در پروژه پرداخته می‌شود. در فصل چهارم درباره نیازمندی‌های سامانه توضیح داده می‌شود. در فصل پنجم با تکیه بر اصول مهندسی نرم افزار طراحی و نمونه سازی سامانه، به همراه ابزار های پیاده سازی به تفصیل توضیح داده می‌شود؛ پیاده سازی سرویس ها با عملکرد متفاوت به همراه نمودار های طراحی در این قسمت قرار دارد. در فصل ششم با استفاده از ابزار ها و فناوری های مختلف، سرویس ها و سامانه مورد نظر ارزیابی می‌گردد. در فصل هفتم، فصل انتهایی، نتایج حاصل از انجام این پروژه تحلیل شده و با ارائه پیشنهادها برای آینده، پایان‌نامه جمع‌بندی می‌گردد.

فصل دوم: مرور کارهای پیشین

در این قسمت سامانه های پیشین مورد بحث قرار گرفته و ارتباط این پروژه با آنها بررسی می‌گردد. لازم به ذکر است سامانه‌های مشابه با یکدیگر و همچنین با سامانه هدف مقایسه می‌شوند.

۲-۱- نگاه کلی به سامانه ها و مقالات پیشین

همانطور که در قسمت مقدمه گفته شد امروزه معماری در طراحی نرم افزار نقش بسزایی دارد از این رو مقالات علمی و کار های صنعتی زیادی در این حوزه انجام گردیده است. طی سالیان گذشته سامانه های متعددی از معماری یکپارچه به معماری میکروسرویس مهاجرت کرده اند. به طور مثال سامانه اسپاتیفای[1] که ماهیانه ۷۵ میلیون کاربر دارد، از این معماری استفاده کرده و در حال حاضر ۸۱۰ سرویس دارد. معماری میکروسرویس و هوش مصنوعی دو فناوری هستند که در سال‌های اخیر در حوزه بهداشت به شدت محبوب شده‌اند. معماری میکروسرویس چندین مزیت در حوزه پزشکی دارد که شامل مقیاس پذیری، انعطاف پذیری و پیمانه‌ای بودن[2] است؛ در حالی که هوش مصنوعی ظرفیت بزرگی در تشخیص بیماری را از طریق تجزیه و تحلیل داده ها و مدل‌سازی بر آنها دارد. در این بخش، فعالیت‌های مرتبط با معماری میکروسرویس و مزایای آن در حوزه پزشکی و تشخیص بیماری با استفاده از هوش مصنوعی و همچنین کارهای پیشین در استفاده از معماری میکروسرویس در سامانه‌های پزشکی و تشخیص بیماری، بررسی می‌شود.

چندین مطالعه بر روی استفاده از میکروسرویس در سامانه های پزشکی صورت گرفته است. به عنوان مثال، توسط پال[3] و جمشیدی [۱] یک معماری مبتنی بر میکروسرویس برای سامانه های بهداشتی پیشنهاد شد که برای مقیاس پذیری، پیمانه ای بودن و انعطاف پذیری طراحی شده است. این معماری با استفاده از یک سامانه نمونه برای نظارت بر بیماران، ارزیابی شد که نشان داد میکروسرویس ظرفیت مناسب در بهبود بهداشت را دارد. از طرفی در مقاله‌ای که توسط وانگ[4] و دیگران [۲] نوشته شده است، پیشنهاد شده که از فناوری اینترنت موبایل و سامانه میکروسرویس برای یکپارچه‌سازی داده‌های پزشکی بیماران استفاده شود و یک ساختار سلسله مراتبی از پلتفرم مدیریت پزشک-بیمار طراحی و پیاده‌سازی شود. با ساختاری که در این پلتفرم ایجاد شده، مشکل دریافت نکردن واحد های اولیه اطلاعاتی بهداشتی که منجر به درک ناکافی از بیماری و تاثیرات آن داشت، حل می‌شود. همینطور بیمارستان‌ها، پزشکان و بیماران به طور نزدیکی به هم متصل می‌شوند.

[1] Spotify
[2] Modularity
[3] Pahl
[4] Wang

۲-۲- مزایا و معایب سامانه های مشابه طراحی شده

پروژه ها و سامانه های پزشکی زیادی در داخل و خارج از کشور طراحی و پیاده سازی شده‌اند که با توجه به ویژگی ها، عملکرد، رابط کاربری و ... نقاطی مثبت و منفی دارند. در این قسمت قصد داریم با توجه به ویژگی ها و کاربرد ها، مثال های متنوعی بزنیم و نسبت به آن پروژه خود را طراحی کنیم.

سامانه های داخلی:

- مدیست [۳] : سامانه ای با ویژگی های مجله پزشکی، جستجوی پزشک و اطلاعات آن ها، دریافت مشاوره از پزشک مربوطه
- دکتر سایا [۴] : سامانه ای با ویژگی های مجله پزشکی، جستجوی پزشک و اطلاعات آن ها، دریافت مشاوره از پزشک مربوطه، نوبت دهی، اطلاعات مراکز درمانی، آزمایش در محل
- نوبت دات آی آر [۵] : سامانه ای با ویژگی های جستجوی پزشک و اطلاعات آن ها، نوبت دهی، اطلاعات مراکز درمانی، پرسش و پاسخ پزشکی

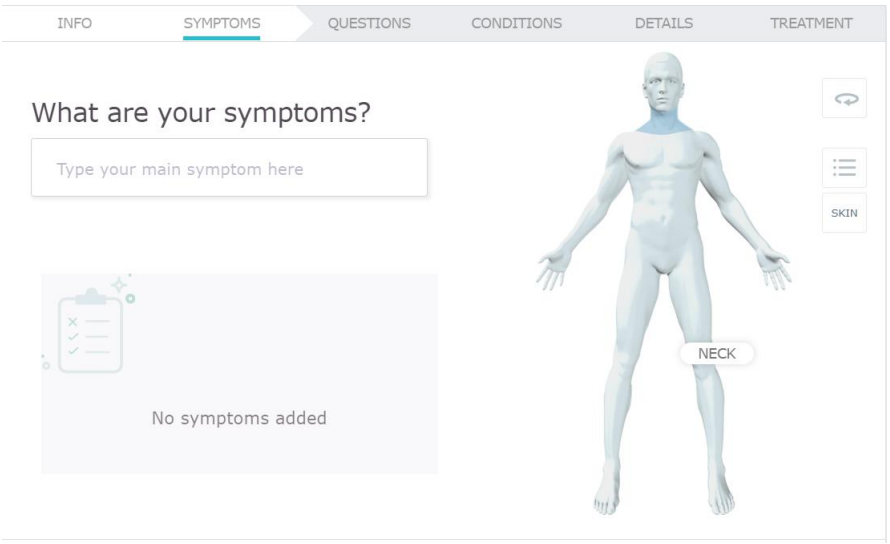
سامانه های خارجی:

- هات داک *(استرالیا)* [۶] : سامانه ای با ویژگی های مجله پزشکی، جستجوی پزشک و اطلاعات آن ها، نوبت دهی، اطلاعات مراکز درمانی، آزمایش در محل، برگزاری کارگاه های پزشکی به صورت برخط و غیر برخط
- مدسک *(انگلیس)* [۷] : سامانه ای با ویژگی های مجله پزشکی، جستجوی پزشک و اطلاعات آن ها، دریافت مشاوره از پزشک مربوطه، نوبت دهی، اطلاعات مراکز درمانی، آزمایش در محل، پرونده الکترونیک سلامت، پرسش و پاسخ، فعالیت بین‌المللی، پشتیبانی بعد از خدمات، تحلیل بر اساس گزارشات کاربر و پیشبینی بیماری
- وبامدی *(آمریکا)* [۸] : سامانه ای با ویژگی های مجله پزشکی، جستجوی پزشک و اطلاعات آن ها، اطلاعات مراکز درمانی، پرسش و پاسخ، تشخیص بیماری از علائم، اخبار، برگزاری کارگاه های پزشکی به صورت برخط و غیر برخط

همانطور که گفته شد سامانه های بسیار زیادی در حوزه خدمات پزشکی طراحی شده است، جامعیت پروژه های بالا از این جهت است که در هر کدام ویژگی هایی وجود دارد که در سامانه دیگر نیست، به طور مثال در بخش تشخیص بیماری سامانه وبامدی پیشتاز بوده و در سامانه های داخلی همچنین قابلیتی وجود ندارد، از طرفی در سامانه مدسک امکان تحلیل بیماری خود بر اساس گزارشات مختلف وجود دارد. در سامانه های داخلی نوبت دات آی آر اطلاعات بسیار جامعی از پزشکان و مراکز دارد و با توجه به تحقیقات میدانی، یکی از

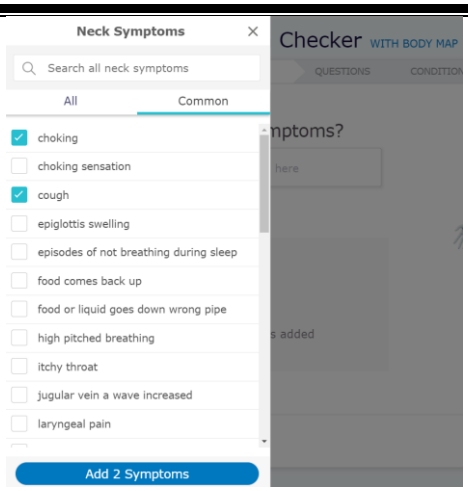
پر استفاده ترین سامانه نوبت دهی اینترنتی ایرانی می‌باشد و در برای کلینیک ها خیلی اهمیت دارد که تا حد امکان از یک سامانه تجمیع شده استفاده کنند، از طرفی در سامانه دکتر ساینا امکان دریافت خدمات آزمایش در محل سکونت می‌باشد. بنابراین نمی‌توان گفت که کدام سامانه از همه بهتر است، یا سامانه ای طراحی کرد که در عین حال ویژگی های مثبت همگی را داشته باشد.

از طرفی یکی از ویژگی هایی که حتی در سامانه های پزشکی خارجی کمتر دیده می‌شود، یافتن حوزه تخصصی بیماری می‌باشد. به طور مثال سامانه وب‌امدی بیماری را تشخیص داده و تخمینی می‌زند ولی اطلاع نمی‌دهد به طور خاص به چه متخصصی باید رجوع شود. سامانه وب‌امدی در قسمت تشخیص بیماری شباهت بسیار بالایی با سامانه هدف دارد. یکی از مزایای آن نسبت به سامانه ما تشخیص دقیق‌تر بیماری است. در این سامانه از یک مدل هوش مصنوعی استفاده شده که علاوه بر گرفتن علائم بیماری، شدت آنها را هم مورد ارزیابی قرار می‌دهد. شکل ۲-۱ نمایانگر مرحله از تشخیص بیماری می‌باشند؛ به طور مثال کاربر گردن را انتخاب کرده و علائم مربوط به آن نشان داده می‌شود.



شکل ۲-۱ مرحله اول تشخیص بیماری در سامانه وب‌امدی [۸]

پس از انتخاب کردن نقطه مورد نظر، علائم مربوط آن مشابه شکل ۲-۲ نمایش داده می‌شود. بعد از آن از کاربر کاربر پرسیده می‌شود که کدام علائم بیشترین تاثیر را دارد و مطابق شدت آن پرسیده می‌شود. از اصلی‌ترین مزایای این سامانه نسبت به سامانه هدف، در نظر گرفتن شدت علائم و تشخیص دقیق‌تر بیماری می‌باشد.



شکل ۲-۲ انتخاب علائم بیماری در سامانه وب‌امدی[۸]

هر کدام از سامانه‌های گفته شده مزایا و معایب مختلفی نسبت به یکدیگر دارند که بررسی دقیق آن خارج از مطالب این پایان‌نامه است. سامانه ایده‌آل سامانه‌ای است که در صورت امکان مزایا سامانه‌های مختلف را بدون داشتن معایب آنها داشته باشد و در نهایت بهترین خدمات را به کاربر بدهد.

با توجه به اینکه سامانه های بالا متن باز نیستند و قابلیت دسترسی به معماری و طراحی آنها نیست، امکان مقایسه معماری آنها وجود ندارد ولی در این پروژه سعی می‌شود با تمرکز بر معماری نرم افزار، مزایا و معایب سامانه بررسی شود.

۲-۳- جمع‌بندی کارهای پیشین

بنظر می‌رسد در سامانه های داخلی سامانه معروفی جهت تشخیص بیماری وجود ندارد برای همین نقطه قوت این پروژه این ویژگی می‌باشد که اگر بیمار مورد نظر حوزه تخصصی که باید به پزشک مربوط به آن مراجعه کند را نداند، امکان تشخیص رایگان و برخط آن وجود دارد. سامانه نهایی بیشترین شباهت را به سامانه وب‌امدی دارد و حتی در پیاده سازی قسمت تشخیص بیماری از آن الگو گرفته شده است. در این پروژه قصد داریم از نقاط مثبت و منفی هر سامانه بهره ببریم و با تمرکز بر معماری میکروسرویس سامانه خود را پیاده سازی کنیم.

فصل سوم: معماری میکروسرویس

برای پرداختن به معماری میکروسرویس، ابتدا نیاز است با معماری نرم‌افزار آشنا شویم. در ادامه به ویژگی‌های معماری دسته بندی‌های آن نظیر سبک و الگو معماری آشنا شده و سپس به تفصیل به معماری میکروسرویس پرداخته می‌شود. سپس به الگوهای طراحی پرکاربرد در معماری میکروسرویس نظیر پایگاه داده اشتراکی، ساگا و سرویس دیسکاوری پرداخته می‌شود. در انتها مقایسه‌ای بین این سبک معماری و دیگر معماری‌های نرم‌افزار صورت می‌گیرد.

۳-۱- معماری نرم افزار

معماری نرم افزار در یک سیستم، نشان‌دهنده تصمیمات طراحی ساختار و رفتار کلی در آن سیستم است. یک سیستم مجموعه‌ای از مولفه‌ها است که هر یک کارکرد یا کارکردهای خاصی را انجام می‌دهند. به عبارت دیگر، معماری نرم‌افزار پایه محکمی را فراهم می‌کند که بر اساس آن می‌توان نرم‌افزار را توسعه داد. معماری نرم افزار تعیین‌کننده کیفیت، عملکرد، قابلیت نگهداری و موفقیت کلی سیستم است. چندین الگو و اصول معماری سطح بالا وجود دارند که معمولاً در سیستم‌های مدرن استفاده می‌شوند. این الگوها اغلب به عنوان سبک‌های معماری شناخته می‌شوند. معماری یک سیستم نرم افزاری معمولا محدود به یک سبک معماری واحد نیست، بلکه ترکیبی از سبک‌هایی است که اغلب در سیستم مورد استفاده قرار می‌گیرد. به بیانی دیگر، معماری نرم افزار به ساختارهای اساسی یک سیستم نرم افزاری و نظم و انضباط ایجاد چنین ساختارهایی اشاره دارد. هر ساختار شامل عناصر نرم افزار، روابط بین آنها و ویژگی‌های آن عناصر و آن روابط است.

در ادامه در مورد مفاهیم مهم معماری نرم افزار اعم از مدل کلاینت-سرور[1]، کاربرد ای‌پی‌آی [2]، رابط برنامه نویسی کاربردی، در معماری نرم افزار، پیمانه‌بندی در توسعه کد بحث می‌شود.

۳-۱-۱- مدل کلاینت سرور

کلاینت-سرور یک ساختار کاربردی توزیع شده است که وظایف یا حجم کاری را بین ارائه‌دهندگان یک منبع یا سرویس، به نام سرور، و درخواست‌کنندگان خدمات، به نام کلاینت، تقسیم می‌کند. به زبان ساده، کلاینت یک برنامه کاربردی است که نوعی اطلاعات را درخواست می‌کند یا اقداماتی را انجام می‌دهد و سرور برنامه‌ای است که اطلاعات را ارسال می‌کند یا مطابق با فعالیت کاربر، اقداماتی را انجام می‌دهد. کلاینت‌ها معمولاً توسط برنامه‌های فرانت‌اند[3] که روی وب یا برنامه‌های تلفن همراه اجرا می‌شوند، ارائه می‌شوند.

[1] Client-Server
[2] API: Application Programing Interface
[3] Frontend

۳-۱-۲- کاربرد ای‌پی‌آی

همان‌طور که پیشتر اشاره شد کلاینت‌ها و سرور ها موجودیت هایی هستند که برای ارسال درخواست و ارسال پاسخ با یکدیگر ارتباط برقرار می‌کنند. ای‌پی‌آی یا رابط برنامه نویسی برنامه روشی است که این دو بخش برای برقراری ارتباط استفاده می‌کنند. ای‌پی‌آی مجموعه‌ای از قوانین تعریف شده به حساب می آید که نحوه برقراری ارتباط برنامه‌ای را با برنامه دیگر مشخص می‌کند. در واقع ای‌پی‌آی قراردادی بین کلاینت و سرور است که مثلا بیان می‌کند: «اگر A را بفرستید، پاسخ من همیشه B خواهد بود. اگر C را بفرستید، پاسخ من همیشه D خواهد بود» و تا انتها به همین شکل ادامه پیدا می‌کند.

یکی دیگر از مفاهیم مهم در حوزه ای‌پی‌آی، درگاه ای‌پی‌آی[1] است. این مفهوم به یک نقطه ورودی مشترک برای دسترسی به سرویس‌ها و منابع مختلف در یک سیستم نرم‌افزاری اشاره دارد و وظیفه مدیریت، کنترل امنیت، تغییر مسیردهی[2] و ترتیب دسترسی به این منابع را بر عهده دارد. درگاه ای‌پی‌آی به عنوان یک لایه واسط بین مشتریان، معمولاً برنامه‌های کاربردی یا دستگاه‌های مختلف، و سرویس‌ها یا منابع در داخل یک سیستم یا در محیط‌های متعددی که می‌توانند مختلف باشند، عمل می‌کند.

۳-۱-۳- پیمانه ای بودن در معماری

پیمانه ای بودن نرم افزار، تجزیه نرم افزار به بخش‌های کوچک‌تر با رابط‌های استاندارد است. ما می‌خواهیم محصولاتی با قطعه کد های قابل استفاده مجدد ایجاد کنیم، بنابراین فقط یک‌بار یک قابلیت عملکردی را پیاده‌سازی کرده و سپس از آن مکرراً استفاده می‌کنیم. پیمانه‌ای بودن برای ساده‌سازی برنامه‌های کاربردی و کدهای پایه‌ای بزرگ انجام می‌شود و دارای مزایایی است که در ادامه به برخی از آن‌ها اشاره شده است.

- پیمانه‌بندی کمک می‌کند تا قابلیت‌های عملکردی تقسیم‌بندی شوند و در نتیجه بصری‌سازی، درک و سازماندهی پروژه تسهیل خواهد شد.
- هنگامی که پروژه به‌صورت شفاف سازمان‌دهی و تقسیم‌بندی‌شده باشد، نگهداری آن آسان‌تر است و خطاها و باگ های کمتری تجربه خواهد شد.
- اگر پروژه شما به بخش‌های مختلفی تقسیم شده باشد، می‌توان هر بخش را به طور مستقل توسعه داد و اصلاحات لازم را پیاده‌سازی کرد که اغلب بسیار مفید است.

۳-۲- ویژگی‌های معماری

یک شرکت تصمیم می گیرد یک مشکل خاص را با استفاده از نرم افزار حل کند، بنابراین فهرستی از نیازمندی‌های آن سیستم را جمع آوری می کند. طیف گسترده ای از تکنیک ها برای تمرین جمع آوری نیازمندی ها وجود دارد که عموماً توسط فرآیند توسعه نرم افزار مورد استفاده تیم تعریف می شود. اما معمار

[1] API Gateway
[2] Routing

باید بسیاری از عوامل دیگر را در طراحی یک راه حل نرم افزاری در نظر بگیرد. معماران ممکن است در تعریف دامنه یا الزامات تجاری با یکدیگر همکاری کنند، اما یک مسئولیت کلیدی مستلزم تعریف، کشف، و در غیر این صورت تجزیه و تحلیل تمام کارهایی است که نرم افزار باید انجام دهد که مستقیماً به عملکرد دامنه مربوط نمی شود، ویژگی های معماری می‌باشد. [۹]

ویژگی‌های معماری در طول طیف گسترده‌ای از سیستم نرم‌افزاری وجود دارد، از ویژگی‌های کد سطح پایین، مانند پیمانه‌ای بودن، تا نگرانی‌های عملیاتی پیچیده، مانند مقیاس‌پذیری و کشش. هیچ استاندارد جهانی واقعی علیرغم تلاش برای تدوین استانداردهای گذشته وجود ندارد. در عوض، هر سازمان تفسیر خود را از این اصطلاحات ایجاد می کند. علاوه بر این، از آنجایی که اکوسیستم نرم افزاری بسیار سریع تغییر می کند، مفاهیم، اصطلاحات، معیارها و تأییدهای جدید دائماً ظاهر می شوند و فرصت های جدیدی را برای تعاریف ویژگی های معماری فراهم می کنند.

۳-۲-۱- استانداردهای ویژگی‌های معماری

لیست کاملی از استانداردها وجود ندارد. سازمان بین‌المللی استانداردها (ISO) فهرستی را منتشر می‌کند که بر اساس قابلیت‌ها سازمان‌دهی شده‌اند، با بسیاری از مواردی که ما فهرست کرده‌ایم همپوشانی دارند، اما عمدتاً یک فهرست دسته‌بندی ناقص ایجاد می‌کند. برخی از تعاریف ISO در زیر آمده است:

۱- بهره وری عملکرد:

اندازه گیری عملکرد نسبت به مقدار منابع مورد استفاده در شرایط شناخته شده. این شامل رفتار زمانی، اندازه گیری پاسخ، زمان پردازش و/یا نرخ توان عملیاتی، استفاده از منابع، مقدار و انواع منابع استفاده شده، و ظرفیت، درجه ای که از حداکثر محدودیت های تعیین شده فراتر رفته است، می شود.

۲- سازگاری:

درجه ای که یک محصول، سیستم یا مؤلفه می تواند اطلاعات را با سایر محصولات، سیستم ها یا مؤلفه ها مبادله کند و/یا عملکردهای مورد نیاز خود را در حالی که محیط سخت افزاری یا نرم افزاری را به اشتراک می گذارد انجام دهد. این شامل همزیستی (می تواند عملکردهای مورد نیاز خود را به طور موثر انجام دهد در حالی که یک محیط و منابع مشترک را با سایر محصولات به اشتراک می گذارد و قابلیت همکاری درجه ای که دو یا چند سیستم می توانند اطلاعات را مبادله و استفاده کنند.

۳- قابلیت استفاده بودن:

کاربران می توانند از سیستم به طور موثر، کارآمد و رضایت بخش برای هدف مورد نظر خود استفاده کنند. این شامل تشخیص مناسب بودن، کاربران می توانند تشخیص دهند که آیا نرم افزار برای نیازهای آنها مناسب است؛ یادگیری (چقدر آسان کاربران می توانند نحوه استفاده از نرم افزار را یاد بگیرند)، محافظت از خطای

کاربر (محافظت در برابر خطاهای کاربران) و دسترسی (در دسترس قرار دادن نرم افزار).

۴- قابلیت اطمینان:

درجه ای که یک سیستم تحت شرایط مشخص برای مدت زمان مشخصی کار می کند. این ویژگی شامل زیرمجموعه‌هایی مانند بلوغ (آیا نرم‌افزار نیازهای قابلیت اطمینان را در حالت عادی برآورده می‌کند)، در دسترس بودن (نرم‌افزار عملیاتی و قابل دسترسی است)، تحمل خطا (آیا نرم‌افزار علی‌رغم ایرادات سخت‌افزاری یا نرم‌افزاری طبق برنامه عمل می‌کند)، و قابلیت بازیابی (می‌تواند نرم‌افزار با بازیابی هر گونه داده آسیب دیده از شکست بازیابی می کند و وضعیت مطلوب سیستم را دوباره برقرار می کند.

۵- امنیت:

درجه ای که نرم افزار از اطلاعات و داده ها محافظت می کند به طوری که افراد یا سایر محصولات یا سیستم ها از درجه دسترسی به داده ها متناسب با انواع و سطوح مجوز آنها برخوردار باشند. این خانواده از ویژگی ها شامل محرمانه بودن (داده ها فقط برای افرادی که مجاز به دسترسی هستند قابل دسترسی است)، یکپارچگی (نرم افزار از دسترسی غیرمجاز یا تغییر نرم افزار یا داده ها جلوگیری می کند)، عدم انکار، (آیا می توان اعمال یا رویدادهایی را ثابت کرد که رخ داده اند)، پاسخگویی (آیا می توان اقدامات کاربر یک کاربر را ردیابی کرد)، و اصالت (اثبات هویت کاربر).

۶- قابلیت نگهداری:

نشان‌دهنده میزان اثربخشی و کارایی است که توسعه‌دهندگان می‌توانند نرم‌افزار را برای بهبود، تصحیح یا تطبیق آن با تغییرات محیطی و/یا الزامات تغییر دهند. این مشخصه شامل مدولار بودن (درجه ای که نرم افزار از اجزای مجزا تشکیل شده است)، قابلیت استفاده مجدد (درجه ای که توسعه دهندگان می توانند از یک دارایی در بیش از یک سیستم یا در ساختن دارایی های دیگر استفاده کنند)، قابلیت تجزیه و تحلیل (توسعه دهندگان به راحتی می توانند معیارهای مشخصی را در مورد نرم‌افزار)، قابلیت تغییر (درجه‌ای که توسعه‌دهندگان می‌توانند نرم‌افزار را بدون ایجاد نقص یا کاهش کیفیت محصول موجود تغییر دهند)، و آزمایش‌پذیری (به راحتی توسعه‌دهندگان و دیگران می‌توانند نرم‌افزار را آزمایش کنند).

۷- قابل حمل بودن:

درجه ای که توسعه دهندگان می توانند یک سیستم، محصول یا جزء را از یک سخت افزار، نرم افزار، یا سایر محیط های عملیاتی یا کاربری به دیگری منتقل کنند. این ویژگی شامل ویژگی‌های فرعی سازگاری (آیا توسعه‌دهندگان می‌توانند به طور موثر و کارآمد نرم‌افزار را برای سخت‌افزار، نرم‌افزار، یا سایر محیط‌های عملیاتی یا کاربری متفاوت یا در حال تکامل تطبیق دهند)، قابلیت نصب (آیا نرم‌افزار را می‌توان در یک محیط مشخص نصب و/یا حذف نصب کرد)، و قابلیت تعویض (به راحتی توسعه دهندگان می توانند عملکرد را با نرم افزارهای دیگر جایگزین کنند).

آخرین مورد در لیست ISO به جنبه های کاربردی نرم افزار می پردازد، که ما معتقد نیستیم به این لیست تعلق دارد:

۸- تناسب عملکردی:

این مشخصه نشان دهنده درجه ای است که یک محصول یا سیستم عملکردهایی را ارائه می دهد که نیازهای اعلام شده و ضمنی را هنگام استفاده در شرایط خاص برآورده می کند. این ویژگی از زیر ویژگی های زیر تشکیل شده است:

۹- کامل بودن عملکردی:

درجه ای که مجموعه توابع تمام وظایف مشخص شده و اهداف کاربر را پوشش می دهد.

۱۰- صحت عملکردی:

درجه ای که یک محصول یا سیستم نتایج صحیح را با درجه دقت مورد نیاز ارائه می دهد.

۳-۲-۳- رعایت تعادل ویژگی‌ها در معماری

برنامه‌ها تنها می‌توانند از تعدادی از ویژگی‌های معماری که ما فهرست کرده‌ایم، به دلایل مختلف پشتیبانی کنند. اول، هر یک از ویژگی های پشتیبانی شده نیاز به تلاش طراحی و شاید پشتیبانی ساختاری دارد. دوم، مشکل بزرگتر در این واقعیت نهفته است که هر ویژگی معماری اغلب بر دیگران تأثیر می گذارد. به عنوان مثال، اگر یک معمار بخواهد امنیت را بهبود بخشد، تقریباً به طور قطع بر عملکرد تأثیر منفی خواهد گذاشت: برنامه باید رمزگذاری در حین پرواز، غیرمستقیم برای پنهان کردن اسرار و سایر فعالیت هایی که به طور بالقوه عملکرد را کاهش می دهند، انجام دهد.

یک استعاره به نشان دادن این ارتباط متقابل کمک خواهد کرد. ظاهراً خلبانان اغلب در یادگیری پرواز با هلیکوپتر دچار مشکل می شوند، زیرا به کنترل هر دست و هر پا نیاز دارد و تغییر یکی روی دیگران تأثیر می گذارد. بنابراین، پرواز با هلیکوپتر یک تمرین متعادل کننده است که به خوبی فرآیند مبادله را هنگام انتخاب ویژگی های معماری توصیف می کند. هر ویژگی معماری که یک معمار از آن پشتیبانی می کند، به طور بالقوه طراحی کلی را پیچیده می کند. بنابراین، معماران به ندرت با موقعیتی مواجه می شوند که بتوانند یک سیستم را طراحی کنند و هر ویژگی معماری را به حداکثر برسانند. اغلب، تصمیمات به مبادله بین چندین نگرانی رقیب منجر می شود.

بسیاری از ویژگی‌های معماری منجر به راه‌حل‌های عمومی می‌شود که در تلاش برای حل هر مشکل تجاری هستند، و این معماری‌ها به ندرت کار می‌کنند زیرا طراحی ناکارآمد می‌شود.

این نشان می دهد که معماران باید تلاش کنند تا معماری را تا حد امکان تکرار شونده طراحی کنند. اگر بتوانید آسان‌تر تغییراتی در معماری ایجاد کنید، می‌توانید کمتر در مورد کشف درستی در اولین تلاش

استرس داشته باشید. یکی از مهمترین درس های توسعه نرم افزار چابک[1] ارزش تکرار است. این در تمام سطوح توسعه نرم افزار از جمله معماری صادق است.

۳-۳- سبک[2] ها و الگو[3] های معماری نرم افزار

در این قسمت می خواهیم مفهوم مهمی که دسته‌بندی‌های مختلف معماری نرم‌افزار را تحت تاثیر قرار می‌دهد، بررسی کنیم و به سبک معماری، الگوی معماری و الگوی طراحی بپردازیم. در واژه‌شناسی توسعه نرم افزار این سه مفهوم شفاف نیستند و گاهی افراد مختلف نظرات متفاوتی دارند. که در ادامه به بررسی هر یک به صورت جداگانه خواهیم پرداخت.

۳-۳-۱- سبک معماری

سبک معماری نرم‌افزار به مجموعه‌ای از الگوها، قواعد و مبانی گفته شده که برای طراحی سیستم‌های نرم‌افزاری استفاده می‌شود. سبک معماری نرم‌افزار برای تعیین ساختار و تنظیم روابط بین اجزای سیستم مورد استفاده قرار می‌گیرد. در ادامه چند سبک متداول بررسی می‌شود.

- سبک لایه‌ای[4]: سبک لایه‌ای مناسب برنامه‌هایی است که شامل گروه‌هایی از وظایف فرعی هستند و با ترتیب خاصی اجرا می‌شوند، مثلاً وب اپلیکیشن‌های[5] تجارت الکترونیکی را می‌توان مثال زد. الگوی لایه‌ای، توسعه سریع برنامه‌ها را آسان می‌کند، اما نقطه ضعف آن این است که بعداً تقسیم لایه‌ها دشوار خواهد بود.

- سبک لوله فیلتر[6]: از این سبک معماری می‌توان برای ساخت سیستم‌هایی استفاده کرد که وظیفهٔ تولید و پردازش جریان داده‌ها را بر عهده دارند؛ به طوری که هر مرحله پردازش روی داده‌های ورودی و خروجی در کامپوننت فیلتر انجام می‌شود و داده‌هایی که بایستی مورد پردازش قرار گیرند نیز از طریق لوله ها و از یک کامپوننت به کامپوننت بعدی منتقل می‌شوند. همچنین کامپوننت لوله را می‌توان برای کار هایی همچون بافر[7] کردن داده‌ها یا برای هماهنگ‌سازی کامپوننت‌ها با یکدیگر نیز مورد استفاده قرار داد.

- سبک سرویس‌گرا[8]: معماری سرویس گرا یا SOA، یک مدل معماری برای طراحی سیستم‌های نرم‌افزاری است که در آن سرویس‌ها به عنوان واحدهای اصلی طراحی و پیاده‌سازی استفاده می‌شوند. در این مدل، هر سرویس به صورت مستقل عمل می‌کند و می‌تواند به صورت مجزا توسعه یابد و به سرویس های دیگری متصل شود و در سیستم‌های مختلف استفاده شود. در معماری سرویس گرا، هر سرویس، یک واسط مشخص دارد

[1] Agile
[2] Software Architecture Styles
[3] Software Architecture Pattern
[4] Layered
[5] Web Applications
[6] Pipe-filter
[7] Buffer
[8] Service Oriented Architecture

که از آن می‌توان برای دسترسی به توابع استفاده کرد. سرویس‌ها می‌توانند به صورت مستقیم با یکدیگر ارتباط برقرار کنند یا از طریق یک پایگاه داده مشترک به اشتراک گذاشته شوند. از مزایای معماری سرویس گرا می‌توان به افزایش قابلیت اطمینان، افزایش قابلیت توسعه، افزایش سازگاری و افزایش قابلیت استفاده مجدد سرویس‌ها اشاره کرد. همچنین، استفاده از این مدل معماری، امکان ارتقاء و بهبود قابلیت‌های سرویس‌ها را نیز فراهم می‌کند.

۳-۳-۲- الگوی معماری

الگوی معماری یک راه‌حل جامع با قابلیت استفادهٔ مجدد برای مسائل رایج در معماری نرم‌افزار در یک حوزهٔ خاص است. به عبارت دیگر، الگوهای معماری می‌توانند یک راه‌کار معماری پیشنهاد دهند که به عنوان مبنایی برای طراحی معماری نرم‌افزار در محدودهٔ وسیع‌تر عمل کند. الگوهای معماری یک پاسخ تکراری به یک مشکل تکراری هستند و در واقع الگو های معماری برطرف کننده مشکلات مربوط به سبک های طراحی هستند. برای مثال با الگوی معماری تعیین می شود که چه کلاس هایی داشته باشیم و این کلاس ها چگونه با یکدیگر در لایه ها ارتباط داشته باشند یا در یک معماری سرویس‌گرا چه ماژول هایی داشته باشیم و چگونه با یکدیگر تبادل اطلاعات خواهند کرد. در واقع سبک معماری در بالاترین سطح انتزاع قرار دارد و الگوی معماری راهی برای پیاده سازی سبک معماری می‌باشد. در ادامه چند الگو متداول بررسی می‌شود.

- الگو n-tier: معمول ترین الگوی معماری نرم افزار، معماری لایه‌ای است که یا به عنوان معماری n-tier نیز شناخته می‌شود. این نوع معماری نرم افزار به طور گسترده‌ای توسط بیشتر معماران، طراحان و توسعه دهندگان نرم‌افزار شناخته شده است. اگرچه از نظر تعداد و نوع لایه‌هایی که باید وجود داشته باشد محدودیت خاصی وجود ندارد، اما در اکثر مواقع معماری لایه‌ای شامل سه طبقه: نمایش(رابط کاربری)، بیزینس و داده می‌باشد.

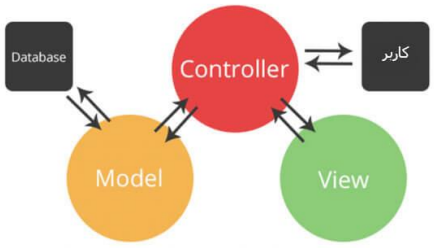
- الگو ام‌وی‌سی[1]: شکل ۳-۱ نمای کلی از این معماری را نشان می‌دهد. این الگو معماری برنامه‌های تعاملی را به سه قسمت تقسیم می‌کند که عبارتند از:

- مدل: شامل قابلیت‌های اصلی برنامه و داده‌ها است (در واقع، مدل مسئول کاری است که برنامه به خاطر آن توسعه یافته است.)
- نما: مسئولیت نمایش داده مورد نیاز و رابط کاربری برنامه به کاربر را بر عهده دارد (در برخی برنامه‌ها ممکن بیش از یک مورد برای کاربران تعریف شود.)
- کنترل گر: مسئولیت مدیریت کردن داده‌های ورودی کاربران و برقراری ارتباط مابین مدل و نما را بر عهده دارد.

الگوی معماری ام‌وی‌سی موجب جداسازی اجزا فوق در یک برنامه می‌شود؛ به عبارت دیگر، این الگوی معماری موجب جدا شدن روش‌های به‌کارگیری داده در داخل برنامه از روش‌های ارائه داده و دریافت آن‌ها از کاربران می‌شود و همین مسئله نیز موجب کاهش پیچیدگی و سهولت توسعه برنامه خواهد شد و این امکان را برای

[1] MVC: Model View Controller

توسعه دهندگان فراهم می‌آورد تا بتوانند به شیوه‌ای مؤثر، از کد منبع برنامه استفادهٔ مجدد داشته باشند



شکل ۳-۱ نمایی از معماری ام وی سی [۱۰]

.

۳-۴- معماری یکپارچه[1]

معماری یکپارچه یک نوع معماری نرم‌افزاری است که در آن تمامی اجزاء و قسمت‌های یک برنامه در یک واحد بزرگ و تکلیف‌محور متمرکز شده‌اند. در این معماری، همه اجزاء از جمله واسط کاربری، منطق کار و پایگاه داده در یک برنامه ترکیب شده‌اند و به صورت یک بسته واحد توسعه، تست و استقرار می‌شوند. در معماری یکپارچه، تمامی قسمت‌های برنامه در یک فرآیندهای تکی تعبیه شده‌اند و ارتباط بین آن‌ها به وسیله تماس تابعی، فراخوانی روش و به‌کارگیری کتابخانه‌های داخلی برقرار می‌شود.

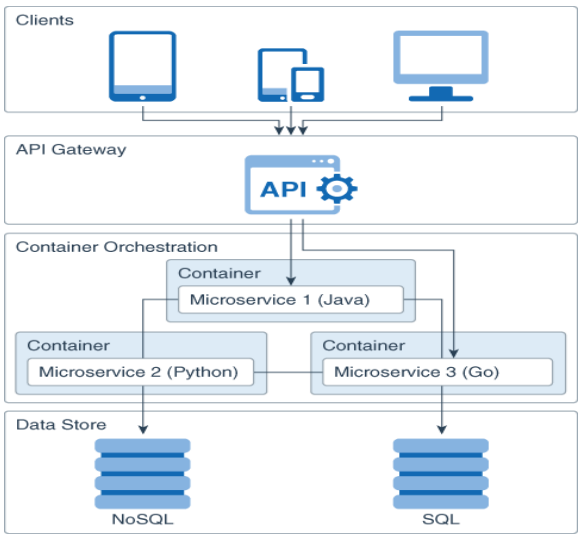
۳-۵- معماری میکروسرویس

معماری میکروسرویس یک الگوی معماری نرم‌افزاری است که در آن برنامه‌های کاربردی را از طریق تقسیم‌بندی به قطعات کوچکتر و مستقل به نام میکروسرویس‌ها، سازماندهی می‌کند. در این معماری، هر میکروسرویس مستقل از سایر میکروسرویس‌ها است و می‌تواند به صورت مستقل توسعه، استقرار و مقیاس‌پذیری داشته باشد.

میکروسرویس‌ها بر مبنای معماری مبتنی بر سرویس ساخته شده‌اند. معماری مبتنی بر سرویس، که در بخش های قبل توضیح داده شد، به برنامه‌ها امکان ارتباط با یکدیگر روی یک رایانه منفرد و یا در زمان توزیع برنامه‌ها روی چندین رایانه در یک شبکه را ارائه می‌کند. هر میکروسرویس ارتباط اندکی با سرویس‌های دیگر

[1] Monolithic

دارد. شکل ۳-۲ نمای کلی از این معماری را نشان می‌دهد.



شکل ۳-۲ نمایی از معماری میکروسرویس [۱۰]

این سرویس‌ها خودکفا هستند و یک کارکرد منفرد (یا گروهی از کارکردهای مشترک) را ارائه می‌کنند. معماری میکروسرویس‌ها به طور طبیعی در سازمان‌های بزرگ و پیچیده استفاده می‌شود که در آن‌ها چند تیم توسعه می‌توانند مستقل از هم برای ارائه یک کارکرد تجاری کار بکنند و یا برنامه‌ها ملزم به ارائه خدمات به یک حوزه تجاری باشند.

ویژگی‌های کلیدی معماری میکروسرویس عبارتند از:

تجزیه‌پذیری[1]: برنامه‌ها در معماری میکروسرویس به قطعات کوچکتر تقسیم می‌شوند که به صورت مستقل قابل توسعه و استقرار هستند. این تقسیم‌بندی منجر به کاهش پیچیدگی و افزایش انعطاف‌پذیری می‌شود.

استقلال: هر میکروسرویس مستقل از دیگران است و می‌تواند با استفاده از زبان‌ها، فناوری‌ها و ابزارهای مختلف پیاده‌سازی شود. این استقلال به تیم‌ها اجازه می‌دهد که به صورت مستقل بر روی میکروسرویس‌های خود کار کنند.

ارتباطات محدود[2]: هر میکروسرویس مسئولیت‌های محدودی را انجام می‌دهد و تنها از طریق رابط‌های خارجی با سایر میکروسرویس‌ها ارتباط برقرار می‌کند. این محدودیت امکان تغییر و ارتقا فقط در محدوده مربوطه را فراهم می‌کند و تاثیر سایر سرویس‌ها را کاهش می‌دهد.

[1] Decentralization
[2] Bounded Context

مقیاس‌پذیری[1]: به دلیل تقسیم‌بندی برنامه‌ها به صورت میکروسرویس‌ها، امکان مقیاس‌پذیری به صورت مستقل برای هر میکروسرویس وجود دارد. این به تیم‌ها اجازه می‌دهد که بخش‌های مورد نیاز را به صورت مستقل افزایش دهند و منابع را بهینه به کار گیرند.

مدیریت زنجیره ارزش[2]: هر میکروسرویس مسئولیت‌های خاص خود را دارد و با ترکیب میکروسرویس‌ها، زنجیره ارزش کلیه برنامه‌های کاربردی شکل می‌گیرد. این معماری به تیم‌ها اجازه می‌دهد براز جنبه‌های مدیریتی نوآورانه و بهبود فرآیندهای توسعه نرم‌افزار در محیط‌های پیچیده بهره‌برداری کنند.

با وجود مزایای معماری میکروسرویس، پیاده سازی آن برخی چالش‌های خاص را به همراه دارد. با افزایش تعداد میکروسرویس‌ها پیچیدگی سیستم نیز افزایش می‌یابد. مدیریت و نگهداری تعداد زیاد از میکروسرویس‌ها در رفع خطاهای احتمالی مشکل ایجاد می‌کند. هر میکروسرویس ممکن است به سرویس‌های دیگری وابستگی داشته باشد. این وابستگی‌ها می‌توانند در زمان تغییرات، نسخه‌بندی و ارتقاء سرویس‌ها مشکلاتی را ایجاد کنند و نیازمند مدیریت دقیق وابستگی ها است. مدیریت انتشار و استقرار: استقرار و به‌روزرسانی میکروسرویس‌ها می‌تواند به دلیل تعداد زیاد آن‌ها و وابستگی‌های مختلف، پیچیده باشد. به طور خاص، تضمین عدم اختلال در سرویس‌های فعال در زمان انتشار و به‌روزرسانی چالش‌هایی را ایجاد می‌کند. در سیستم‌های مبتنی بر میکروسرویس، ممکن است تراکنش‌هایی وجود داشته باشد که نیازمند هماهنگی بین چندین میکروسرویس باشند. مدیریت تراکنش‌ها و حفظ هماهنگی بین سرویس‌ها می‌تواند چالش‌هایی ایجاد کند، به خصوص زمانی که نیاز به تراکنش‌های توزیع‌شده و پیچیده‌تر وجود داشته باشد. مدیریت امنیت در معماری میکروسرویس نیز چالش‌های خود را دارد. با افزایش تعداد میکروسرویس‌ها و ارتباطات بین آن‌ها نیازمندی به استراتژی‌ها و شیوه‌های امنیتی مناسب برای حفظ امنیت سیستم و اطلاعات است. بررسی عملکرد و عیب‌یابی در سیستم مبتنی بر میکروسرویس همچنین چالش‌های خاصی دارد. نیاز به مانیتورینگ و نظارت مستمر بر عملکرد میکروسرویس‌ها، شناسایی و رفع مشکلات عملکردی و عیب‌ها، و تحلیل روند عملکرد سیستم به منظور بهبود آن، چالش‌هایی است که باید مورد توجه قرار گیرد.

در کل، معماری میکروسرویس به دلیل پیچیدگی‌ها و چالش‌هایی که ایجاد می‌کند، نیازمند برنامه‌ریزی، طراحی دقیق، استراتژی‌های مناسب و ابزارهای قدرتمند برای مدیریت آن است.

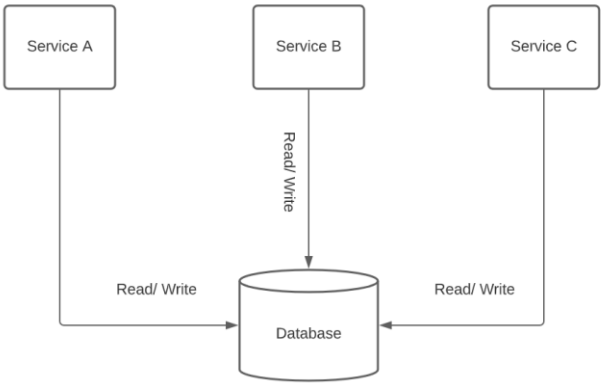
۳-۶- الگو های طراحی کاربردی برای پیاده سازی میکروسرویس

[1] Scalability
[2] Value Chain Management

پیاده‌سازی میکروسرویس‌ها به‌طور معمول با الگوهای طراحی خاصی همراه می‌شود تا این معماری به درستی اجرا و مدیریت شود. در ادامه برخی از الگوهای طراحی کاربردی برای پیاده‌سازی میکروسرویس‌ها که در پیاده سازی پروژه استفاده شده را معرفی می‌کنیم.

۳-۶-۱- الگوی پایگاه داده اشتراکی[1]

در الگوی اشتراکی پایگاه داده در هر سرویس، پایگاه داده یکسان توسط چندین میکروسرویس به اشتراک گذاشته می شود. قبل از اتخاذ این الگو، باید معماری برنامه را به دقت ارزیابی شود و مطمئن شده که از جدول های منفرد که بین چندین میکروسرویس به اشتراک گذاشته می شوند، اجتناب شود. تمام تغییرات پایگاه داده نیز باید سازگار با عقب باشد. برای مثال، توسعه‌دهندگان می‌توانند ستون‌ها یا جداول را تنها در صورتی حذف شوند که اشیا توسط نسخه‌های فعلی و قبلی همه میکروسرویس‌ها ارجاع نشده باشند. همانطور که در شکل ۳-۳ دیده می‌شود، پایگاه داده بین سرویس های مختلف مشترک است.



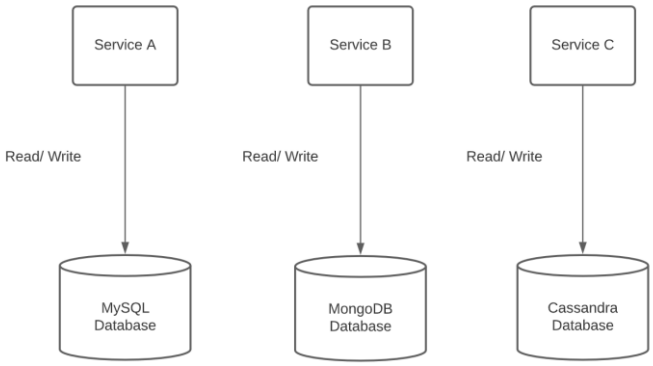
شکل ۳-۳ الگوی پایگاه داده اشتراکی [۱۰]

۳-۶-۲- الگوی پایگاه داده به ازای هر سرویس[2]

الگوی پایگاه داده به ازای هر سرویس زمانی در معماری میکروسرویس ها استفاده می شود که در آن هر میکروسرویس پایگاه داده اختصاصی خود را دارد. این الگو امکان تفکیک بهتر نگرانی ها، جداسازی داده ها و مقیاس پذیری را فراهم می کند. در معماری میکروسرویس، سرویس‌ها به‌گونه‌ای طراحی می‌شوند که کوچک، متمرکز و مستقل باشند که هر کدام وظیفه خاصی را بر عهده دارند. برای حفظ این جداسازی، اطمینان از اینکه هر میکروسرویس به طور مستقل داده های خود را مدیریت می کند، ضروری است. این الگو با تخصیص یک پایگاه داده جداگانه برای هر میکروسرویس، این امر را اعمال می کند. همانطور که در شکل ۳-۴ مشاهده

[1] Shared Database
[2] Database per Service

می‌شود این امکان وجود دارد پایگاه داده های هر سرویس نوع متفاوتی داشته باشند.



شکل ۳-۴ الگوی پایگاه داده به ازای هر سرویس [۱۰]

۳-۶-۳- الگوی ساگا[1]

الگوی کاربردی ساگا را می‌توان یک مفهوم معماری قدیمی دانست که هنوز هم برای میکروسرویس‌های امروزی بسیار سودمند است. ساگا دنباله‌ای از تراکنش‌های محلی در هر یک از میکروسرویس‌های شرکت‌کننده است. برای پیاده سازی ساگا مراحلی وجود دارد که اجرا شدن هرکدام از آن‌ها الزامی است و زمانی که کامل شوند، منطقی وجود دارد که تصمیم می‌گیرد چه کاری انجام شود. ساگا باید تضمین کند که تمامی مراحل با موفقیت به پایان رسیده است؛ در غیر این صورت، باید مقدمات لازم برای بازگشت را فراهم کند. هنگام ارائه درخواست ممکن است با مشکلات قانون زیرساخت یا منطق تجاری نیز روبرو شویم. البته نیاز است که تمامی این مشکلات را مدیریت کنیم و اگر در مرحله خاصی استثنا داریم، تمام تغییرات مراحل قبلی را نیز لغو کنیم. در بعضی مواقع، برای انجام بازگشت کامل باید درخواست‌های اضافی را به میکروسرویس‌ها ارائه دهیم.

۳-۶-۴- الگوی سرویس دیسکاوری[2]

سرویس دیسکاوری فرآیند پیدا کردن اتوماتیک سرویس ها روی شبکه است. دو نوع سرویس دیسکاوری وجود دارد: سمت سرور و سمت کلاینت. دیسکاوری سمت سرور به کلاینت ها این امکان را می‌دهد تا سرویس ها را از طریق مسیریاب پیدا کنند سرویس دیسکاوری سمت کلاینت به کلاینت ها این امکان را می‌دهد تا سرویس ها را از طریق ارسال درخواست به سرویس ثبت کننده پیدا کنند. سرویس دیسکاوری سه کامپوننت دارد: سرویس ارائه دهنده سرویس مصرف کننده و سرویس ثبت کننده. سرویس ارائه دهنده در ابتدا خودش را در سرویس ثبت کننده ثبت میکند و در انتها خودش را لغو ثبت میکند سرویس مصرف کننده آدرس سرویس های ارائه دهنده را از سرویس ثبت کننده می گیرد و بعد از آن به سرویسهای ارائه دهنده متصل

[1] Saga
[2] Service Discovery

میشود سرویس ثبت کننده یک پایگاه داده دارد که در آن آدرس شبکه نمونه‌های سرویس ها را ثبت می‌کند.

۳-۷- بررسی مزایا و معایب و مقایسه میکروسرویس با دیگر معماری ها

معماری میکروسرویس را می‌توان با چندین معماری نرم‌افزاری دیگر مقایسه کرد. در ادامه، به مقایسه معماری میکروسرویس با دو معماری یکپارچه و سرویس‌گرا می‌پردازیم:

- تجزیه پذیری: در معماری میکروسرویس برنامه‌ها به قطعات کوچکتر و مستقل تقسیم می‌شوند، در حالی که در معماری یکپارچه برنامه‌ها به صورت یک برنامه کلی و تکلیف متمرکز طراحی می‌شوند. در معماری میکروسرویس تقسیم‌بندی بر اساس مسئولیت‌های مشخص انجام می‌شود، در حالی که در معماری سرویس‌گرا تقسیم‌بندی بر اساس سرویس‌ها و قابلیت آنها است.
- استقلال: میکروسرویس‌ها در معماری میکروسرویس به طور مستقل قابل توسعه، استقرار و مقیاس‌پذیری هستند. در حالی که در معماری یکپارچه تغییرات و توسعه نیازمند تغییر در کلیه اجزاء برنامه است.
- ارتباطات: در معماری میکروسرویس، میکروسرویس‌ها تنها از طریق رابط‌های خارجی با یکدیگر ارتباط برقرار می‌کنند، در حالی که در معماری دیگر تقسیم‌شده، ارتباطات بین اجزاء داخلی برنامه به صورت مستقیم انجام می‌شود. در معماری سرویس‌گرا هر سرویس به طور مستقیم با هم ارتباط برقرار می‌کنند در حالی که در میکروسرویس معمولا از بروکر[1] برای ارتباط بین سرویس ها استفاده می‌شود.
- مقیاس‌پذیری: در معماری میکروسرویس، امکان مقیاس‌پذیری به صورت مستقل برای هر میکروسرویس وجود دارد. در معماری دیگر یکپارچه مقیاس‌پذیری برای کلیه اجزاء برنامه انجام می‌شود. میکروسرویس نسبت به سرویس‌گرا مقیاس‌پذیر تر است و برای پروژه های مبتنی بر وب مناسب تر می‌باشد.
- انعطاف‌پذیری: معماری میکروسرویس به تیم‌ها امکان می‌دهد به صورت مستقل با زبان‌ها، فناوری‌ها و ابزارهای مورد علاقه خود کار کنند. در حالی که در معماری دیگر یکپارچه، فناوری ها و ابزارهای استفاده شده معمولاً متحد الشکل است.

به طور کلی معماری نرم افزار نقش مهمی در توسعه نرم افزار و کسب و کار دارد. استفاده از سبک ها و الگو های معماری، در تصمیم گیری می‌تواند کمک کند. برای طراحی نرم افزار با در نظر گرفتن مزایا و معایب هر گونه از معماری ها، باید معماری مناسب سامانه خود را انتخاب کرده و نسبت به آن پیاده سازی کرد.

با توجه به موارد گفته شده در سامانه مورد نظر، سامانه پزشکی تشخیص بیماری و ارجاع به متخصص، از

[1] Broker

معماری میکروسرویس استفاده می‌کنیم که در فصل پنجم در مورد جزئیات پیاده سازی آن بحث می‌شود.

۳-۸- خلاصه فصل

در ابتدا این فصل معماری نرم افزار توضیح داده شده و به مفاهیم مهم و دسته بندی های آن، نظیر سبک و الگو معماری، اشاره شد. به تفصیل در مورد معماری میکروسرویس و اجزای آن و الگو های طراحی پر کاربرد در آن توضیح داده شد. در انتها با بررسی مزایا و معایب میکروسرویس، به مقایسه آن با دیگر معماری ها پرداخته شد.

فصل چهارم: نیازمندی‌های سامانه

در این فصل ابتدا به اصول مهندسی نیازمندی‌ها پرداخته شده و نیازمندی‌های عملکرد و غیر عملکردی توضیح داده می‌شود. سپس به نیازمندی‌های سامانه مورد نظر پرداخته می‌شود که این نیازمندی‌ها در فصل ششم، ارزیابی سامانه، استفاده می‌گردند.

۴-۱- مقدمه و مفاهیم پایه

نیازمندی‌های یک سیستم، شرح خدماتی است که یک سیستم باید ارائه دهد و محدودیت‌های عملکرد آن را نشان دهد. این نیازمندی‌ها منعکس کننده نیازهای مشتریان برای سیستمی است که هدف خاصی مانند کنترل یک دستگاه، ثبت سفارش یا یافتن اطلاعات را دنبال می کند. فرآیند کشف، تجزیه و تحلیل، مستندسازی و بررسی این خدمات و محدودیت ها را مهندسی نیازمندی ها[1] می نامند.

برخی از مشکلاتی که در طول فرآیند مهندسی نیازمندی‌ها به وجود می‌آیند، نتیجه ناتوانی در ایجاد تفکیک واضح بین این سطوح مختلف توصیف است. من بین آنها با استفاده از اصطلاح نیازمندی‌های کاربر به معنای نیازمندی‌های انتزاعی سطح بالا و نیازمندی‌های سیستم به معنای توصیف دقیق آنچه که سیستم باید انجام دهد، تمایز قائل شدم. نیازمندی‌های کاربر و سیستم مورد نیاز ممکن است به صورت زیر تعریف شود:

۱. نیازمندی‌های کاربر: بیانیه هایی هستند، به زبان طبیعی به اضافه نمودارها، در مورد خدماتی که انتظار می رود سیستم به کاربران سیستم ارائه دهد و محدودیت هایی که تحت آن باید عمل کند. نیازمندی‌های کاربر ممکن است از بیانیه های گسترده ویژگی های سیستم مورد نیاز تا توضیحات دقیق و دقیق از عملکرد سیستم متفاوت باشد.

۲. نیازمندی‌های سیستم: توضیحات دقیق تری از عملکردها، خدمات و محدودیت های عملیاتی سیستم نرم افزاری است. سند الزامات سیستم (که گاهی اوقات مشخصات عملکردی نامیده می شود) باید دقیقاً آنچه را که باید پیاده سازی شود، تعریف کند. ممکن است بخشی از قرارداد بین خریدار سیستم و توسعه دهندگان نرم افزار باشد.

نیازمندی‌های سیستم اطلاعات دقیق تری را در مورد خدمات و عملکردهای سیستمی که قرار است پیاده سازی شود ارائه می دهد. شما باید الزامات را در سطوح مختلف جزئیات بنویسید زیرا انواع مختلف خواننده از آنها به روش های مختلف استفاده می کنند. خوانندگان نیازمندی‌های سیستم باید بدانند که سیستم دقیقاً چه کاری انجام خواهد داد، زیرا نگران این هستند که چگونه از فرآیندهای تجاری پشتیبانی می‌کند یا به این

[1] Requirement Engineering

دلیل که در اجرای سیستم مشارکت دارند.

مهندسی نیازمندی ها معمولاً به عنوان اولین مرحله از فرآیند مهندسی نرم افزار ارائه می شود. با این حال، ممکن است قبل از تصمیم گیری برای ادامه خرید یا توسعه یک سیستم، برخی درک از الزامات سیستم ایجاد شود. باید در نظر داشت گاهی اوقات یک رویکرد چابک برای استخراج همزمان الزامات در حین توسعه سیستم ممکن است برای افزودن جزئیات و اصلاح نیازهای کاربر استفاده شود. [۱۱]

۴-۲- دسته‌بندی نیازمندی‌ها

نیازمندی‌های سیستم اغلب به عنوان نیازمندی‌های عملکردی یا غیرعملکردی[1] طبقه بندی می شوند:

۱. نیازمندی‌های عملکردی: اینها بیانگر خدماتی هستند که سیستم باید ارائه دهد، سیستم چگونه باید به ورودی های خاص واکنش نشان دهد و چگونه سیستم باید در موقعیت های خاص رفتار کند. در برخی موارد، نیازمندی‌های عملکردی نیز ممکن است به صراحت بیان کند که سیستم نباید انجام دهد.

۲. نیازمندی‌های غیرعملکردی: اینها محدودیت‌هایی در خدمات یا عملکردهای ارائه شده توسط سیستم هستند. آنها شامل محدودیت‌های زمان‌بندی، محدودیت‌های فرآیند توسعه و محدودیت‌های اعمال‌شده توسط استانداردها هستند. نیازمندی‌های غیرعملکردی اغلب در کل سیستم به جای ویژگی ها یا خدمات منفرد سیستم اعمال می شود. در واقع، تمایز بین انواع مختلف نیازمندی‌ها چندان واضح نیست

همانطور که این تعاریف ساده نشان می دهد. یک نیازمندی کاربر مربوط به امنیت، مانند بیانیه ای که دسترسی به کاربران مجاز را محدود می کند، ممکن است یک نیاز غیر کاربردی به نظر برسد. با این حال، زمانی که این نیاز با جزئیات بیشتر توسعه یابد، ممکن است نیازهای دیگری ایجاد کند که به وضوح عملکردی هستند، مانند نیاز به گنجاندن امکانات احراز هویت کاربر در سیستم.

این موارد نشان می‌دهد که نیازمندی‌ها مستقل نیستند و یک نیاز اغلب نیازمندی‌های دیگر را ایجاد یا محدود می‌کند. بنابراین الزامات سیستم فقط خدمات یا ویژگی های سیستم مورد نیاز را مشخص نمی کند. آنها همچنین عملکردهای لازم را برای اطمینان از اینکه این خدمات/ویژگی ها به طور موثر ارائه می شوند، مشخص می کنند.

۴-۲-۱- نیازمندی‌های عملکردی

[1] Functional and Non-functional Requirements

نیازمندی‌های عملکردی یک سیستم توصیف می کند که سیستم باید چه کاری انجام دهد. این نیازمندی‌ها به نوع نرم افزار در حال توسعه، کاربران مورد انتظار نرم افزار و رویکرد کلی سازمان در هنگام نوشتن نیازمندی ها بستگی دارد. هنگامی که به عنوان نیازهای کاربر بیان می شود، نیازمندی‌های عملکردی باید به زبان طبیعی نوشته شود تا کاربران و مدیران سیستم بتوانند آنها را درک کنند. سیستم مورد نیاز عملکردی نیازهای کاربر را گسترش می دهد و برای توسعه دهندگان سیستم نوشته شده است. آنها باید توابع سیستم، ورودی ها و خروجی های آنها و استثناها را با جزئیات شرح دهند.

نیازمندی‌های سیستم عملکردی از نیازمندی‌های عمومی که شامل کارهایی که سیستم باید انجام دهد تا نیازمندی‌های بسیار خاص که منعکس کننده روش های محلی کار یا سیستم های موجود سازمان است، متفاوت است. نیازمندی‌های عملکردی، همانطور که از نام آن پیداست، به طور سنتی بر آنچه که سیستم باید انجام دهد متمرکز شده است. با این حال، اگر سازمانی تصمیم بگیرد که یک محصول نرم‌افزاری موجود در سیستم خارج از قفسه می‌تواند نیازهای آن را برآورده کند، در آن صورت توسعه یک مشخصات عملکردی دقیق، اهمیت کمی دارد.

در حالت ایده آل، مشخصات نیازمندی‌های عملکردی یک سیستم باید کامل و سازگار باشد. منظور از کامل بودن این است که کلیه خدمات و اطلاعات مورد نیاز کاربر باید تعریف شود. سازگاری به این معنی است که نیازمندی‌ها نباید متناقض باشند.

در عمل، دستیابی به سازگاری و کامل بودن نیازمندی‌ها فقط برای سیستم های نرم افزاری بسیار کوچک امکان پذیر است. یکی از دلایل این است که هنگام نوشتن مشخصات سیستم های بزرگ و پیچیده به راحتی می توان اشتباه کرد و از قلم افتاد. دلیل دیگر این است که سیستم های بزرگ دارای ذینفعان زیادی با پیشینه ها و انتظارات متفاوت هستند. ذینفعان احتمالاً نیازهای متفاوت و اغلب متناقض دارند. این تناقضات ممکن است زمانی که نیازمندی‌ها در اصل مشخص شده باشند آشکار نباشند و الزامات ناسازگار ممکن است تنها پس از تجزیه و تحلیل عمیق تر یا در طول توسعه سیستم کشف شوند.

۴-۲-۲- نیازمندی‌های غیر عملکردی

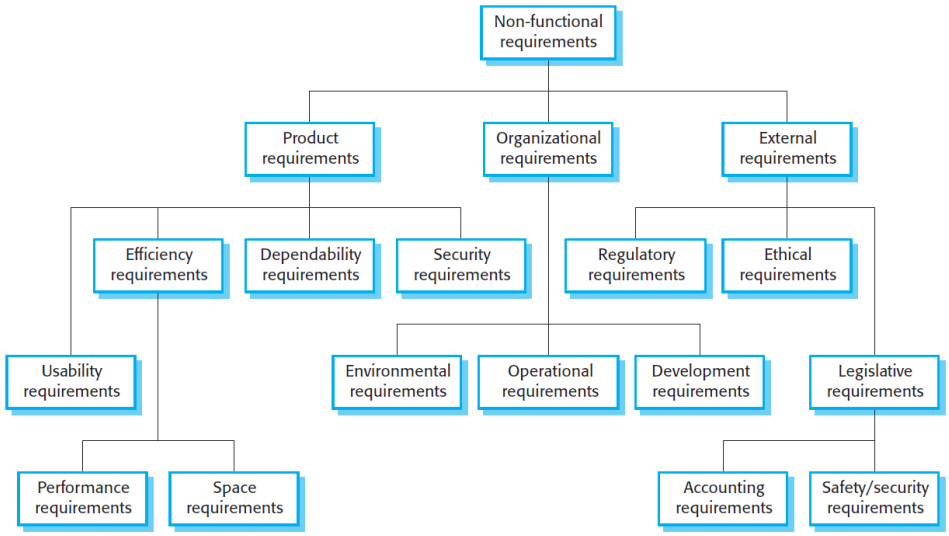
همانطور که از نام آن پیداست، نیازمندی‌های غیرعملکردی، نیازمندی‌هایی هستند که مستقیماً به خدمات خاص ارائه‌شده توسط سیستم به کاربران مربوط نمی‌شوند. این الزامات غیرعملکردی معمولاً ویژگی های سیستم را به عنوان یک کل مشخص یا محدود می کند. آنها ممکن است به ویژگی های سیستم اضطراری مانند قابلیت اطمینان، زمان پاسخگویی و استفاده از حافظه مربوط باشند. از طرف دیگر، آنها ممکن است محدودیت هایی را بر روی آن تعریف کنند

پیاده سازی سیستم، مانند قابلیت های دستگاه های ورودی/خروجی یا نمایش داده های مورد استفاده در رابط با سایر سیستم ها. نیازمندی‌های غیرعملکردی اغلب حیاتی تر از الزامات عملکردی فردی هستند. کاربران سیستم معمولاً می‌توانند راه‌هایی برای کار در مورد عملکرد سیستمی بیابند که واقعاً نیازهای آنها را برآورده نمی‌کند. با این حال، عدم برآورده کردن یک نیاز غیر عملکردی می تواند به معنای غیرقابل استفاده بودن کل سیستم باشد. به عنوان مثال، اگر یک سیستم هواپیما الزامات قابلیت اطمینان خود را برآورده نکند، برای عملیات ایمن بودن آن تایید نمی شود. اجرای این الزامات ممکن است به دو دلیل در سراسر سیستم پخش شود:

۱- نیازمندی‌های غیرعملکردی ممکن است معماری کلی یک سیستم را به جای اجزای منفرد تحت تأثیر قرار دهد. به عنوان مثال، برای اطمینان از برآورده شدن الزامات عملکرد در یک سیستم تعبیه شده، ممکن است مجبور شوید سیستم را سازماندهی کنید تا ارتباطات بین اجزا را به حداقل برسانید.

۲- یک نیاز غیرعملکردی فردی، مانند یک نیاز امنیتی، ممکن است چندین مورد نیاز عملکردی مرتبط را ایجاد کند که سرویس‌های سیستم جدیدی را تعریف می‌کند که در صورت اجرای نیازمندی‌های غیرعملکردی مورد نیاز است.

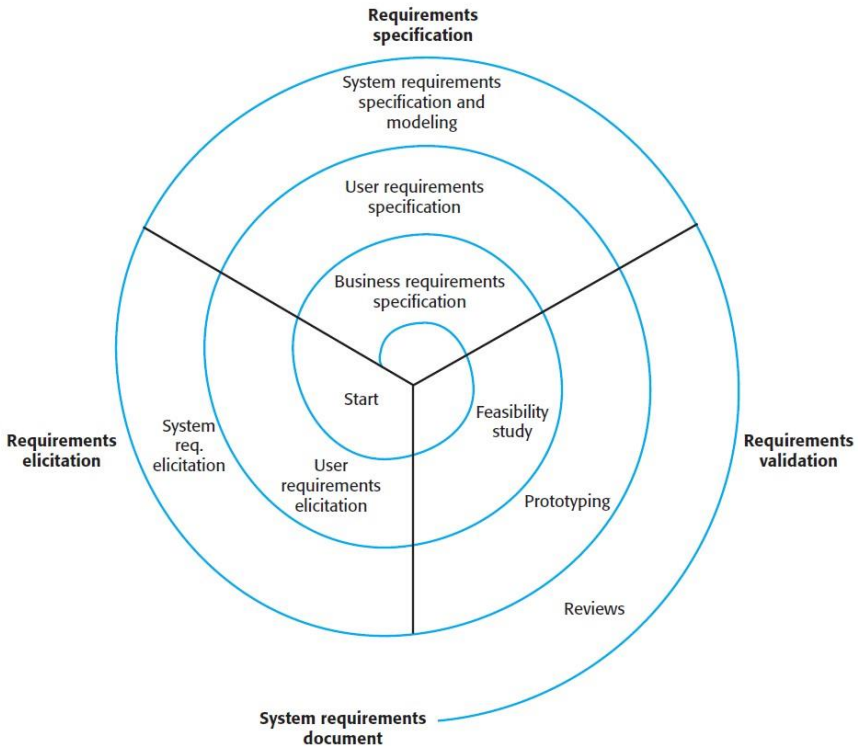
نیازمندی‌های غیرعملکردی از طریق نیازهای کاربر به دلیل محدودیت های بودجه، سیاست های سازمانی، نیاز به قابلیت همکاری با سایر سیستم های نرم افزاری یا سخت افزاری، یا عوامل خارجی مانند مقررات ایمنی یا قوانین حفظ حریم خصوصی به وجود می آیند. شکل ۴-۱ طبقه بندی نیازمندی‌های غیرعملکردی است.



شکل ۴-۱ طبقه‌بندی نیازمندی‌های غیرعملکردی

۴-۳- فرایند مهندسی نیازمندی

همانطور که در فصل‌های گذشته بحث شد ، مهندسی نیازمندی ها شامل سه فعالیت کلیدی است. اینها کشف نیازمندی‌ها از طریق تعامل با ذینفعان هستند (استنباط و تجزیه و تحلیل). تبدیل این نیازمندی‌ها به یک فرم استاندارد (مشخصات)؛ و بررسی اینکه نیازمندی‌ها واقعاً سیستم مورد نظر مشتری را تعریف می کنند (اعتبارسنجی). این حال، در عمل، مهندسی نیازمندی ها یک فرآیند تکراری است که در آن فعالیت ها به هم متصل می شوند. شکل ۴-۲ این درهم آمیختگی را نشان می دهد. فعالیت ها به عنوان یک فرآیند تکرار شونده حول یک مارپیچ سازماندهی می شوند. خروجی فرآیند مهندسی نیازمندی یک سند مورد نیاز سیستم است. مقدار زمان و تلاشی که برای هر فعالیت در یک تکرار اختصاص می‌یابد به مرحله فرآیند کلی، نوع سیستم در حال توسعه و بودجه موجود بستگی دارد. در اوایل فرآیند، بیشترین تلاش صرف درک کسب و کار سطح بالا خواهد شد و نیازمندی‌ها غیر عملکردی و الزامات کاربر برای سیستم. در مراحل بعدی، در حلقه‌های بیرونی مارپیچ، تلاش بیشتری برای استخراج و درک نیازمندی‌های غیرعملکردی و نیازمندی‌های سیستم دقیق‌تر اختصاص خواهد یافت. این مدل مارپیچی رویکردهای توسعه را در خود جای می دهد که در آن نیازمندی‌ها به سطوح مختلف جزئیات توسعه می یابند. در توسعه چابک می تواند به جای نمونه سازی استفاده شود تا نیازمندی‌ها و پیاده سازی سیستم با هم توسعه یابند. شکل ۴-۲ نمایانگر فرایند مهندسی نیازمندی است که توضیح مفصل این مراحل خارج از مفاهیم این پایان‌نامه می‌باشد.



شکل ۴-۲ فرایند مهندسی نیازمندی [۱۱]

۴-۴- نیازمندی‌های سامانه هدف

مطابق مطالب گفته شده در قسمت های قبل، فرایند مهندسی نیازمندی‌ها طی شد. سپس بر اساس مراحل این فرایند نیازمندی‌های عملکردی و غیرعملکردی نوشته شد. که در این قسمت به آنها اشاره می‌شود. برای بررسی میزان موثر بودن و عملکرد سامانه نیاز است که از هر دو نوع نیازمندی استفاده شود.

نیازمندی‌های عملکردی:

- کاربر باید بتواند در سیستم ثبت‌نام کرده و در دفعات بعد با نام کاربری و رمز عبور مختص خود وارد شود.
- کاربر باید بتواند پس از وارد شدن، پروفایل کاربری خود را مشاهده کند.
- کاربر باید علائم بیماری را در رابط کاربری مناسب ببینید و بتواند آنها را انتخاب کند.
- سامانه باید بتواند پس از گرفتن علائم بیماری به عنوان ورودی، بیماری‌های احتمالی را تشخیص داده و آنها را با اولویت‌بندی به کاربر نمایش دهد.
- سامانه باید بتواند بیماری‌ها را به کاربر نمایش دهد و کاربر این قابلیت ها را داشته باشد تا از آنها انتخاب کرده و تخصص‌های مربوط به آن بیمارها را به عنوان خروجی ببیند.
- کاربری که بیماری خود و متخصص خود را می‌داند، بدون نیاز به فرایند تشخیص بیماری، باید بتواند که متخصصین مربوطه را جستجو کند.
- سامانه باید متخصصین را به ترتیب امتیاز آنها نمایش دهد.
- اطلاعات مربوط به سامانه و توسعه دهندگان باید در صفحه‌ای با عنوان درباره ما وجود داشته باشد.
- سامانه باید این قابلیت را داشته باشد که فرایند های زیر را به شکل مستقل انجام دهد: ثبت‌نام و ورود کاربر، تشخیص بیماری از علائم، تشخیص تخصص از بیماری، جستجوی متخصص

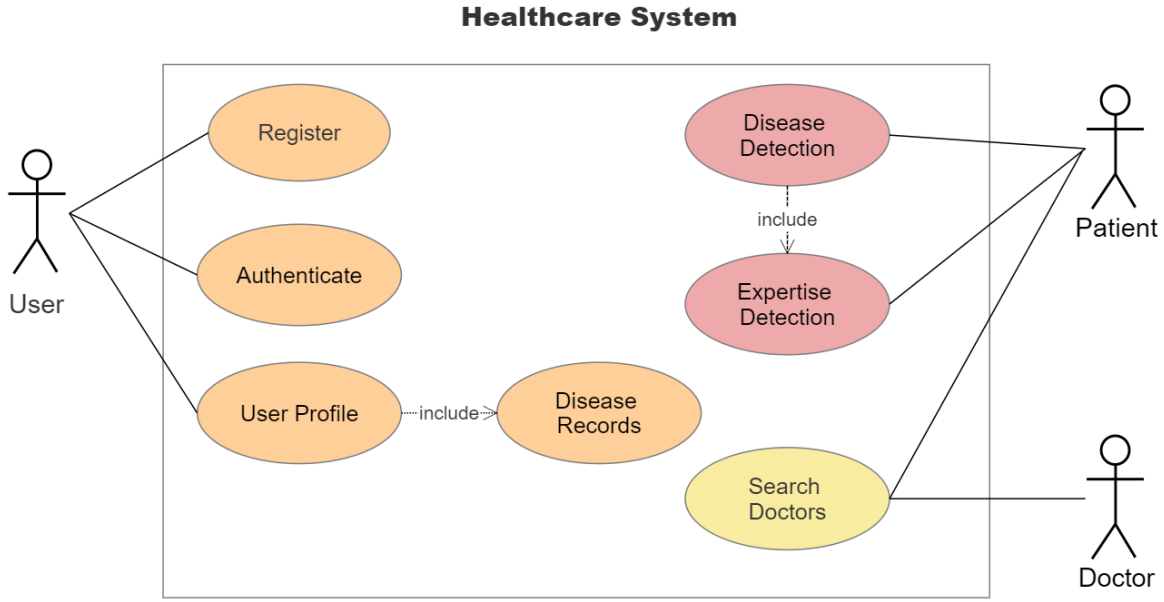
نیازمندی‌های غیرعملکردی:

- قابل اعتماد بودن: سامانه نباید با تعداد درخواست بالا یا تکرار کردن همزمان سرویس‌ها مختل شود.
- مقیاس پذیری: سامانه باید به نوعی طراحی شود که هر سرویس به طور مستقل طراحی و پیاده‌سازی شود؛ به نوعی که اگر بدون رابط کاربری و صرفا با استفاده از ای‌پی‌آی به سامانه درخواست زده شود، بدون در نظر گرفتن دیگر سرویس‌ها پاسخ دهد.
- قابل استفاده بودن: رابط کاربری باید به نوعی باشد که به راحتی انتقال بین صفحات و سرویس‌ها ردگیری شود و بتوان از آنها استفاده کرد.
- امنیت: در صورتی که کاربر بخواهد با نام کاربری که از قبل در پایگاه داده ذخیره شده است، وارد شود باید خطا بگیرد.

- زمان پاسخ تشخیص بیماری: پس از گرفتن علائم ورودی، سامانه باید زیر ۲ ثانیه بیماری‌های مرتبط را نمایش دهد.
- زمان پاسخ تشخیص تخصص از بیماری: پس از گرفتن بیماری‌های ورودی، سامانه باید زیر ۱ ثانیه تخصص‌های مرتبط را نمایش دهد.
- زمان جستجوی متخصص: پس از گرفتن یک تخصص از کاربر، سامانه باید زیر ۳ ثانیه متخصصین مرتبط را جستجو کرده و نمایش دهد. بارگذاری تصاویر متخصصین باید زمان کمی بگیرد.
- زمان استقرار بر سرور: زمان اجرا برنامه و قرار گرفتن آن بر سرور نباید از ۵ ثانیه بیشتر شود.

همانطور که گفته شد، سامانه هدف باید به نوعی طراحی شود که نیازمندی‌های گفته شده را عملی کند.

نمودار مورد کاربرد[1]، عملکردهای سطح بالا و دامنه یک سیستم را توصیف می کند. این نمودار همچنین تعاملات بین سیستم و بازیگران آن را مشخص می کند. همچنین این نمودار توصیف می‌کند که سیستم چه می‌کند و بازیگران چگونه از آن استفاده می‌کنند، اما نه نحوه عملکرد داخلی سیستم در آن مشخص نیست. شکل ۴-۳ نمودار مورد کاربرد سامانه است.



شکل ۴-۳ نمودار مورد کاربرد سامانه

[1] Use-case Diagram

۴-۵- خلاصه فصل

در این فصل ابتدا به مفاهیم نظری در رابطه با مهندسی نیازمندی‌ها پرداخته شده و بعد از ذکر اصول آن، دسته‌بندی‌های نیازمندی اعم از نیازمندی‌های عملکردی و غیرعملکردی توضیح داده شد. سپس در مورد فرایند مهندسی نرم‌افزار بحث شد و بر آن اساس نیازمندی‌های سامانه هدف جمع‌آوری شد. در انتها پس از گفتن نیازمندی‌های عملکردی و غیرعملکردی سامانه، نمودار مورد کاربرد آن رسم شد.

همانطور که می‌دانیم طراحی سامانه ارتباط تنگاتنگی با نیازمندی‌های آن دارد. بر اساس نیازمندی‌های مطرح شده سامانه طراحی شده و در فصل بعد به تفصیل به آن پرداخته می‌شود. در نهایت این نیازمندی ها در فصل ششم، ارزیابی سامانه، مورد تحلیل قرار می‌گیرند.

فصل پنجم: طراحی و پیاده سازی

در این فصل به طراحی و نمونه سازی سامانه مورد نظر پرداخته می‌گردد. ابتدا به معرفی کلی اجزای سامانه و نقش و وظایف هر کدام پرداخته می‌شود، سپس هر کدام از سرویس ها به تفصیل توضیح داده می‌شود. در هر یک از این بخش‌ها، ابتدا ابزار های پیاده سازی گفته شده و در ادامه نحوه پیاده سازی و روش های آن بررسی می‌گردد.

۵-۱- معرفی کلی سامانه

همان‌طور که در بخش های قبلی گفته شد، در این پروژه یک سامانه پزشکی تشخیص بیماری و ارجاع به متخصص با معماری میکروسرویس طراحی و نمونه‌سازی شده است. در واقع سامانه یک وب اپلیکیشن است که به صورت محلی اجرا شده و با استفاده از رابط کاربری قابلیت استفاده می‌باشد. به صورت کلی این سامانه به نوعی طراحی شده است که مقیاس‌پذیر و در آینده قابل توسعه مستمر و نگهداری باشد، در واقع یکی از اصلی‌ترین دلایل استفاده از میکروسرویس همین مسئله می‌باشد. بنابراین به مرور می‌توان قابلیت های این برنامه را افزایش داد. با توجه به عملکرد برنامه، که در بخش سرویس‌ها بیشتر توضیح داده می‌شود.

۵-۲- طراحی سامانه

در این قسمت در مورد طراحی سامانه صحبت می‌شود.در فصل سوم در مورد معماری میکروسرویس توضیح داده شد، در این بخش سعی می‌شود از توضیحات تکراری صرف نظر شود.

معماری سامانه پزشکی یک جنبه حیاتی است که زیربنای عملکرد کلی، مقیاس پذیری و عملکرد آن است. این بخش کاوش عمیقی از معماری سیستم ارائه می‌کند و منطق اتخاذ رویکرد میکروسرویس و پیامدهای گسترده‌تر آن را برجسته می‌کند. این سامانه بر اساس معماری میکروسرویس ها طراحی شده است، یک الگوی معماری مدرن که پیمانه‌ای بودن، مقیاس پذیری و انعطاف پذیری را ارتقا می دهد. میکروسرویس‌ها شامل تجزیه یک برنامه پیچیده به مجموعه‌ای از خدمات کوچک و مستقل قابل استقرار است که هر کدام مسئول جنبه خاصی از عملکرد هستند. این معماری به چند دلیل قانع کننده انتخاب شده است:

یکی از مزایای اصلی میکروسرویس ها پیمانه‌ای بودن آنهاست. در زمینه مراقبت های بهداشتی، اجزای مختلف سیستم، مانند تشخیص بیماری، مدیریت بیمار، و ارجاع به پزشک، می توانند به عنوان میکروسرویس های جداگانه پیاده سازی شوند. این ساختار پیمانه‌ای اجزای منفرد را قادر می‌سازد تا به طور مستقل تکامل یابند، که به ویژه در یک چشم‌انداز مراقبت‌های بهداشتی که به سرعت در حال تغییر است، سودمند است.

مقیاس پذیری یکی دیگر از عوامل محوری است. میکروسرویس ها را می توان به صورت افقی مقیاس بندی کرد و به سرویس های خاص اجازه می دهد تا بارهای افزایش یافته را به طور مستقل مدیریت کنند. به عنوان

مثال، در زمان اوج تقاضا، خدمات تشخیص بیماری را می توان به آسانی برای پاسخگویی به درخواست ها بدون تأثیر بر سایر بخش های سیستم، مقیاس کرد. این مقیاس پذیری پویا تضمین می کند که سیستم می تواند به طور موثر با بارهای کاری متفاوت سازگار شود.

میکروسرویس ها انعطاف پذیری را برای استفاده از فناوری های مختلف برای سرویس های مختلف اعطا می کنند. این تنوع فناوری، انتخاب مناسب ترین ابزارها و چارچوب ها را برای هر میکروسرویس، بهینه سازی عملکرد و عملکرد، امکان پذیر می سازد. به عنوان مثال، خدمات تشخیص بیماری ممکن است به کتابخانه های تخصصی یادگیری ماشین نیاز داشته باشد، در حالی که خدمات مدیریت بیمار ممکن است از یک سیستم پایگاه داده قوی بهره‌مند شود. معماری میکروسرویس ها این نیازهای متنوع را برآورده می کند.

میکروسرویس ها از شیوه های توسعه چابک پشتیبانی می کنند. تیم های توسعه می توانند بر اجزای کوچکتر و قابل مدیریت تمرکز کنند که چرخه های توسعه و آزمایش سریعتر را تسهیل می کند. علاوه بر این، میکروسرویس‌های فردی را می‌توان به طور مستقل مستقر کرد، که امکان یکپارچه‌سازی مداوم و استقرار مداوم خطوط لوله تحویل مستمر و ادغام مستمر را فراهم می‌کند. این چابکی زمان ورود به بازار برای ویژگی‌ها و پیشرفت‌های جدید را کاهش می‌دهد، که در حوزه مراقبت‌های بهداشتی بسیار مهم است، جایی که به‌روزرسانی‌های به موقع می‌تواند مراقبت و نتایج بیمار را بهبود بخشد.

۵-۲-۱- اجزای سامانه

این برنامه به سه سرویس اصلی تقسیم می‌شود. هر سرویس عملکرد خاص خودش را دارد و به صورت کاملا مستقل است به گونه‌ای که راه اصلی ارتباط بین سرویس‌ها، ای‌پی‌آی هر سرویس است. هر سرویس در پوشه متفاوت پیاده‌سازی شده و کاملا به صورت پیمانه‌ای است. سرویس‌ها عبارتند از: سرویس کاربر، سرویس تشخیص بیماری، سرویس تشخیص تخصص، سرویس جستجوی پزشک

۱- سرویس کاربر: در این سرویس کاربر می‌تواند ثبت نام کرده یا وارد شود. در بخش پروفایل کاربر، لیستی از بیماری ها و لیستی از تخصص هایی که باید مراجعه کرده، قرار دارد.

۲- سرویس تشخیص بیماری و ارجاع به متخصص: در این قسمت کاربر علائم خود را به عنوان ورودی داده و با پردازش الگوریتم یادگیری ماشین بیماری‌های احتمالی نمایش داده می‌شود.

۳- سرویس تشخیص تخصص: بعد از یافتن بیماری‌ها این قابلیت وجود دارد که کاربر تخصص مورد نظر را از بیماری‌های گفته شده پیدا کند و لیستی از آنها نمایش داده می‌شود.

۴- سرویس جستجوی پزشک: کاربر با وارد کردن تخصص مورد نظر، می‌تواند متخصصین آن حوزه را مشاهده کند. لازم به ذکر است این سرویس در دو حالت کاربرد دارد؛ اگر کاربر تخصص مورد نظر را بداند مستقیما از رابط کاربری می‌تواند جستجو را انجام دهد و در صورتی که آنرا نداند، باید از سرویس تشخیص بیماری و ارجاع به متخصص استفاده کند و بعد از آن از این سرویس استفاده کند.

۵-۲-۲- پایگاه داده و نماها

الگوی طراحی استفاده شده در معماری میکروسرویس مورد نظر پایگاه داده اشتراکی، در فصل سه توضیح داده شده، می‌باشد. در معماری میکروسرویس این امکان وجود دارد که به ازای هر سرویس، پایگاه داده مجزایی ایجاد شود؛ از آنجایی که سرویس ها اگرچه مستقل هستند با همدیگر تعامل دارند، و تعداد داده‌های ما و انواع آن متنوع نبود، همگی در پایگاه داده رابطه‌ای مای‌اس‌کیوال[1] قرار گرفتند. دلایل استفاده از آن عبارتند از:

- راحتی در استفاده: تنظیمات این برنامه بسیار راحت است و با ابزار های دیگر سازگار می‌باشد.
- عملکرد قوی: با توجه به اینکه ساختار داده ها پیچیده نیست، سرعت آن بسیار بالا است.
- پشتیبانی از مفاهیم اسید[2] در تراکنش ها: تراکنش های پایگاه داده از این اصول پیروی می‌کنند.
- متن باز بودن آن: متن باز و رایگاه بوده و کتابخانه های ارتباطی آن در زبان های مختلف موجود است.

در شکل ۵-۱ مدل پایگاه داده قرار دارد که در بخش سرویس ها هر کدام از آن ها توضیح داده می‌شوند.

شکل ۵-۱ مدل پایگاه داده

۵-۳- پیاده‌سازی سرویس‌ها

همانطور که گفته شد، این برنامه به سه سرویس اصلی تقسیم می‌شود. هر سرویس عملکرد خاص خودش را دارد و به صورت کاملا مستقل است به گونه‌ای که راه اصلی ارتباط بین سرویس‌ها، ای‌پی‌آی هر سرویس است. هر سرویس در پوشه متفاوت پیاده‌سازی شده و کاملا به صورت پیمانه‌ای است. سرویس‌ها عبارتند از: سرویس

[1] MySQL
[2] ACID: Atomicity, Consistency, Isolation, Durability

کاربر، سرویس تشخیص بیماری و ارجاع به متخصص، سرویس جستجوی پزشک

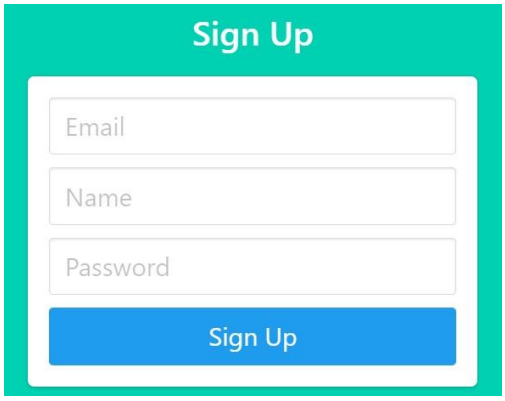
در ادامه ابتدا توضیحات مربوط به هر سرویس و عملکرد آن داده شده، سپس تصویر صفحه وب مربوط به آن نمایش داده می‌شود.

۵-۳-۱- سرویس کاربر

این سرویس شامل سه عملکرد اصلی می‌باشد: ثبت نام، اعتبارسنجی و ورود، پروفایل کاربری، صفحه خانه

عمده پیاده‌سازی این سرویس با کتابخانه های درون چارچوب فلسک انجام گردیده است.

۱- ثبت نام: همانطور که در شکل ۵-۱ نمایش داده شد، هر کاربر دارای ویژگی های مختلفی نظیر نام، ایمیل، کلمه عبور، شهر و شماره همراه است. در صفحه ثبت‌نام کاربر می‌تواند با وارد کردن اسم، ایمیل و کلمه عبور خود در سامانه ثبت نام کند. تابع مربوط به آن بررسی می‌کند که اگر کاربر با یک ایمیل مشخص از قبل در سامانه ثبت نام کرده باشد، خطا دهد. از طرفی برای حفظ امنیت سامانه، کلمه عبور به صورت رمزنگاری شده در پایگاه داده ذخیره می‌گردد. شکل ۵-۲ تصویر صفحه ثبت‌نام می‌باشد.

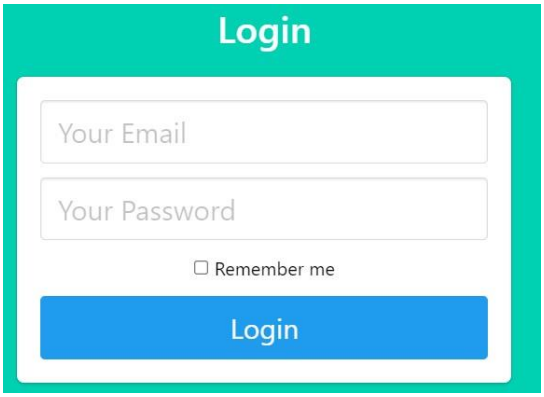


شکل ۵-۲ صفحه ثبت نام

۲- اعتبارسنجی و ورود: در این قسمت کاربر ایمیل و رمز عبور خود را وارد می‌کند. اگر این مشخصات در پایگاه داده وجود داشته باشد، وارد می‌شود و در صورتی که وجود نداشته باشد، خطا می‌دهد. همانطور که در شکل ۵-۳ مشاهده می‌کنید، گزینه بخاطر سپردن[1]، وجود دارد و در صورتی که کاربر آنرا بزند، نشست آن برای بازه زمانی مشخصی ذخیره می‌گردد و با متوقف کردن و مجددا اجرا کردن برنامه همچنان کاربر در حالت ورود در سامانه قرار دارد. در واقع نشست آن در سطح برنامه (نه پایگاه داده یا حافظه مرورگر) ذخیره می‌گردد.

[1] Remember Me

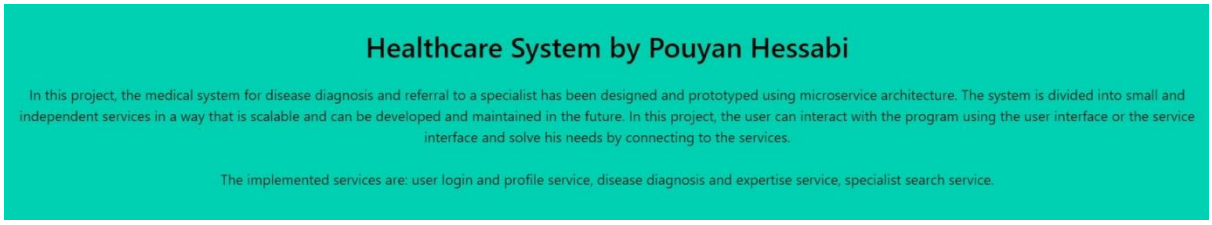
همچنین دکمه‌ای در بالای صفحه وجود دارد که در صورتی که برآن کلیک شود کاربر خارج می‌شود.



شکل ۵-۳ صفحه ورود کاربر

۳- صفحه پروفایل کاربری: در این صفحه با توجه به نشست موجود در برنامه، کاربر وارد شده را از برنامه دریافت می‌کند. پس از دریافت کاربر مورد نظر، در پایگاه داده جستجو کرده و مطابق شکل ۵-۱ از جدول کاربر-بیماری[1]، تمام بیماری های کاربر را نمایش می‌دهد. همچنین این قابلیت وجود دارد که از جدول کاربر-تخصص[2] تمام تخصص هایی که کاربر مورد نظر باید به آن مراجعه کند، نمایش داده شود.

۴- صفحه خانه: این صفحه عملیات خاصی انجام نمی‌دهد و توضیحاتی در مورد سامانه به کاربر می‌دهد. از آنجایی که هدف این پروژه محصول نهایی برای کاربر نیست، تمرکز زیادی بر این صفحه و گرافیک آن نشده است. در آینده می‌توان قابلیت های بیشتری مطابق استاندارد های طراحی و توسعه وب اپایکیشن ها، به این صفحه اضافه کرد. شکل ۵-۴ این صفحه را نشان می‌دهد.



شکل ۵-۴ صفحه خانه

شکل ۵-۵ نمایانگر نوار ناوبری بالای صفحه است که می‌توان به صفحات و سرویس های دیگر مراجعه کرد.

شکل ۵-۵ نوار ناوبری بالای صفحه

[1] User_Disease
[2] User_Expertise

۵-۳-۲- سرویس تشخیص بیماری و مدل هوش مصنوعی آن

این سرویس در واقع به دو قسمت تشخیص بیماری و ارجاع به متخصص تقسیم می‌شود. در این بخش هر کدام از این دو قسمت به تفکیک توضیح داده می‌شود.

در بخش تشخیص بیماری کاربر می‌تواند علائم مورد نظر را از نوار جانبی سمت چپ واقع در رابط کاربری، انتخاب کرده و به مدل هوش مصنوعی به عنوان ورودی بدهد؛ سپس با پردازش در بک‌اند و مدل هوش مصنوعی خروجی این تابع به شکل لیستی از تخصص ها نمایش داده می‌شود. به دلیل اینکه تعداد آنها بالاست فقط پنج تا از بیشترین مقادیر نمایش و در جدول کاربر-بیماری ثبت می‌شود. سرویس قسمت دوم مربوط به ارجاع به متخصص می‌باشد که ورودی آن لیستی از بیماری‌ها با احتساب مقادیر آن‌ها، که به درصد نوشته شده، به بک‌اند داده می‌شود. سپس با ضرب برداری مقادیر با جدول تناسب بیماری و تخصص، که در ادامه توضیح داده می‌شود، به لیستی از تخصص‌های متناظر با بیماری می‌رسد؛ که در نهایت پنج تا از مرتبط ترین تخصص ها را به کاربر نمایش می‌دهد. در ادامه به جزئیات آن بیشتر پرداخته می‌شود.

ابتدا نیاز است مجموعه داده [۱۲] مربوط به تشخیص بیماری را ببینیم تا مدل هوش مصنوعی آن را طراحی کنیم. همانطور که در شکل ۵-۶ مشاهده می‌کنید، داده ها به صورتی وجود دارد که به ازای هر سطر که نام بیماری است، علائم آن نوشته شده است. در واقع هر سطح از این مجموعه داده بیانگر یک بیمار است.

Disease Diseases that may be present	Symptom_1 the symptoms experienced during the disease	Symptom_2 the symptoms experienced during the disease	Symptom_3 the symptoms experienced during the disease	Symptom_4 the symptoms experienced during the disease
41 unique values	vomiting 17% fatigue 14% Other (3408) 69%	vomiting 18% fatigue 8% Other (3648) 74%	fatigue 15% high_fever 7% Other (3870) 79%	high_fever 8% [null] 7% Other (4194) 85%
Fungal infection	itching	skin_rash	nodal_skin_eruptions	
Fungal infection	itching	skin_rash	nodal_skin_eruptions	dischromic _patches
Allergy	continuous_sneezing	shivering	chills	watering_from_eyes
Allergy	shivering	chills	watering_from_eyes	

شکل ۵-۶ نمونه ای از مجموعه داده علائم-بیماری [۱۲]

متناسب با این مجموعه داده می‌توان رویکرد حل این مسئله را انتخاب کرد. الگوریتم هوش مصنوعی استفاده شده در این مسئله ایکس‌جی‌بوست[1] می‌باشد که در ادامه به طور مختصر در مورد آن توضیح داده می‌شود. لازم به ذکر بررسی دقیق و مقایسه با مدل‌های هوش مصنوعی دیگر خارج از سرفصل های این پایان‌نامه است که در این مقال نمی‌گنجد.

[1] XGBoost

الگوریتم ایکس‌جی‌بوست: ایکس‌جی‌بوست یک پیاده‌سازی متن باز محبوب و کارآمد از الگوریتم درخت تقویت‌شده گرادیان[1] است. تقویت گرادیان یک الگوریتم یادگیری نظارت شده است که تلاش می‌کند تا با ترکیب تخمین‌های مجموعه‌ای از مدل‌های ساده‌تر و ضعیف‌تر، متغیر هدف را به‌طور دقیق پیش‌بینی کند. در یادگیری ماشین، تقویت یک الگوریتم مجموعه ای[2] است که برای کاهش بایاس[3] و همچنین واریانس در یادگیری نظارت شده استفاده می‌گردد. همچنین خانواده ای از الگوریتم های یادگیری ماشین است که یادگیرندگان ضعیف را به افراد قوی تبدیل می کند. ایکس‌جی‌بوست چند مزیت فنی نسبت به سایر روش‌های تقویت گرادیان ارائه می‌کند، از جمله مسیر مستقیم‌تر به حداقل خطا، همگرایی سریع‌تر با مراحل کمتر، و محاسبات ساده برای بهبود سرعت و کاهش هزینه‌های محاسباتی.

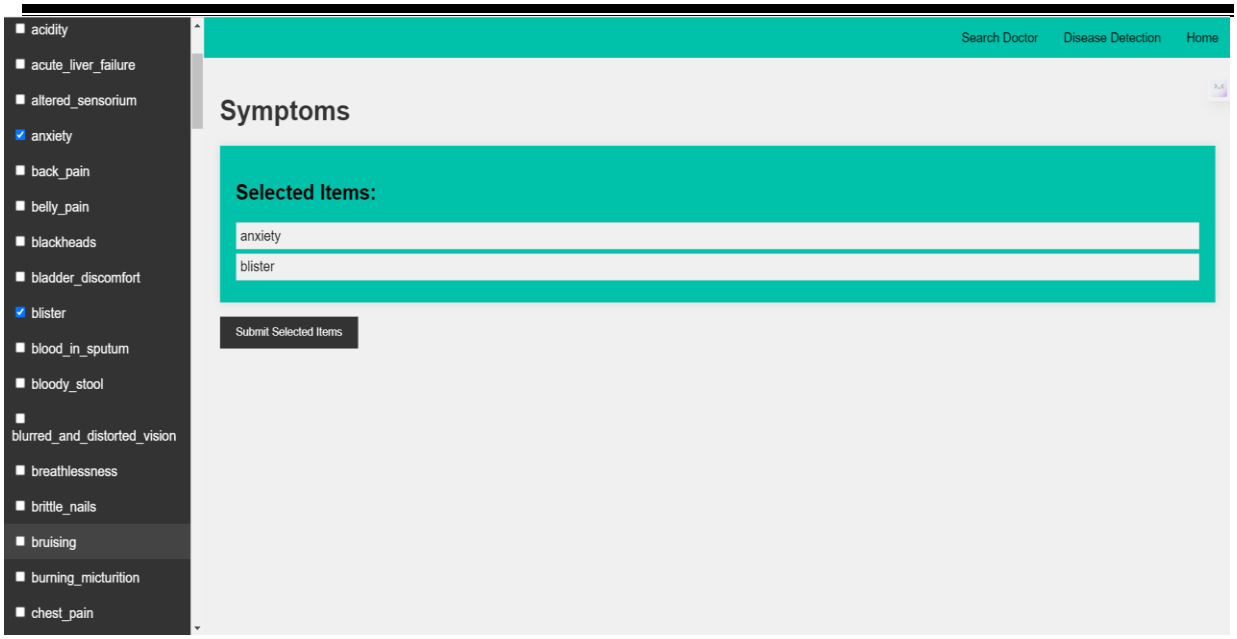
این الگوریتم در زبان پایتون یک کتابخانه متن-باز دارد که به راحتی پیاده‌سازی شده و سرعت بسیار بالایی دارد. مدل هوش مصنوعی بر اساس مجموعه داده شکل ۵-۷ یکبار آموزش داده می‌شود و بعد آن شی و تابع آن فراخوانی شده و در لحظه پاسخ درخواست سرویس تشخیص بیماری را می‌دهد.

برای ویژگی مقیاس‌پذیری برنامه، بارگذاری مدل مورد نظر در هنگام اجرای برنامه به دو حالت انجام می‌گردد. حالت اول به این شکل بوده که هر بار برنامه اجرا شود مدل مورد نظر آموزش ببیند. در این حالت این مزیت را دارد که هر زمان داده‌های مورد نظر تغییر کرد در زمان اجرا، مدل هوش مصنوعی داده های جدید را هم آموزش می‌بیند؛ ولی سرعت را کم کرده چرا که به ازای هر اجرا از برنامه یکبار مدل آموزش می‌بیند. البته لازم به ذکر است که سرعت الگوریتم ایکس‌جی‌بوست بسیار بالا بوده و مجموعه داده هم مقدار کمی دارد برای همین هر بار آموزش دیدن آن در حد چند ثانیه است. حالت دوم زمانی است که مدل آموزش دیده و خروجی مدل در فایلی ذخیره می‌شود که به ازای هر بار فراخوانی سرویس، مدل هوش مصنوعی فقط بارگذاری می‌شود و مجددا آموزش نمی‌بیند.

از آنجایی که مجموعه داده با ایستا بوده و تغییر نمی‌کند در پیاده‌سازی تشخیص بیماری از حالت دوم استفاده می‌کنیم و به ازای هر اجرا مدل را از پوشه محلی بارگذاری می‌کنیم.

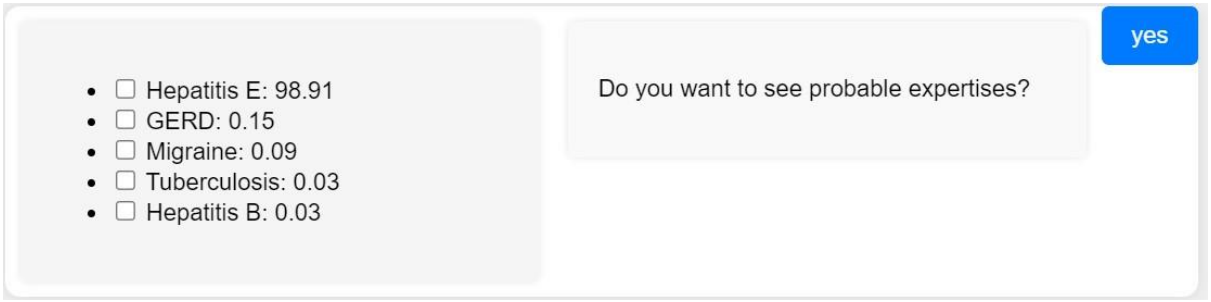
در نهایت کاربر از صفحه وب مربوط به این سرویس، علائم بیماری خود را مشابه شکل ۵-۷ انتخاب می‌کند.

[1] Gradient Boosted Trees
[2] Ensemble Algorithm
[3] Bias



شکل ۵-۷ تشخیص بیماری از علائم

همانطور که مشاهده می‌کنید علائمی که انتخاب شده‌اند در صفحه نمایش پیدا می‌کند و کاربر با زدن دکمه تایید می‌تواند خروجی مدل هوش مصنوعی و بیماری‌های احتمالی را ببیند. زمانی که بیماری ها نمایش داده می‌شود، شکل ۵-۸، یک سابقه در جدول کاربر-بیماری در پایگاه داده ذخیره می‌شود. کاربر می‌تواند از بیماری های نمایش داده، انتخاب کند که چه بیماری هایی را حدس می‌زند داشته باشد تا رابط نگاشت بیماری و تخصص فراخوانی شود. لازم به ذکر است تنها پنج تا از بیشترین مقادیر، به صورت مرتب شده، به کاربر نمایش داده می‌شود.



شکل ۵-۸ صفحه بیماری های محتمل

بعد از انتخاب کردن بیماری ها و تایید آن، رابط تشخیص تخصص فراخوانی می‌شود.

۵-۳-۳- سرویس تشخیص تخصص

ورودی این رابط، تعدادی بیماری است که بر اساس آن تخصص های احتمالی شناسایی شود. برای انجام دادن این کار نیاز است مجموعه داده ای داشته که بیماران و تخصص ها را مرتبط کند.

به‌دلیل آنکه مجموعه داده مورد نظر در اینترنت موجود نبود، در این پروژه به صورت میدانی داده‌ها را جمع آوری کردیم.

فرایند جمع‌آوری و تولید داده ها:

همانطور که گفته شد در داده‌های موجود نگاشتی بین نوع بیماری و تخصص‌های مربوط به آن وجود نداشت. برای همین نیاز شد که از متخصصین واقعی به صورت میدانی پرسیده شود که هر بیماری به چه تخصص‌هایی مربوط می‌شود. برای انجام دادن آن از سه نفر که در این حوزه متخصص بودند کمک گرفته شد. این سه نفر فارغ التحصیلان پزشکی عمومی از دانشگاه شهید بهشتی هستند که در حال حاضر برای اخذ مدرک تخصص دانشجو می‌باشند.

به هر یک از این متخصصین یک فایل داده شد که سطر های آن نوع بیماری و ستون‌های آن انواع تخصص‌های مختلف پزشکی وجود داشت. لازم به ذکر است که لیست تخصص‌های موجود با مشورت ایشان نوشته شد. سپس به ازای هر سطر (گونه بیماری) تعدادی ستون (گونه تخصص) وجود دارد که در آن خانه، میزان ارتباط بین بیماری مربوطه و تخصص‌ها قرار دارد.

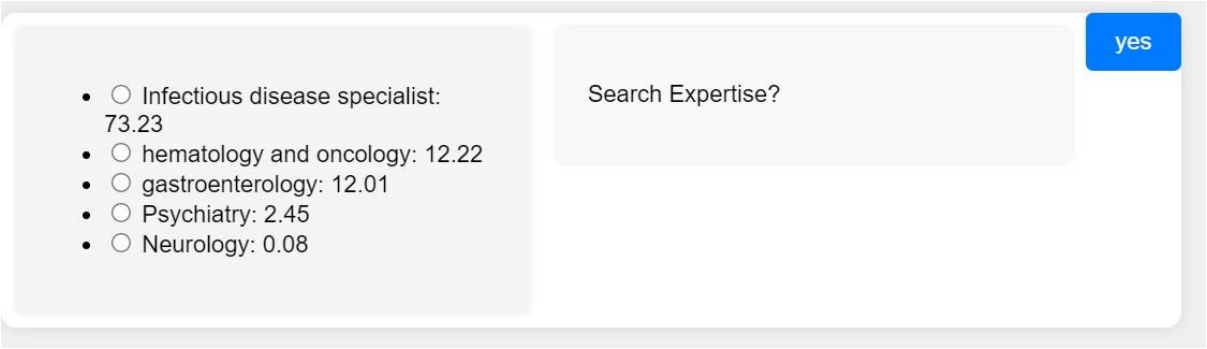
در واقع هر بیماری را به عنوان یک سطر، یک رکورد، نوشته و تخصص‌های پزشکی را به عنوان ستون در یک مجموعه داده جمع آوری کردیم. از سه متخصص عمومی پزشکی درخواست شد که به نگاشت بیماری و تخصص یک عدد نسبت دهند. سپس با میانگین گرفتن نظرات آنها و نرمال‌سازی داده‌ها به مجموعه داده مورد نظر رسیدیم. شکل ۵-۹ نمایانگر قسمتی از آن است.

Disease	Obstetrics and Gynecology	Neurology	ENT	orthopedics	nephrology	dermatology
Paroymsal Positiona	0	0.32854348	0.410848	0	0	0
AIDS	0.046948357	0	0	0	0	0
Acne	0.05952381	0	0	0	0	0.538584286
Alcoholic hepatitis	0	0	0	0	0	0
Allergy	0	0	0.157899	0	0	0.307271038
Arthritis	0	0	0	0.154768295	0	0
Bronchial Asthma	0	0	0.174545	0	0	0

شکل ۵-۹ قسمتی از مجموعه داده بیماری-تخصص

زمانی که این رابط از طریق رابط تشخیص بیماری فراخوانی شود، مقدار احتمال هر بیماری را در مقدار نگاشت مجموعه داده بیماری تخصص ضرب کرده، سپس مقادیر را جمع می‌کند و احتمال نهایی هر تخصص بدست می‌آید. با این روش که مبتنی بر توزیع احتمال توام[1] و ضرب ماتریسی[2] است، تخصص های کاندید بدست می‌آید و به کاربر نمایش داده می‌شود. در شکل ۵-۱۰ صفحه رابط کاربری و نتایج آن آورده شده است.

[1] Joint Probability Distribution
[2] Matrix Multiplication



شکل ۵-۱۰ رابط کاربری تخصص های احتمالی

از بین پنج نتیجه بدست آمده، کاربر فقط می‌تواند یکی از تخصص ها را انتخاب کند. اگر بعد از انتخاب تخصص مورد نظر دکمه تایید را بزند سرویس جستجوی پزشک فراخوانی می‌شود.

صحت سنجی داده‌ها و نقاط حساس:

به دلیل اینکه این مجموعه داده در فرایند تولید این پروژه انجام گردید، نیاز است داده‌های آن ارزیابی شوند. می‌دانیم این امکان وجود دارد که پیشفرض‌های ذهنی و تجربیات شخصی متخصصین که نگاشت بین تخصص و بیماری را انجام داده‌اند، متفاوت باشد. برای همین هر چه تعداد متخصصین بالاتر رود، خرد جمعی حکم می‌کند که دقت تشخیص تخصص هم بالاتر رود.

از طرفی براساس مجموعه‌داده جمع‌آوری شده توسط این سه نفر مشاهده شد در مواردی اختلاف نظر وجود دارد. برای همین تصمیم گرفته شد که اختلاف نظر آنها ارزیابی گردد و در داده‌هایی که این اختلاف پر رنگ است، به اصطلاح آنها را داده‌های حساس می‌نامیم، این موضوع بررسی شود.

برای یافتن اختلاف نظر آنها از روش انحراف از معیار[1] استفاده می‌کنیم. انحراف از معیار به این شکل محاسبه می‌شود که ابتدا میانگین محاسبه می‌گردد. سپس برای هر نقطه داده، مربع فاصله آن تا میانگین پیدا می‌شود. این مجموع مقادیر بر میانگین تقسیم شده و با جذر آن به انحراف از معیار می‌رسیم. ویژگی انحراف از معیار این است که متوجه می‌شویم نظر هر شخص چقدر از میانگین فاصله دارد.

از آنجایی که تعداد کسانی که مشارکت کرده‌اند بالا نبوده و سه نفر هستند، عملا انحراف از معیار عدد بالایی نمی‌گیرد. شکل ۵-۱۱ نمایانگر نمونه‌ای از جدول انحراف از معیار داده‌ها و اختلاف نظر متخصصین می‌باشد.

[1] Standard Deviation

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Disease	cs and Gyn	Neurology	ENT	rthopedic	ephrolog	ermatolog	hthalmol	troenterol	cardiology	ndocrinolo	Psychiatry	ulmonolog	logy and o	s disease
2	Paroymsal Positiona	0.00	0.12	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.17
3	AIDS	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.13	0.26
4	Acne	0.08	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.17
5	Alcoholic hepatitis	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.05
6	Allergy	0.00	0.00	0.02	0.00	0.00	0.08	0.00	0.00	0.00	0.04	0.00	0.15	0.08	0.04
7	Arthritis	0.00	0.00	0.00	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.07
8	Bronchial Asthma	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.07	0.00	0.01	0.17	0.00	0.13
9	Cervical spondylosis	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09
10	Chicken pox	0.00	0.00	0.00	0.00	0.00	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.12
11	Chronic cholestasis	0.00	0.00	0.00	0.00	0.15	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.08	0.04
12	Common Cold	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.07
13	Dengue	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.18	0.00	0.00	0.01	0.00	0.00	0.18
14	Diabetes	0.00	0.07	0.00	0.00	0.07	0.00	0.07	0.00	0.04	0.24	0.01	0.00	0.00	0.00
15	rphic hemmorhoids(	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.00	0.00	0.01	0.00	0.00	0.08
16	Drug Reaction	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.10
17	Fungal infection	0.05	0.00	0.05	0.00	0.00	0.16	0.00	0.00	0.00	0.00	0.01	0.03	0.00	0.29
18	GERD	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.00	0.11	0.00
19	Gastroenteritis	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.15
20	Heart attack	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.20	0.00	0.01	0.04	0.08	0.00

شکل ۵-۱۱ نمونه‌ای از جدول اختلاف نظر متخصصین

همانطورکه دیده می‌شود به طور مثال داده شماره O17 یعنی بیماری Fungal infection در تخصص عفونی یک داده حساس می‌باشد که انحراف از معیار آن ۰.۲۹ است. پس در صورتی که یکی از بیماری‌های تشخیص داده شده در سرویس تشخیص بیماری، Fungal infection باشد؛ در صورتی که تخصص عفونی تشخیص داده شد باید احتیاط کرد.

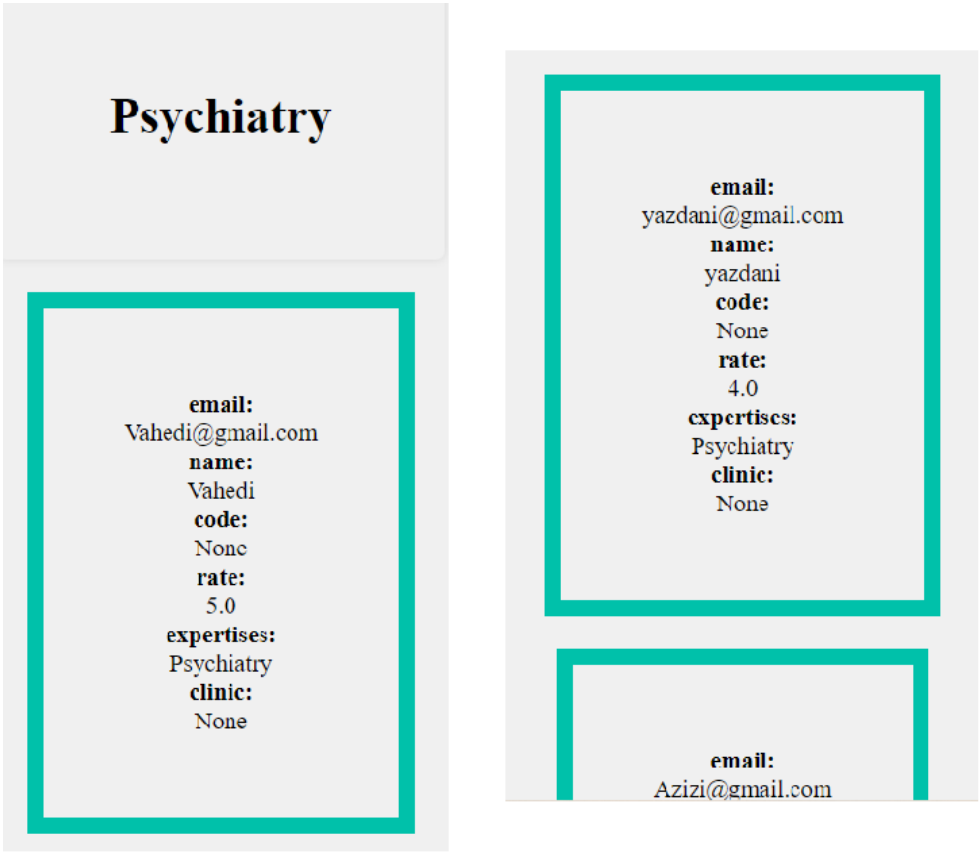
می‌توان در این داده‌ها بررسی مجدد انجام داد و از متخصصین خواسته شود که در این باره بیشتر صحبت کنند و مجموعه داده مورد نظر اصلاح شود.

۵-۳-۴- سرویس جستجوی پزشک

سرویس جستجوی پزشک، تخصص مورد نظر را به عنوان ورودی گرفته و لیستی از پزشکان با آن تخصص را خروجی می‌دهد. این امکان وجود دارد که با رابط کاربری یا درخواست از طریق سرویس دیگر فراخوانی شود. در واقع از جدول پزشک موجود در پایگاه داده این اطلاعات را پرس و جو می‌کند. سرویس جستجوی پزشک در دو حالت کار می‌کند؛ حالت اول زمانی است که با درخواست پست[1] تخصص مورد نظر را از سرویس تشخیص بیماری می‌گیرد، و حالت دومی زمانی که کاربر به صورت جداگانه از این سرویس استفاده کرده و با رابط کاربری، درخواست گت[2]، تخصص مورد نظر را می‌گیرد و خروجی می‌دهد.

شکل ۵-۱۲ نمایانگر لیست پزشکان دارای تخصص روانپزشکی است که بر اساس امتیاز دکتر مرتب شده است.

[1] POST
[2] GET



شکل ۵-۱۲ لیست پزشکان بدست آمده از سرویس جستجوی پزشک

۵-۴- ابزارهای پیاده‌سازی

در این قسمت به ابزار هایی که برای پیاده‌سازی به‌کار گرفته شده، می‌پردازیم. لازم به ذکر است به طور مختصر در مورد چرایی انتخاب آنها توضیح داده شده به شکلی که در چارچوب موضوعات این پایان‌نامه باشد.

۵-۴-۱- زبان‌های برنامه‌نویسی و کتابخانه‌ها

نرم‌افزار با زبان برنامه‌نویسی پایتون[1] نوشته است. این زبان کتابخانه‌ها و چارچوب‌های بسیار قدرتمندی داشته که به‌راحتی پیاده‌سازی می‌شود. چارچوب استفاده شده برای مدیریت میکروسرویس و اجرای برنامه فلسک[2] می‌باشد. این چارچوب کتابخانه های درونی زیادی دارد که نشست‌ها[3]، ورود کاربر، ارتباط با پایگاه داده و غیره

[1] Python
[2] Flask
[3] Session

را آسان می‌کند. ارتباط با پایگاه داده با نگاشت شی-رابطه‌ای[1] کتابخانه اس‌کیوال‌الکمی[2] انجام می‌گردد.

برای فرانت‌اند[3] و صفحات رابط کاربری از زبان‌های اچ‌تی‌ام‌ال، سی‌اس‌اس و جاوا اسکریپت[4] استفاده می‌شود.

سرویس تشخیص بیماری، که در ادامه بیشتر به آن پرداخته می‌شود، با الگوریتم ایکس‌جی‌بوست پیاده سازی شده که در کتابخانه های زبان پایتون قرار دارد.

۵-۴-۲- داکر[5]

داکر ابزاری متن-باز است که با استفاده از آن می‌توان فرایند ایجاد، پیاده سازی و اجرای برنامه‌ها را با استفاده از کانتینر[6] ها بسیار ساده کرد. در واقع نوعی ماشین مجازی است و این امکان را برای برنامه‌ها فراهم می‌کند تا از هسته لینوکس استفاده کرده و از امکاناتی بهره مند شوند که در سیستم عامل میزبان ارائه نشده است. به این ترتیب می‌توانند به صورت مستقل از پیش نیازها و امکانات مازاد بهره برداری کنند. این موضوع باعث می‌شود سرعت و عملکرد برنامه بهبود قابل ملاحظه‌ای پیدا کند و حجم آن نیز کاهش یابد. از طرفی نیازی به نصب کتابخانه‌ها و زبان‌های برنامه نویسی نمی‌باشد و همه برنامه‌ها در آن نصب می‌گردد.

۵-۵- خلاصه فصل

در این قسمت ابتدا روش پیاده‌سازی و ابزارهای استفاده شده توضیح داده شد. سپس به اجزای سامانه، به طور خاص سرویس‌های طراحی شده، پرداخته شد. سامانه به سه سرویس کوچکتر تجزیه شده که هر کدام وظیفه متفاوت و نقش مستقلی دارند. سرویس‌ها عبارتند از: سرویس کاربر، سرویس تشخیص بیماری، سرویس تشخیص تخصص، سرویس جستجوی پزشک. نمودارها و تصاویر مربوط به طراحی و اجرای برنامه در هر زیربخش قرار دارد تا نمایانگر بُعد جامعی از این سامانه باشد. هدف اصلی استفاده از این معماری در این پروژه تجزیه سامانه به بخش‌های کوچکتر بود که با طراحی و پیاده‌سازی آن محقق گردید.

[1] ORM: Object Relational-Mapping
[2] SQL Alchemy
[3] Front-end
[4] HTML / CSS / JavaScript
[5] Docker
[6] Container

فصل ششم: ارزیابی سامانه

در این بخش ارزیابی سامانه انجام می‌گردد. به طور کلی سه معیار اصلی در ارزیابی قرار گرفته است: کیفیت کد، آزمون سرویس‌ها، زمان پاسخ مدل هوش مصنوعی و سامانه. در این زیر بخش‌ها آزمون سرویس‌ها از اهمیت بالاتری برخوردار است.

لازم به ذکر است این آزمون‌ها به نوعی تعریف شده که پاسخگوی نیازمندی‌های سامانه هدف که در فصل چهارم گفته شده، باشد.

۶-۱- کیفیت کد

ابزارهای زیادی در دنیای نرم‌افزار برای ارزیابی کیفیت کد وجود دارند، یکی از ابزار های متن-باز پرطرفدار برای بررسی کد پایتون، برنامه فلیک‌ایت[1] می‌باشد. این برنامه قادر است هم به صورت کتابخانه در قسمتی از کد و برنامه قرار گیرد و هم از قسمت ترمینال و محیط مجازی[2] کیفیت کد را بررسی کند.

در زمان توسعه و پیاده‌سازی سامانه بعد از اتمام هر بخش، غالبا بعد از اتمام هر سرویس، این برنامه اجرا می‌شد. با اجرای این برنامه چند خطا از جنس خطا های کد تمیز[3] مشاهده شد که بعد از اصلاح آنها و اجرای مجدد برنامه، خروجی این ابزار نشان داد کد از کیفیت مناسبی برخوردار است و مورد جدیدی را گزارش نکرد.

طبق گزارش گیت‌هاب[4] حدود ۶۰٪ سامانه مورد نظر با زبان برنامه‌نویسی پایتون پیاده‌سازی شده است، بنابراین استفاده از این ابزار در این پروژه منطبق با فناوری‌های آن می‌باشد.

۶-۲- آزمون سرویس‌ها

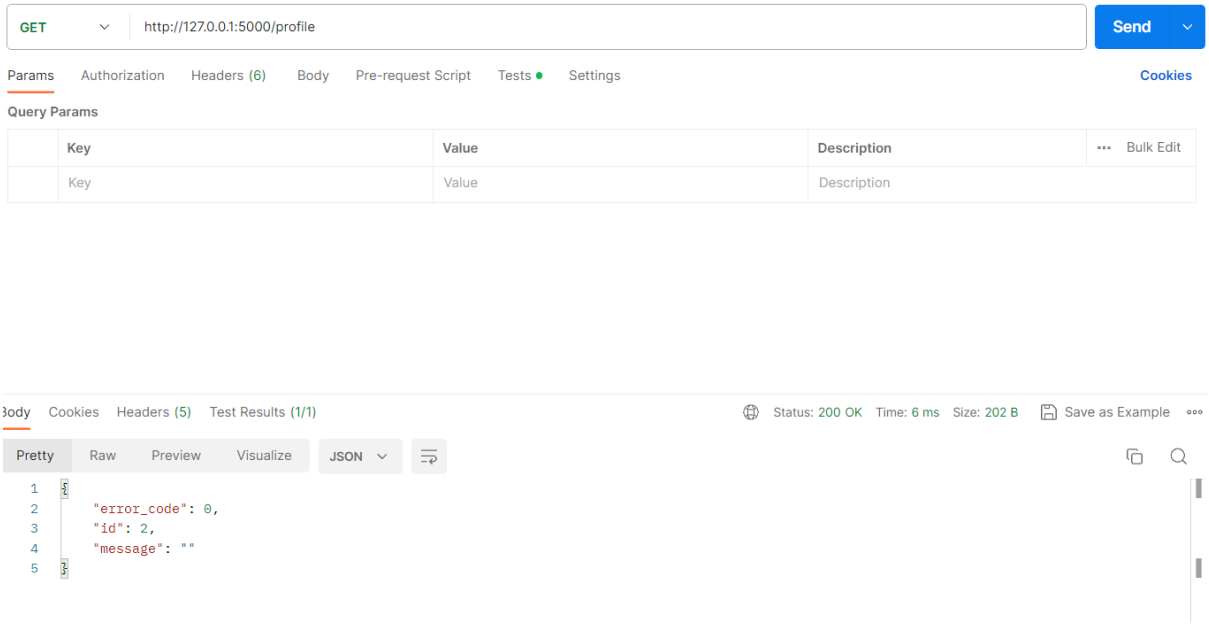
برای بررسی و آزمون میکروسرویس می‌توان از روش‌های مختلف بهره گرفت. یکی از آزمون‌های رایج استفاده از رابط سرویس و درخواست های ای‌پی‌ای و بررسی خروجی آن است. نرم‌افزار پست‌من[5] فضایی را آماده می‌کند که به راحتی به یوآرال[6] مربوطه درخواست زده و پاسخ آنرا دریافت کرد. به طور کلی درخواست های GET برای زمانی است که بارگذاری صفحه وب آمده باشد، درخواست POST درخواستی است بدنه داشته یا اطلاعات را از فرم های رابط کاربری گرفته باشد.

۶-۲-۱- سرویس کاربر

[1] Flake8
[2] Virtual Environment
[3] Clean Code
[4] Github
[5] Postman
[6] URL

در قسمت پروفایل کاربر، وقتی کاربر درخواست را می‌زند یک جیسان[1] که حاوی شناسه کاربر است بازگردانده می‌شود. در شکل ۶-۱ خروجی درخواست زیر قرار دارد.

GET Request: http://127.0.0.1:5000/profile



شکل ۶-۱ درخواست پروفایل کاربر

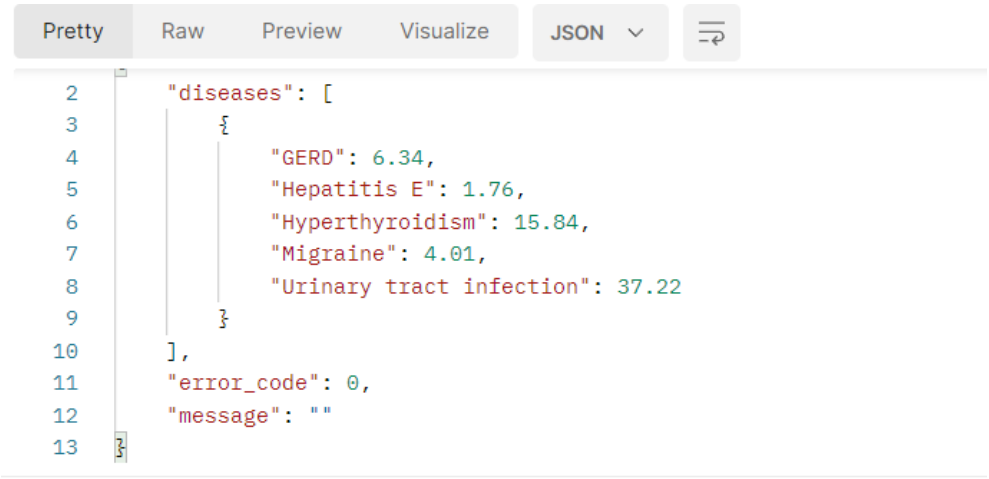
۶-۲-۲- سرویس تشخیص بیماری

در سرویس تشخیص بیماری، علائم به رابط مورد نظر داده شده و خروجی بیماری‌ها نشان داده می‌شود. شکل ۶-۲ خروجی آنرا نشان می‌دهد.

POST Request: http://127.0.0.1:5001/disease

Input: ['abnormal_menstruation', 'acidity', 'acute_liver_failure', 'bladder_discomfort']

[1] Json



شکل ۶-۲ درخواست رابط تشخیص بیماری

۶-۲-۳- رابط ارجاع به متخصص

در سرویس تشخیص بیماری، به رابط ارجاع به متخصص یک لیست از بیماری‌ها به همراه مقادیر احتمالی آن داده شده و به عنوان خروجی، تخصص های مرتبط، مشابه شکل ۶-۳، نمایش داده می‌شود.

POST Request: http://127.0.0.1:5001/expertise

Input: 'Urinary tract infection': 37.22, 'Hyperthyroidism': 15.84, 'GERD': 6.34, 'Migraine': 4.01, 'Hepatitis E': 1.76}

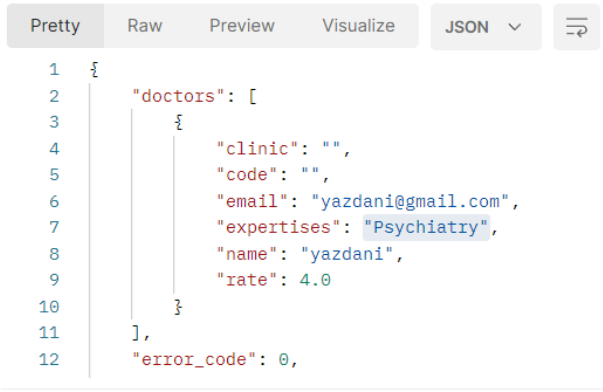
شکل ۶-۳ درخواست رابط ارجاع به متخصص

۶-۲-۴- سرویس جستجو

در سرویس جستجوی متخصص، ورودی یکی از تخصص‌های پزشکی است و خروجی لیستی از پزشکان می-باشد. در شکل ۶-۴ یک نمونه از خروجی آمده است.

POST Request: http://127.0.0.1:5002/search

Input: "Psychiatry"



شکل ۶-۴ درخواست رابط سرویس جستجو

۶-۳- زمان پاسخ

یکی دیگر از مواردی که می‌توان در ارزیابی سامانه در نظر گرفت، عملکرد سیستم بر اساس زمان‌های صرف شده و زمان پاسخ می‌باشد. البته معماری میکروسرویس تنها معماری است که برای این سامانه طراحی و پیاده‌سازی شده، بنابراین معیار مقایسه دقیقی مبنی بر کم یا زیاد بودن زمان صرف شده وجود ندارد و به این روش ارزیابی نقد وارد است. ولی همچنان با ذکر این زمان‌ها می‌توان شهود خوبی نسبت به عملکرد سامانه داشت. برخی از زمان‌های قابل ارزیابی سامانه عبارتند از:

- زمان استقرار بر سرور: در حالتی که در چارچوب فلسک اجرا شود، ۲ ثانیه و زمانی که با داکر اجرا شود ۴ ثانیه می‌باشد.
- زمان صرف شده بین درخواست سرویس ارجاع به متخصص و پاسخ سرویس جستجوی پزشک تقریبا ۲ ثانیه است.
- ساخت شی ایکس‌جی‌بوست و تشخیص بیماری در حالتی که مدل را آموزش ببیند حدود ۵ ثانیه و در حالتی که فقط بارگذاری انجام دهد، ۱.۵ ثانیه می‌باشد.

۶-۴- نتیجه ارزیابی

با ارزیابی‌های صورت گرفته مشاهده می‌شود اگرچه سرویس‌ها مستقل بوده و وظایف متفاوتی دارند، ولی در کنار هم به‌درستی کار می‌کنند و از نظر عملکرد هم مناسب می‌باشند. از طرفی هر کدام از سرویس‌ها به طور جداگانه کاربرد دارند.

همانطور که در فصل چهارم، نیازمندی‌های عملکردی و غیر عملکردی سامانه ذکر شد، با انجام دادن این آزمون ها همگی به نتایج خوبی رسیدند و سامانه پاسخگوی نیازمندی‌های ما بود.

ویژگی‌های گفته شده مفاهیم مقیاس‌پذیری، تحویل مستمر، اتصال سست، مسئولیت واحد و استقلال اجزای کوچکتر برنامه را تصدیق می‌کند؛ همچنین نشان می‌دهد معماری میکروسرویس انتخاب خوبی برای این سامانه می‌باشد.

فصل هفتم: جمع‌بندی و پیشنهادات

۷-۱- جمع‌بندی و نتیجه‌گیری

همانطور که در ابتدای این پایان‌نامه ذکر شد، انگیزه اصلی انجام این پروژه طراحی و نمونه‌سازی یک سامانه کاربردی و قابل استفاده بود. می‌دانیم پیاده‌سازی یک سامانه پزشکی در حدی که نیازهای کاربر نهایی را رفع کند و به یک محصول قابل عرضه برسد، کار آسانی نیست و فراتر از پروژه کارشناسی می‌باشد؛ ولی با استفاده از مفاهیم معماری نرم‌افزار و معماری میکروسرویس این قابلیت وجود دارد که چند نفر به طور مستقل بر یک سامانه واحد کار کنند و خروجی مناسبی بگیرند. فلسفه معماری میکروسرویس جداسازی اجزای بزرگ سامانه به سرویس‌های کوچکتر است، برای همین مفاهیمی مانند مقیاس‌پذیری، اتصال سست، ایزوله‌سازی خطا، عدم تمرکز سامانه، تحویل مستمر و چابکی اهمیت زیادی پیدا می‌کند.

در این پروژه سامانه پزشکی تشخیص بیماری و ارجاع به متخصص طراحی شد. به طور کلی سامانه به سه بخش کوچکتر اعم از سرویس مدیریت کاربر، سرویس تشخیص بیماری و ارجاع به متخصص، و سرویس جستجوی پزشک تقسیم شد. همچنین این قابلیت فراهم شد که با روش های مبتنی بر هوش مصنوعی، کاربر بتواند با دانستن علائم خود به متخصص مناسب رجوع کند و از سردرگمی خود بکاهد. با نمونه‌سازی و ارزیابی این سامانه، ثابت گردید مفاهیم ذکر شده در معماری میکروسرویس، نظیر مقیاس‌پذیری و استقلال بخش‌های مختلف سامانه به‌درستی نمود پیدا می‌کنند.

معماری نرم‌افزار بخش مهمی از مهندسی نرم‌افزار می‌باشد. تصمیم‌های مربوط به طراحی می‌تواند حوزه‌های دیگر نظیر فرایند توسعه نرم‌افزار، عملکرد و نگهداری برنامه، ابزارهای فنی، روش‌های پیاده‌سازی و مدیریت و پروژه را تحت تاثیر قرار دهد، برای همین باید دقت و توجه بسیار زیادی به آن شود.

اگرچه امروزه معماری میکروسرویس از سبک‌های پرکاربرد و محبوب معماری نرم‌افزار است، ولی معایب و چالش‌هایی دارد که می‌تواند هزینه‌بر باشد. برای همین باید برای انتخاب معماری نرم‌افزار و طراحی سامانه، زمان و هزینه زیادی صرف شود و متناسب با قابلیت های سامانه مورد نظر این تصمیم اخذ شود.

۷-۲- پیشنهادات و کارهای آینده

هدف از این پروژه طراحی و نمونه سازی سامانه تشخیص بیماری و ارجاع به متخصص با معماری میکروسرویس است و استاندارد های لازم برای رسیدن به محصول قابل استفاده برای کاربر را ندارد چرا که در حد نمونه سازی می‌باشد. برای اینکه به محصول قابل عرضه برسد نیاز است به ابعاد بیشتری توجه شود.

در آینده می‌توان به موارد زیر پرداخت:

۱. بهبود مدل هوش مصنوعی: همانطور که گفته شد، سرویس تشخیص بیماری علائم بیمار را به عنوان ورودی گرفته و بیماری های احتمالی را خروجی می‌دهد. از آنجایی که داده های مورد نظر همگی متنی بوده و انواع مختلف بیماری زیاد نبوده است، این قابلیت وجود دارد که داده ها گسترده تر شده و مدل های پیچیده تری استفاده شوند. در آن صورت نیاز است عملکرد و دقت مدل مجددا بررسی شود.

به غیر از اضافه کردن داده های بیشتر، قابلیت یادگیری آنلاین و یادگیری مستمر هم دقت مدل را بالاتر می‌برد که روش های یادگیری تقویتی برای این ویژگی توصیه می‌شود.

۲. امنیت میکروسرویس ها: یکی از مواردی که امروزه دنیای نرم افزار ها را تحت تاثیر قرار داده و در طراحی باید در نظر گرفته شود، امنیت سامانه می‌باشد. همانطور که گفته شد سرویس ها با رابط های متفاوت در ارتباط هستند و در صورتی که به آنها درخواست زده شود، پاسخ می‌دهند. سامانه باید توانایی مدیریت درخواست ها نسبت به دسترسی های آنها را داشته باشد. سطح دسترسی ها باید در لایه پایگاه داده، درخواست های رابط سرویس ها، مدیریت داکر، مدیریت تعداد درخواست های یک یا چند کاربر باشد.

۳. اضافه کردن ویژگی های بیشتر در سامانه: معماری سامانه به نوعی طراحی شده است که مقیاس پذیری بالایی داشته باشد و در آینده بتوان آنرا بهبود مستمر داد. جزئیات بیشتر سامانه می‌تواند برای کاربر نهایی سامانه را کاربردی تر و جذاب تر کند. پیاده سازی هر یک از موارد زیر سامانه را ارتقا می‌دهد:

- بهبود رابط کاربری و گرافیک صفحات
- اضافه کردن صفحه مجله سلامت برای خواندن مقالات پزشکی مرتبط به بیماری ها
- اضافه کردن قابلیت نوبت دهی پزشکی برای کاربران بر اساس تخصص هر دکتر
- اضافه کردن قابلیت پیشنهاد دادن درمان های خانگی برای بهبود مریضی

مراجع

[1] Jamshidi, Pooyan & Pahl, Claus & Mendonça, Nabor & Lewis, James & Tilkov, Stefan. Microservices: The Journey So Far and Challenges Ahead. IEEE Software. 35. 24-35, 2018.

[2] X. Wang, Y. Chen, J. Wang and A. Yu, "Design of Doctor-Patient Management Platform Based on Microservice Architecture," 2020 2nd International Conference on Applied Machine Learning (ICAML), Changsha, China, pp. 99-102, 2020.

[3] "پلتفرم آموزش پزشکی آنلاین مدیست," Medist.ir, Accessed on: Jul. 15, 2023. Available: https://medist.ir/.

[4] "دکتر ساینا - مرجع آموزش پزشکی و سلامت," Drsaina.com, Accessed on: Jul. 15, 2023. Available: https://www.drsaina.com/.

[5] "نوبت دهی اینترنتی پزشکان و مراکز درمانی," Nobat.ir, Accessed on: Jul. 15, 2023. [Online]. Available: https://nobat.ir/.

[6] "Online Booking System for Doctors, Patients & Clinics," HotDoc, Accessed on: Jul. 15, 2023. Available: https://www.hotdoc.com.au/.

[7] "Medical Practice Management Software," Medesk, Accessed on: Jul. 15, 2023. Available: https://www.medesk.net.

[8] "Health Information, Resources, Tools & News Online," WebMD, Accessed on: Jul. 15, 2023. Available: https://www.webmd.com.

[9] Richards, Mark, and Ford, Neal. Fundamentals of Software Architecture: An Engineering Approach. Japan, O'Reilly Media, Incorporated, 2020.

[10] "Microservices Architecture", Microservices.io, Accessed: October 10, 2023. Available: https://microservices.io.

[11] Sommerville. Software Engineering. 10th Edition, Pearson Education Limited, Boston, 2016

[12] Pranay Patil, Disease Symptom Prediction, Accessed: October 10, 2023. Available: https://www.kaggle.com/datasets/itachi9604/disease-symptom-description-dataset.

Abstract

Applications in the field of healthcare have developed according to the user's needs and their capabilities have increased. Accurate disease diagnosis with methods based on artificial intelligence is one of the up-to-date issues of these systems. Apart from the diagnosis of the disease, in many cases the patient does not know which doctor he should consult with which specialty. This confusion may lead to referring to an unrelated specialist and waste a lot of time.

Software architecture has always been one of the important issues in design and development. Microservices architecture, as an indicative model of software architecture, which was created with the philosophy of separating large system components into smaller services, offers many solutions for design and implementation. In this project, the healthcare system for disease diagnosis and referral to a specialist has been designed and prototyped using microservice architecture. The system is divided into small and independent services in a way that is scalable and can be developed and maintained in the future. In this project, the user can interact with the program using the user interface and meet his needs by connecting to the services. The implemented services are: user login service, disease diagnosis service, specialty diagnosis service, specialist search service. Each of the services has different capabilities that will be discussed below.

In the end, with the tools and evaluation methods and their matching with the system requirements, relying on the principles of software engineering, it can be seen that the microservice architecture and the technologies used are a suitable option for the design of the desired system.

Key Words: Software Architecture, Microservices Architecture, Healthcare System, Disease Diagnosis

Amirkabir University of Technology
(Tehran Polytechnic)

Department of Computer Engineering

BSc Thesis

Designing and prototyping disease diagnosis and physician referral system using microservice architecture

By
Pouyan Hessabi

Supervisor
Dr. Amir Kalbasi

October 2023