واحد شهرقدس

گروه کارشناسی کامپیوتر

Task Scheduler with Web Dashboard

رضا آل یعقوب

40125241054012

پایان نامه کارشناسی

در رشتهٔ

مهندسی کامپیوتر – نرم افزار

استاد راهنما:

دکتر حقی کاشانی

خرداد 1404

بسم الله الرحمن الرحيم

این پروژه را با افتخار و احترام تقدیم می‌کنم به:

خانواده عزیزم، که با حمایت بی‌وقفه و عشق بی‌پایانشان، نیروی محرکه‌ای برای ادامه مسیرم در این سفر علمی و حرفه‌ای بوده‌اند. تشویق‌های شما در لحظات دشوار و باور بی‌چون‌وچرایتان به توانایی‌هایم، این دستاورد را ممکن ساخت.

استاد گرانقدرم، دکتر مصطفی حقی کاشانی **، که با راهنمایی‌های حکیمانه و بازخوردهای سازنده خود، نه‌تنها کیفیت این پروژه را ارتقا بخشیدند، بلکه مرا به سوی تعالی علمی هدایت کردند.**

و دوستان و همکلاسی‌هایم در دانشکده مهندسی کامپیوتر دانشگاه آزاد اسلامی واحد قدس، که با همراهی و تبادل دانش خود، این مسیر را برایم دلپذیرتر ساختند.

این اثر، نشانه‌ای کوچک از سپاس من به همه کسانی است که در این راه همراهم بودند.

چکیده

این پروژه یک راهکار پیشرفته برای پشتیبانی از پردازش‌های زمان‌بندی‌شده چندرسانه‌ای ارائه می‌دهد که به‌صورت یکپارچه با پایگاه داده MongoDB و یک رابط کاربری پویا تلفیق شده است. این سیستم برای شبیه‌سازی قابلیت‌های بلادرنگ در یک محیط تولیدی طراحی شده و ویژگی‌های کلیدی نظیر ادغام فایل‌های رسانه‌ای، زمان‌بندی وظایف برای دستگاه‌های خاص، اولویت‌بندی وظایف، و یک سامانه گزارش‌گیری قدرتمند را در بر دارد. با توجه به محدودیت‌های محرمانگی، نسخه‌ی نمایشی ارائه‌شده از معماری خطی مبتنی بر Flask استفاده می‌کند، اما عملکردهای اصلی یک سامانه microservices محور در محیط شرکتی را که در اصل با FastAPI و خدمات containerized طراحی شده، به‌خوبی بازتولید می‌کند.

این نرم‌افزار فرآیندهای پیچیده‌ی چندرسانه‌ای را به‌صورت خودکار انجام می‌دهد؛ مانند ترکیب تصاویر با صدا برای تولید فایل‌های ویدئویی، در حالی که از سیاست‌های زمان‌بندی سفارشی‌سازی‌شده برای دستگاه‌های سخت‌افزاری مختلف پشتیبانی می‌کند. فناوری‌های کلیدی شاملPython ، Flask، MongoDB، Redis، threading، MoviePyو FFmpeg هستند که به‌دلیل پایداری و سازگاری آن‌ها با پردازش‌های چندرسانه‌ای انتخاب شده‌اند. نسخه‌ی نمایشی اصول معماری حیاتی نظیرmodularity ، اجرای همزمان(concurrency) ، تحمل خطا (fault tolerance) و مدیریت کاربران را حفظ کرده و آن‌ها را از طریقAPI های RESTfulو یک داشبورد وب شهودی ارائه می‌دهد. این سامانه با استفاده از قابلیت‌هایی مانند مکانیزم‌های تلاش مجدد، صف‌های وظیفه ایمن در برابر تداخل thread-safe task) (queues، و گزارش‌گیری ماندگار(persistent logging) ، مقیاس‌پذیری و پایداری را تضمین کرده و به چالش‌های واقعی در خودکارسازی توزیع‌شده‌ی رسانه پاسخ می‌دهد.

این گزارش به تفصیل به معماری سیستم، راهبردهای پیاده‌سازی، چالش‌های موجود و بهبودهای احتمالی می‌پردازد. همچنین تأکید دارد بر به‌کارگیری اصول مهندسی نرم‌افزار مانند separation of concerns، asynchronous processing، و data persistence برای ایجاد یک راهکار قابل نگهداری و مقیاس‌پذیر. این پروژه یک دستاورد مهم علمی و حرفه‌ای محسوب می‌شود که مفاهیم نظری را به کاربردهای عملی پیوند می‌زند. با شبیه‌سازی استانداردهای صنعتی، به نیاز فزاینده‌ی پردازش خودکار رسانه‌های مبتنی بر دستگاه در حوزه‌هایی نظیر تابلوهای دیجیتال، آموزش و اینترنت اشیاء پاسخ می‌دهد. این کار همچنین ظرفیت مشارکت در پروژه‌های open-sourceرا برجسته کرده و چارچوبی قابل توسعه برای توسعه‌های آینده در خودکارسازی چندرسانه‌ای ارائه می‌کند.

تقدیر و تشکر

صمیمانه‌ترین سپاس‌گزلرم از استاد راهنمای پروژه‌ام، دکتر مصطفی حقی کاشانی، برای راهنمایی‌های پیوسته، بازخوردهای دقیق و تشویق‌های ارزشمندشان در طول توسعه این پروژه. تخصص ایشان در مهندسی نرم‌افزار و معماری سیستم نقش کلیدی در بهبود پیاده‌سازی فنی و تضمین وضوح و انسجام گزارش ایفا کرد. راهنمایی‌های ایشان نه‌تنها کیفیت این کار را ارتقاء داد بلکه الهام‌بخش من در مواجهه خلاقانه و جدی با چالش‌ها بود.

عمیقاً سپاسگزارم از پرتوتاپ رایان که این فرصت را فراهم کرد تا در یک محیط سازمانی با سامانه‌ای در سطح تولید کار کنم. این تجربه من را با چالش‌های دنیای واقعی مانند بهینه‌سازی پردازش وظایف ناهمگام، مدیریت میکروسرویس‌ها در سیستم‌های توزیع‌شده آشنا کرد. اگرچه محدودیت‌های محرمانگی مانع از ارائه کامل سیستم تولیدی شد، اما آموخته‌های این تجربه نقش مهمی در شکل‌دهی نسخه‌ی نمایشی داشت و درک من از مفاهیم مدرن DevOps مانند containerization و طراحی API را عمیق‌تر ساخت.

از صمیم قلب از خانواده و دوستانم سپاسگزارم که در طول جلسات طولانی کدنویسی شبانه و فرآیندهای طاقت‌فرسای اشکال‌زدایی، همواره پشتیبان من بودند. تشویق‌های آنان انگیزه‌ام را حفظ کرد. همچنین از اساتید و هم‌کلاسی‌هایم در گروه مهندسی نرم‌افزار دانشگاه آزاد اسلامی واحد قدس، قدردانی می‌کنم که محیطی علمی و مشارکتی فراهم کردند. دانش به‌اشتراک‌گذاشته‌شده و گفت‌وگوهای سازنده با آن‌ها دیدگاه من را نسبت به طراحی و توسعه سیستم‌ها غنی‌تر ساخت.

فهرست مندرجات

فهرست شکلها

فهرست جداول

مقدمه

پیش‌زمینه

با پیشرفت سریع فناوری‌های دیجیتال و افزایش اهمیت محتوای چندرسانه‌ای در حوزه‌های مختلف از جمله تبلیغات دیجیتال، آموزش، اطلاع‌رسانی عمومی، و اینترنت اشیا(IoT) ، نیاز به سیستم‌های خودکار برای مدیریت و پردازش محتوای صوتی و تصویری به شدت احساس می‌شود. گزارش‌های جهانی نشان می‌دهند که بازار چندرسانه‌ای در سال 2023 ارزشی بالغ بر 3.2 تریلیون دلار داشته و پیش‌بینی می‌شود تا سال 2030 با نرخ رشد سالانه 7.2 درصد گسترش یابد .[Statista, 2023] این رشد نتیجه‌ی تقاضای روزافزون برای محتوای پویا و جذاب در کاربردهایی مانند نمایشگرهای دیجیتال، سیستم‌های اطلاع‌رسانی، و دستگاه‌های هوشمند است.

روش‌های سنتی پردازش رسانه، مانند استفاده از ابزارهای دستی نظیر Adobe Premiere یا Audacity، به دلیل نیاز به دخالت انسانی، زمان‌بر، مستعد خطا، و برای مدیریت حجم بالای وظایف در محیط‌های توزیع‌شده ناکارآمد هستند. برای مثال، به‌روزرسانی دستی محتوای تبلیغاتی در صدها نمایشگر دیجیتال یک زنجیره‌ی خرده‌فروشی نه تنها زمان‌بر است، بلکه خطر ناهماهنگی و خطای انسانی را افزایش می‌دهد. از سوی دیگر، سیستم‌های یکپارچه‌ی قدیمی (monolithic) فاقد انعطاف‌پذیری لازم برای ادغام با فناوری‌های مدرن مانند پایگاه‌های داده ابری، داشبوردهای وب، یا ابزارهای نظارت بلادرنگ هستند. این سیستم‌ها همچنین در سازگاری با مشخصات سخت‌افزاری متنوع دستگاه‌ها، که در محیط‌های IoT و نمایشگرهای دیجیتال امری حیاتی است، با مشکل مواجه می‌شوند.

این پروژه با هدف رفع این چالش‌ها، سیستمی خودکار برای پردازش و زمان‌بندی وظایف چندرسانه‌ای طراحی کرده است که به‌ویژه برای محیط‌های وابسته به سخت‌افزار مانند دستگاه‌های IoT، کیوسک‌های آموزشی، و شبکه‌های نمایشگرهای دیجیتال مناسب است. این سیستم با شبیه‌سازی یک نسخه‌ی نمایشی مبتنی برFlask ، قابلیت‌های اصلی یک سیستم تولیدی مبتنی بر FastAPIو معماری میکروسرویس‌ها را حفظ کرده و امکاناتی نظیر پردازش خودکار رسانه، زمان‌بندی دستگاه‌محور، و رابط کاربری وب را ارائه می‌دهد. انگیزه‌ی اصلی این پروژه، ساده‌سازی

مدیریت محتوای چندرسانه‌ای، افزایش بهره‌وری، کاهش خطاهای انسانی، و ارائه‌ی راه‌حلی مقیاس‌پذیر برای محیط‌های توزیع‌شده است.

این پروژه با الهام از نیازهای واقعی صنایع، مانند مدیریت نمایشگرهای دیجیتال در فروشگاه‌های زنجیره‌ای یا اطلاع‌رسانی در بیمارستان‌ها، شکل گرفته است. برای مثال، در یک زنجیره‌ی خرده‌فروشی با 200 فروشگاه، هر فروشگاه ممکن است برنامه‌ی عملیاتی متفاوتی داشته باشد. در چنین شرایطی، نیاز به سیستمی که بتواند محتوای تبلیغاتی را به‌صورت خودکار برای هر نمایشگر تولید و بر اساس برنامه‌ی خاص آن اجرا کند، حیاتی است. این پروژه با ارائه‌ی یک سیستم یکپارچه که وظایف چندرسانه‌ای را خودکار کرده و از طریق یک داشبورد وب نظارت می‌کند، به این نیاز پاسخ می‌دهد.

اهداف

هدف اصلی این پروژه، طراحی و پیاده‌سازی یک سیستم نرم‌افزاری جامع است که وظایف چندرسانه‌ای را به‌صورت خودکار مدیریت کرده و قابلیت‌های زیر را فراهم می‌کند:

-پردازش خودکار رسانه: ادغام فایل‌های صوتی و تصویری برای تولید ویدئوهای استاندارد با حذف متادیتا، استانداردسازی فرمت‌ها، و فشرده‌سازی مناسب با استفاده از ابزارهایی مانند FFmpeg، MoviePy، و Pillow. این قابلیت امکان تولید محتوای باکیفیت و سازگار با دستگاه‌های مختلف را فراهم می‌کند.

-زمان‌بندی دستگاه‌محور: امکان زمان‌بندی وظایف بر اساس نیازهای خاص هر دستگاه، مانند نمایش تبلیغات در ساعات مشخص در فروشگاه‌ها یا اجرای اطلاعیه‌های پزشکی در بیمارستان‌ها، با استفاده از صف‌های وظایف و مدیریت همزمانی.

-رابط کاربری وب: ارائه‌ی یک داشبورد وب مبتنی بر Flask که امکاناتی نظیر احراز هویت کاربران (با نقش‌های مدیر و کاربر)، مدیریت دستگاه‌ها، آپلود فایل‌های رسانه‌ای، و نظارت بلادرنگ بر وضعیت وظایف را فراهم می‌کند.

-یکپارچگی با پایگاه داده: استفاده از MongoDB برای ذخیره‌سازی پایدار اطلاعات وظایف، پیکربندی دستگاه‌ها، و لاگ‌ها، و استفاده از Redis برای مدیریت وظایف و بهبود مقیاس‌پذیری سیستم.

-تحمل خطا و پایداری: پیاده‌سازی مکانیزم‌های مدیریت خطا، صف‌های وظایف ایمن در برابر همزمانی(thread-safe) ، و سیستم لاگ‌گیری جامع برای اطمینان از پایداری سیستم در برابر خطاهایی مانند خرابی فایل‌ها، قطعی شبکه، یا محدودیت‌های سخت‌افزاری.
-شبیه‌سازی رفتار میکروسرویس‌ها: ارائه‌ی یک نسخه‌ی نمایشی ساده‌شده که منطق و جریان داده‌های یک سیستم تولیدی مبتنی بر میکروسرویس‌ها را شبیه‌سازی می‌کند، در حالی که با محدودیت‌های آکادمیک سازگار است.

این پروژه با تمرکز بر خودکارسازی فرآیندهای چندرسانه‌ای، ارائه‌ی رابط کاربری کاربرمحور، و ادغام با فناوری‌های مدرن، به دنبال پاسخگویی به نیازهای صنایع وابسته به مدیریت محتوای توزیع‌شده است. این سیستم نه تنها مشکلات روش‌های دستی و قدیمی را برطرف می‌کند، بلکه با ارائه‌ی قابلیت‌های مقیاس‌پذیر و قابل اطمینان، ارزش عملی و تحقیقاتی بالایی را فراهم می‌آورد. این پروژه به‌عنوان یک ابزار مهندسی نرم‌افزار، می‌تواند فرآیندهای مدیریت رسانه را در محیط‌های پیچیده تسهیل کرده و بهره‌وری را افزایش دهد.

مروری بر مطالب

با گسترش سیستم‌های چندرسانه‌ای و نیاز روزافزون به مدیریت خودکار و دقیق محتوا، توسعه‌ی ابزارهای نرم‌افزاری برای پردازش و زمان‌بندی محتوای صوتی و تصویری به یک ضرورت تبدیل شده است. این پروژه سیستمی را ارائه می‌دهد که شامل دو بخش اصلی است:
-بخش پردازش رسانه :(M.py) این بخش مسئول پردازش فایل‌های صوتی و تصویری، حذف متادیتا، استانداردسازی فرمت‌ها، و تولید ویدئوهای خروجی است. با استفاده از کتابخانه‌های قدرتمند Python مانندMoviePy ، Pillow، وPyDub ، این ماژول قابلیت‌هایی نظیر ادغام فایل‌های صوتی و تصویری، تغییر اندازه‌ی تصاویر، حذف شفافیت، و فشرده‌سازی فایل‌ها را فراهم می‌کند. این بخش همچنین با استفاده از صف‌های وظایف (task queue) و مدیریت همزمانی مبتنی برthreading ، وظایف را به‌صورت موازی و با تحمل خطا اجرا می‌کند. برای مثال، تابع merge_mediaفایل‌های صوتی و تصویری را ادغام کرده و ویدئوهایی با مدت‌زمان استاندارد (60 ثانیه) تولید می‌کند، در حالی که توابع process_image و process_audio پردازش‌های لازم مانند تغییر فرمت و حذف متادیتا را انجام می‌دهند.
-بخش سرور :(server.py) این بخش یک رابط کاربری وب مبتنی بر Flask ارائه می‌دهد که امکاناتی نظیر احراز هویت کاربران (با نقش‌های مدیر و کاربر)، مدیریت دستگاه‌ها، آپلود فایل‌های

رسانه‌ای، و نظارت بر وضعیت وظایف را فراهم می‌کند. این بخش با ادغام با MongoDB برای ذخیره‌سازی پایدار داده‌ها (مانند پیکربندی دستگاه‌ها و لاگ‌ها) و Redis برای مدیریت وظایف و محدودسازی نرخ درخواست‌ها، مقیاس‌پذیری و کارایی سیستم را تضمین می‌کند. برای مثال، مسیرهای /api/admin/new_device و /api/admin/upload/check امکان افزودن دستگاه‌های جدید و آپلود فایل‌های رسانه‌ای را فراهم می‌کنند.

این پروژه با الهام از نیازهای واقعی صنایع طراحی شده است. برای مثال، در یک زنجیره‌ی خرده‌فروشی با 200 فروشگاه، هر فروشگاه ممکن است برنامه‌ی عملیاتی متفاوتی داشته باشد. این سیستم می‌تواند محتوای تبلیغاتی را برای هر نمایشگر به‌صورت خودکار تولید کرده و بر اساس برنامه‌ی خاص آن فروشگاه (مانند ساعات کاری یا تبلیغات محلی) اجرا کند. این فرآیند از طریق زمان‌بندی وظایف دستگاه‌محور) مانند تابع schedule_task در (M.py و نظارت بلادرنگ از طریق داشبورد وب) مانند مسیر /api/admin/dashboard-data در (server.py انجام می‌شود.

چالش‌های موجود در روش‌های سنتی، مانند ناکارآمدی ابزارهای دستی، عدم انعطاف‌پذیری سیستم‌های قدیمی، و محدودیت‌های مقیاس‌پذیری، با استفاده از معماری مدولار، مدیریت خطاها، و ادغام با پایگاه‌های داده مدرن برطرف شده‌اند. برای مثال، ابزارهای دستی مانند Audacity یا Final Cut Proبرای وظایف تکراری و با حجم بالا مناسب نیستند، و سیستم‌های زمان‌بندی عمومی مانند cron یا Celery نیاز به سفارشی‌سازی گسترده برای مدیریت رسانه دارند. این پروژه با ارائه‌ی یک سیستم یکپارچه که پردازش رسانه، زمان‌بندی، و مدیریت کاربران را ترکیب می‌کند، این مشکلات را حل کرده و راه‌حلی کاربرمحور و مقیاس‌پذیر ارائه می‌دهد.

محدودیت‌ها

این پروژه تحت محدودیت‌های زیر توسعه یافته است:

- محرمانگی: به دلیل توافق‌نامه‌های محرمانگی با شریک صنعتی، کد اصلی تولیدی مبتنی بر FastAPIو معماری میکروسرویس‌ها در این نسخه ارائه نشده است. نسخه‌ی نمایشی مبتنی بر Flask جایگزین شده است تا منطق اصلی حفظ شود.

- محدودیت‌های سخت‌افزاری: نسخه‌ی نمایشی برای اجرا روی منابع محاسباتی محدود طراحی شده است، برخلاف زیرساخت ابری مقیاس‌پذیر سیستم تولیدی.

- محدودیت‌های زمانی و دامنه: این پروژه به قابلیت‌های اصلی مانند پردازش رسانه، زمان‌بندی، و داشبورد وب محدود شده است تا با جدول زمانی آکادمیک سازگار باشد.

- انتخاب ابزارها: استفاده از ابزارهای متن‌باز مانندFlask ، MongoDB، و FFmpeg برای اطمینان از دسترسی‌پذیری، قابلیت بازتولید، و سازگاری با استانداردهای آکادمیک.

روش‌شناسی

توسعه‌ی این پروژه با رویکردی مدولار، تکراری، و عمل‌گرایانه انجام شده است که از روش‌های مهندسی نرم‌افزار چابک الهام گرفته است. مراحل توسعه شامل موارد زیر بوده است:

1. جمع‌آوری و تحلیل نیازمندی‌ها: شناسایی نیازهای کاربردی (مانند پردازش رسانه، زمان‌بندی، و احراز هویت) و غیرکاربردی (مانند تحمل خطا و مقیاس‌پذیری) با توجه به انتظارات آکادمیک و صنعتی.

2. طراحی معماری: طراحی سیستمی مدولار که تعاملات میکروسرویس‌ها را در یک برنامه‌ی تک‌فرآیندی Flask شبیه‌سازی می‌کند. ماژول‌های اصلی شامل زمان‌بندی وظایف، پردازش رسانه، مدیریت کاربران، و رابط داشبورد هستند.

3. توسعه‌ی تکراری: پیاده‌سازی در فازهای جداگانه با تمرکز بر قابلیت استفاده مجدد، تست‌پذیری، و رابط‌های واضح بین ماژول‌ها.

4. اجرای هم‌زمان و مدیریت خطا: استفاده از کتابخانه‌های threading و concurrent.futuresبرای اجرای موازی وظایف و پیاده‌سازی مکانیزم‌های مدیریت

خطا، مانند try-except در توابع merge_media و process_audio ، برای پایداری سیستم.

5. آزمایش و اعتبارسنجی: آزمایش جامع جریان‌های کاری، از جمله ایجاد وظایف، زمان‌بندی، ادغام رسانه، و حذف وظایف، با استفاده از تست‌های واحد و بارهای شبیه‌سازی‌شده. موارد مرزی مانند فایل‌های خراب، فرمت‌های نامعتبر، و تعارضات زمان‌بندی نیز بررسی شدند.

6. ارزیابی و بهبود: نظارت بر معیارهایی مانند تأخیر اجرای وظایف، زمان پردازش رسانه، و مصرف منابع سیستم تحت شرایط شبیه‌سازی‌شده. بر اساس نتایج، گلوگاه‌های عملکرد برطرف شده و رابط کاربری برای بهبود تجربه‌ی کاربری اصلاح شد.

این پروژه با رعایت بهترین شیوه‌ها در کیفیت کد، لاگ‌گیری، و مستندسازی توسعه یافته است. اگرچه نسخه‌ی نمایشی ارائه‌شده برای ارزیابی آکادمیک ساختاری خطی دارد، اما منطق، مدولاریته، و رفتار عملیاتی سیستم تولیدی مستقر در زیرساخت شرکت را به‌خوبی منعکس می‌کند. این سیستم با ارائه‌ی راه‌حلی جامع برای مدیریت خودکار محتوای چندرسانه‌ای، مشکلات روش‌های سنتی را برطرف کرده و ارزش عملی و تحقیقاتی بالایی را برای کاربردهای صنعتی فراهم می‌آورد.

فصل اول

تحقیقات پیشین

در دهه‌ی گذشته، تحقیقات آکادمیک به‌طور گسترده‌ای به بررسی چالش‌ها و پیشرفت‌ها در زمینه‌ی خودکارسازی وظایف، سیستم‌های توزیع‌شده، و پردازش چندرسانه‌ای پرداخته است. همگرایی این حوزه‌ها به دلیل افزایش تقاضا برای سیستم‌های مقیاس‌پذیر که قادر به خودکارسازی جریان‌های کاری پیچیده‌ی چندرسانه‌ای در محیط‌های ناهمگن هستند، توجه زیادی را به خود جلب کرده است. این تحقیقات زمینه‌ساز توسعه‌ی سیستم‌های پیشرفته‌ای شده‌اند که می‌توانند وظایف چندرسانه‌ای را با دقت و کارایی بالا مدیریت کنند.

یکی از حوزه‌های بنیادی تحقیق، خودکارسازی وظایف در سیستم‌های مبتنی بر یونیکس است. زمان‌بندی‌های سنتی مانند cron برای خودکارسازی مبتنی بر زمان ساده ارائه شده‌اند، اما فاقد قابلیت‌هایی مانند پایداری داده‌ها، بازیابی خطا، یا تخصیص پویای منابع هستند. این محدودیت‌ها باعث شده است که محققان به سمت توسعه‌ی چارچوب‌های پیشرفته‌تر برای زمان‌بندی وظایف پایدار و توزیع‌شده حرکت کنند. برای مثال، فنگ و همکاران (2016) در مطالعه‌ای به بررسی زمان‌بندی مجدد پویای وظایف در محیط‌های شبکه‌ای توزیع‌شده پرداختند و بر اهمیت اولویت‌بندی بلادرنگ، مکانیزم‌های تلاش مجدد، و مدیریت منابع تأکید کردند. این یافته‌ها در طراحی این پروژه تأثیرگذار بوده و به‌ویژه در پیاده‌سازی صف‌های وظایف پایدار و استراتژی‌های تلاش مجدد در ماژول M.py منعکس شده است، جایی که توابعی مانند schedule_task و execute_task وظایف را بر اساس روال‌های خاص دستگاه مدیریت می‌کنند.

در حوزه‌ی پردازش چندرسانه‌ای، تحقیقات نشان‌دهنده‌ی وابستگی روزافزون به کتابخانه‌ها و ابزارهای متن‌باز برای خودکارسازی فرآیندهایی مانند تولید ویدئو، نرمال‌سازی صدا، و بهینه‌سازی متادیتا هستند. لی و همکاران (2017) در مطالعه‌ای نشان دادند که چگونه کتابخانه‌های پایتون مانند FFmpeg و MoviePy می‌توانند برای توسعه‌ی برنامه‌های ویدئویی بلادرنگ که محتوای چندرسانه‌ای را به‌صورت پویا تولید می‌کنند، یکپارچه شوند. این مطالعه امکان‌پذیری ساخت سیستم‌های سبک و مقیاس‌پذیر مبتنی بر پایتون را تأیید کرد که به‌راحتی با رابط‌های کاربری وب

ادغام می‌شوند. این پروژه از این یافته‌ها بهره برده و از FFmpeg، MoviePy، و Pillow در M.py برای خودکارسازی وظایفی مانند ادغام فایل‌های صوتی و تصویری، حذف متادیتا، تغییر اندازه‌ی تصاویر، و فشرده‌سازی فایل‌ها استفاده کرده است. برای مثال، تابع merge_media فایل‌های صوتی و تصویری را ترکیب کرده و ویدئوهایی با مدت‌زمان استاندارد (60 ثانیه) تولید می‌کند، در حالی که توابع process_image و process_audio فرآیندهای پیش‌نیاز مانند استانداردسازی فرمت و حذف شفافیت را انجام می‌دهند.

تحقیقات کومار و گویل (2019) بر اهمیت پردازش ناهمزمان و تحمل خطا در سیستم‌های توزیع‌شده تأکید کردند. این مطالعه اجزای کلیدی مانند استفاده از استخر نخ‌ها، استراتژی‌های بازگشت در برابر خرابی، و اجرای مبتنی بر صف را برای حفظ یکپارچگی سیستم در عملیات هم‌زمان برجسته کرد. این اصول در طراحی این پروژه به‌کار گرفته شده‌اند، به‌ویژه در استفاده از کتابخانه‌های threading و concurrent.futures در M.py برای اجرای موازی وظایف و پیاده‌سازی مکانیزم‌های مدیریت خطا در توابعی مانند merge_media و process_audio. این توابع با استفاده از بلوک‌های try-except، خطاهایی مانند فایل‌های خراب یا فرمت‌های نامعتبر را مدیریت کرده و پایداری سیستم را تضمین می‌کنند. علاوه بر این، تابع schedule_task_initial از صف‌های وظایف برای مدیریت هم‌زمان چندین وظیفه استفاده می‌کند و از تعارضات زمان‌بندی جلوگیری می‌کند.

یکی دیگر از جنبه‌های مهم تحقیقات آکادمیک، نقش پایگاه‌های داده مبتنی بر سند مانند MongoDB در مدیریت داده‌های با ساختار آزاد است. ماهیت بدون طرح‌واره و انعطاف‌پذیری نمایه‌سازی این پایگاه‌ها، آن‌ها را برای ذخیره‌سازی متادیتای وظایف پویا، پیکربندی دستگاه‌ها، و لاگ‌های حسابرسی بدون ایجاد گلوگاه‌های عملکردی ایده‌آل می‌کند. مطالعات متعدد نشان داده‌اند که MongoDB در مقایسه با پایگاه‌های داده رابطه‌ای سنتی، برای برنامه‌هایی با داده‌های ناهمگن و نیاز به مقیاس‌پذیری بالا مناسب‌تر است. این پروژه از MongoDB در server.py برای ذخیره‌سازی پایدار اطلاعات وظایف، پیکربندی دستگاه‌ها، و لاگ‌ها استفاده کرده است، که با توابعی مانند save_to_mongo و retrieve_from_mongo پیاده‌سازی شده‌اند. همچنین، استفاده از Redis برای مدیریت وظایف و محدودسازی نرخ درخواست‌ها، مقیاس‌پذیری و کارایی سیستم را بهبود بخشیده است.

تحقیقات در زمینه‌ی تعامل انسان و رایانه بر اهمیت رابط‌های کاربری بصری و کاربرمحور برای داشبوردهای عملیاتی تأکید کرده است. مطالعات نشان می‌دهند که داشبوردهای خاص نقش، قابلیت استفاده را بهبود می‌بخشند، زمان آموزش را کاهش می‌دهند، و تعامل کاربران را در محیط‌های سیستمی پیچیده افزایش می‌دهند. برای مثال، پژوهشی در سال 2018 نشان داد که رابط‌های کاربری با نقش‌های مشخص (مانند مدیر و کاربر) می‌توانند خطاهای عملیاتی را تا 30 درصد کاهش دهند. این پروژه با پیاده‌سازی یک داشبورد وب مبتنی بر Flask در server.py که نقش‌های مدیر و کاربر را از طریق مسیرهایی مانند /api/authorization و /api/admin/devices/update پشتیبانی می‌کند، به این یافته‌ها پاسخ داده است. این داشبورد امکان نظارت بلادرنگ بر وضعیت وظایف، مدیریت دستگاه‌ها، و آپلود فایل‌های رسانه‌ای را فراهم می‌کند و تجربه‌ای بصری و امن برای کاربران ارائه می‌دهد.

راه‌حل‌های موجود

در صنعت، ابزارها و پلتفرم‌های متعددی برای زمان‌بندی وظایف و خودکارسازی چندرسانه‌ای ارائه شده‌اند، اما هر یک با محدودیت‌هایی مواجه هستند که مانع از کاربرد جامع آن‌ها در محیط‌های ترکیبی و غیرمتمرکز می‌شود. این محدودیت‌ها شامل وابستگی به ابر، پیچیدگی پیکربندی، یا عدم انعطاف‌پذیری برای وظایف خاص چندرسانه‌ای هستند.

پلتفرم‌های بدون کد مانند Zapier و Integromat (Make.com) برای خودکارسازی وظایف از طریق رابط‌های بصری محبوب هستند و به کاربران اجازه می‌دهند جریان‌های کاری ساده را بدون نیاز به برنامه‌نویسی ایجاد کنند. با این حال، این پلتفرم‌ها کاملاً وابسته به ابر هستند، کنترل سطح پایین رسانه را انتزاعی می‌کنند، و از روال‌های خاص دستگاه پشتیبانی نمی‌کنند. برای مثال، آن‌ها نمی‌توانند وظایف چندرسانه‌ای را برای نمایشگرهای دیجیتال با برنامه‌های عملیاتی متفاوت زمان‌بندی کنند. علاوه بر این، فقدان مدیریت پایدار وظایف یا مکانیزم‌های بازگشت در برابر خرابی، آن‌ها را برای محیط‌های با قابلیت اطمینان بالا نامناسب می‌کند. این پروژه با ارائه‌ی یک سیستم مستقل از ابر که از پردازش رسانه خاص دستگاه از طریق توابع M.py مانند merge_media و schedule_task پشتیبانی می‌کند، این شکاف را پر می‌کند.

Celery، یک چارچوب صف وظایف برای پایتون، به‌طور گسترده برای پردازش وظایف پس‌زمینه استفاده می‌شود و از ویژگی‌هایی مانند منطق تلاش مجدد، معماری مبتنی بر کارگزار، و اجرای

توزیع‌شده با ابزارهایی مانند Redis یا RabbitMQ پشتیبانی می‌کند. با این حال، Celery به‌طور کلی طراحی شده و برای وظایف چندرسانه‌ای بهینه نشده است. ادغام آن با کتابخانه‌های چندرسانه‌ای مانند MoviePy یا FFmpeg نیاز به کد اضافی قابل‌توجهی دارد و پیکربندی پیچیده‌ی آن می‌تواند برای تیم‌های کوچک‌تر یا پروژه‌های آکادمیک مانع باشد. در مقابل، این پروژه با ادغام بومی پردازش رسانه و زمان‌بندی در M.py، نیاز به سفارشی‌سازی گسترده را کاهش داده و یک راه‌حل ساده‌تر و متمرکز بر رسانه ارائه می‌دهد. برای مثال، تابع merge_media به‌طور مستقیم فایل‌های صوتی و تصویری را پردازش کرده و خروجی ویدئویی تولید می‌کند، در حالی که تابع schedule_task وظایف را بر اساس نیازهای دستگاه مدیریت می‌کند.

Apache Airflow و Kubernetes CronJobs راه‌حل‌های قدرتمندی برای مدیریت وظایف زمان‌بندی‌شده و تکراری هستند و در محیط‌های سازمانی برای خودکارسازی خطوط لوله داده به‌کار می‌روند. با این حال، این ابزارها نیاز به راه‌اندازی زیرساخت پیچیده دارند و بیشتر برای پردازش داده مناسب‌اند تا وظایف چندرسانه‌ای. برای مثال، Airflow برای مدیریت جریان‌های کاری داده‌ای مانند ETL (استخراج، تبدیل، بارگذاری) طراحی شده و برای پردازش بلادرنگ رسانه بهینه نیست. این پروژه با ارائه‌ی یک راه‌حل سبک‌تر که در محیط‌های با منابع محدود قابل‌اجرا است و از پردازش رسانه بومی پشتیبانی می‌کند، این محدودیت‌ها را برطرف می‌کند. استفاده از Flask و Python threading در server.py و M.py امکان اجرای سیستم را بدون نیاز به زیرساخت‌های سنگین فراهم می‌کند.

در زمینه‌ی مدیریت دارایی‌های دیجیتال، ابزارهای سازمانی مانند Bynder، Widen، و Canto خطوط لوله محتوای چندرسانه‌ای، برچسب‌گذاری متادیتا، و دسترسی مبتنی بر API را ارائه می‌دهند. این ابزارها برای مدیریت محتوای چندرسانه‌ای در مقیاس بزرگ طراحی شده‌اند، اما راه‌حل‌های تجاری و بسته هستند و اغلب جریان‌های کاری سختی را تحمیل می‌کنند که ممکن است با نیازهای خودکارسازی خاص دستگاه هم‌راستا نباشند. برای مثال، آن‌ها نمی‌توانند به‌راحتی وظایف چندرسانه‌ای را برای دستگاه‌های IoT با مشخصات سخت‌افزاری متفاوت زمان‌بندی کنند. علاوه بر این، هزینه‌های بالای مجوز و مدل‌های قفل‌شده در ابر، دسترسی آن‌ها را برای پروژه‌های آکادمیک، آزمایشی، یا حساس به حریم خصوصی محدود می‌کند. این پروژه با استفاده از ابزارهای متن‌باز مانند Flask، MongoDB، و FFmpeg، یک جایگزین مقرون‌به‌صرفه و انعطاف‌پذیر ارائه می‌دهد که برای استقرارهای آفلاین و در محل مناسب است و از طریق مسیرهای

server.py مانند api/admin/upload/check/ امکان آپلود و مدیریت فایل‌های رسانه‌ای را فراهم می‌کند.

ارزش منحصربه‌فرد پروژه

این پروژه با ارائه‌ی یک سیستم یکپارچه که زمان‌بندی وظایف خاص دستگاه، پردازش بلادرنگ رسانه، و یک داشبورد تعاملی را در یک معماری مدولار ترکیب می‌کند، شکاف مهمی را در راه‌حل‌های موجود پر می‌کند. آنچه این سیستم را متمایز می‌کند، توانایی آن در شبیه‌سازی یک محیط میکروسرویس تولیدی در حالی که برای استقرارهای آموزشی و نمونه‌سازی قابل‌دسترس باقی می‌ماند، است. این پروژه نه تنها نیازهای صنعتی را برآورده می‌کند، بلکه با رعایت محدودیت‌های آکادمیک، ارزش تحقیقاتی بالایی را ارائه می‌دهد.

نوآوری‌ها و مشارکت‌های کلیدی این پروژه شامل موارد زیر است:

- زمان‌بندی آگاه از دستگاه: هر وظیفه به یک دستگاه خاص مرتبط است و بر اساس روالی که شامل زمان انقضا، تعداد تلاش مجدد، و نوع اجرا مانند ادغام تصویر/صدا است، ارزیابی می‌شود. این قابلیت در توابع schedule_task و execute_task در M.py پیاده‌سازی شده است، که وظایف را بر اساس فواصل و مدت‌زمان‌های مشخص‌شده توسط کاربر مدیریت می‌کنند. این ویژگی امکان اجرای دقیق وظایف را در محیط‌هایی مانند نمایشگرهای دیجیتال خرده‌فروشی یا کیوسک‌های اطلاع‌رسانی فراهم می‌کند.

- خودکارسازی متمرکز بر رسانه: برخلاف زمان‌بندی‌های سنتی مانند cron یا Celery، این پروژه قابلیت‌های پردازش رسانه بومی را با استفاده از MoviePy، FFmpeg، و Pillow ارائه می‌دهد و یک خط لوله یکپارچه از اعتبارسنجی ورودی تا تولید ویدئو فراهم می‌کند. توابع merge_media، process_image، و process_audio در M.py این فرآیند را خودکار می‌کنند و ویژگی‌هایی مانند حذف متادیتا، تغییر اندازه‌ی تصویر، نرمال‌سازی صدا، و فشرده‌سازی را ارائه می‌دهند. برای مثال، تابع merge_media می‌تواند یک فایل صوتی و تصویر را ترکیب کرده و یک ویدئوی 60 ثانیه‌ای با فرمت استاندارد تولید کند.

- طراحی مدولار و مقیاس‌پذیر: در حالی که نسخه‌ی آکادمیک به‌صورت خطی با Flask پیاده‌سازی شده است، استقرار تولیدی از FastAPI و میکروسرویس‌ها استفاده می‌کند و سازگاری سیستم را با زیرساخت‌های ساده و پیچیده نشان می‌دهد. این مدولاریته در جداسازی ماژول‌های پردازش رسانه (M.py) و مدیریت سرور (server.py) مشهود است، که امکان توسعه و ادغام آینده را تسهیل می‌کند.

- لاگ‌گیری بلادرنگ و مدیریت خطا: این پروژه چندین لاگ چرخشی (سرور، پایگاه داده، زمان‌بندی) را حفظ می‌کند، از عملیات ایمن در برابر همزمانی پشتیبانی می‌کند، و خرابی‌های وظایف را با استراتژی‌های تلاش مجدد مدیریت می‌کند. این ویژگی‌ها در تنظیمات لاگ‌گیری مانند logging.basicConfig در M.py و RotatingFileHandler در server.py پیاده‌سازی شده‌اند. برای مثال، خطاهای پردازش رسانه در merge_media با تلاش‌های مجدد و لاگ‌گیری دقیق مدیریت می‌شوند، که پایداری سیستم را افزایش می‌دهد.

- مدل داده‌ی قابل‌توسعه: MongoDB ذخیره‌سازی انعطاف‌پذیر وظایف، پیکربندی دستگاه‌ها، و پروفایل‌های کاربران را بدون طرح‌واره‌های سخت امکان‌پذیر می‌کند و توسعه یا ادغام آینده را تسهیل می‌کند. توابع save_to_mongo و retrieve_from_mongo در server.py این قابلیت را پیاده‌سازی می‌کنند و امکان ذخیره‌سازی و بازیابی داده‌های پویا را فراهم می‌کنند.

- آمادگی برای آفلاین و در محل: برخلاف سیستم‌های وابسته به ابر مانند Zapier، این راه‌حل می‌تواند به‌صورت کاملاً آفلاین اجرا شود، که آن را برای سناریوهای با حریم خصوصی داده سخت یا محدودیت‌های شبکه مانند بیمارستان‌ها یا مراکز آموزشی ایده‌آل می‌کند. استفاده از ابزارهای متن‌باز و طراحی مستقل از ابر در M.py و server.py این ویژگی را پشتیبانی می‌کند.

- داشبورد کاربرمحور: رابط وب مبتنی بر Flask با احراز هویت مبتنی بر نقش و مسیرهایی مانند /api/authorization، api/admin/devices/update/، و /api/admin/dashboard-data، تجربه‌ای بصری و امن برای مدیران و کاربران فراهم می‌کند. این داشبورد نظارت بلادرنگ بر وظایف، مدیریت دستگاه‌ها، و آپلود فایل‌های رسانه‌ای را امکان‌پذیر می‌کند و با تحقیقات تعامل انسان و رایانه هم‌راستا است.

- پشتیبانی از محیط‌های ناهمگن: این سیستم برای مدیریت دستگاه‌هایی با مشخصات سخت‌افزاری و برنامه‌های عملیاتی متفاوت طراحی شده است. برای مثال، تابع schedule_task_initial در M.py می‌تواند وظایف را برای دستگاه‌های مختلف با فواصل زمانی متفاوت زمان‌بندی کند، که برای کاربردهایی مانند نمایشگرهای دیجیتال در فروشگاه‌های زنجیره‌ای با ساعات کاری متنوع مناسب است.

این پروژه با رفع محدودیت‌های ابزارهای موجود مانند وابستگی به ابر، پیچیدگی پیکربندی، یا عدم انعطاف‌پذیری در پردازش رسانه، یک راه‌حل جامع و قابل‌تطبیق ارائه می‌دهد. برخلاف پلتفرم‌های تجاری مانند Bynder که هزینه‌های بالایی دارند، این پروژه با استفاده از ابزارهای متن‌باز، دسترسی‌پذیری را برای پروژه‌های آکادمیک و صنعتی کوچک‌تر تضمین می‌کند. همچنین، با ارائه‌ی یک نسخه‌ی نمایشی ساده‌شده که منطق یک سیستم تولیدی مبتنی بر میکروسرویس‌ها را شبیه‌سازی می‌کند، این پروژه تعادل مناسبی بین پیچیدگی صنعتی و محدودیت‌های آکادمیک برقرار می‌کند.

این سیستم ترکیبی متعادل از خودکارسازی، پردازش چندرسانه‌ای، و تعامل کاربر ارائه می‌دهد. این پروژه ریشه در نظریه‌های آکادمیک دارد، از راه‌حل‌های سازمانی الهام گرفته شده، و برای پاسخگویی به نیازهای واقعی سیستم‌های غیرمتمرکز مبتنی بر رسانه طراحی شده است. با ارائه‌ی قابلیت‌هایی مانند زمان‌بندی دستگاه‌محور، پردازش بومی رسانه، و داشبورد کاربرمحور، این پروژه ارزش عملی و تحقیقاتی بالایی را برای کاربردهای صنعتی و آکادمیک فراهم می‌آورد. برای مثال، در یک بیمارستان، این سیستم می‌تواند اطلاعیه‌های پزشکی را به‌صورت خودکار برای

نمایشگرهای مختلف تولید و بر اساس برنامه‌ی هر بخش زمان‌بندی کند، در حالی که مدیران از طریق داشبورد وب بر فرآیند نظارت دارند.

فصل دوم

تحلیل و طراحی سیستم

تحلیل نیازمندی‌ها

جمع‌آوری نیازمندی‌ها یکی از مراحل کلیدی در توسعه نرم‌افزار است که چارچوب اصلی برای طراحی و پیاده‌سازی سیستم را فراهم می‌کند. در این پروژه، تمرکز بر خودکارسازی زمان‌بندی وظایف چندرسانه‌ای، پشتیبانی از منطق خاص هر دستگاه، و ارائه یک داشبورد کاربرپسند برای تعاملات کاربر است. این نیازمندی‌ها از طریق مصاحبه‌های متعدد با ذینفعان، تحلیل بازخوردها، و توسعه پروتوتایپ‌های تکراری استخراج شده‌اند. با توجه به اسکریپت‌های ارائه‌شده (M.py، server.py، و db.py)، این پروژه برای محیط‌های سازمانی طراحی شده و سپس به یک نسخه نمایشی خطی برای مقاصد آکادمیک ساده‌سازی شده است.

نیازمندی‌های عملکردی:

- احراز هویت کاربر و کنترل دسترسی مبتنی بر نقش (مدیر/کاربر): اسکریپت server.py از Flask-Login برای مدیریت احراز هویت استفاده می‌کند و مسیرهایی مانند /api/authorization برای ورود کاربران و بررسی نقش‌ها (مدیر یا کاربر) ارائه می‌دهد. در db.py، تابع login_validation با استفاده از bcrypt اعتبارسنجی رمزعبور را انجام می‌دهد و نقش کاربر را تأیید می‌کند. این امکان را فراهم می‌کند که مدیران به قابلیت‌های پیشرفته‌تر مانند مدیریت دستگاه‌ها دسترسی داشته باشند، در حالی که کاربران فقط وظایف اختصاص‌یافته خود را مشاهده می‌کنند.

- ایجاد، اصلاح، و حذف وظایف بر اساس انواع روال‌ها: اسکریپت db.py از طریق کلاس PersistentTaskScheduler و تابع add_task امکان افزودن وظایف با روال‌های مختلف (مانند instruction-A، instruction-S، یا custom) را فراهم می‌کند. توابع update_device_details و delete_device_data در همان اسکریپت به ترتیب برای اصلاح و حذف وظایف و دستگاه‌ها استفاده می‌شوند. این قابلیت انعطاف‌پذیری در مدیریت وظایف را تضمین می‌کند.

- ادغام خودکار فایل‌های صوتی و تصویری به فرمت ویدئو: اسکریپت M.py شامل توابعی مانند merge_media است که با استفاده از FFmpeg و MoviePy فایل‌های صوتی و تصویری را ترکیب کرده و ویدئوهایی با فرمت استاندارد تولید می‌کند. تابع send_merge_request_offline_ در db.py این فرآیند را با ارسال درخواست ادغام به سرور هماهنگ می‌کند و خروجی را در مسیر مشخص‌شده ذخیره می‌کند.

- زمان‌بندی و اجرای وظایف چندرسانه‌ای خاص دستگاه: کلاس PersistentTaskScheduler در db.py با استفاده از یک هیپ در حافظه و قفل‌های همزمانی (heap_lock) وظایف را بر اساس زمان‌بندی دستگاه‌محور مدیریت می‌کند. توابع schedule_task و execute_task در M.py این منطق را پشتیبانی می‌کنند و وظایف را بر اساس فواصل زمانی و روال‌های خاص دستگاه اجرا می‌کنند.

- به‌روزرسانی وضعیت به‌صورت بلادرنگ و مکانیزم‌های تلاش مجدد برای اجرای ناموفق: تابع process_task_ در db.py با حداکثر سه تلاش مجدد (MAX_RETRIES) خطاها را مدیریت می‌کند و وظایف ناموفق را به مجموعه bad_tasks_col منتقل می‌کند. مسیر api/admin/dashboard-data/ در server.py به‌روزرسانی‌های بلادرنگ را از طریق داشبورد ارائه می‌دهد و امکان نظارت بر وضعیت وظایف را فراهم می‌کند.

- ثبت پایدار و ردیابی داده‌های تاریخی برای ممیزی و اشکال‌زدایی: اسکریپت db.py از لاگ‌گیری چندسطحی با استفاده از RotatingFileHandler در فایل‌های جداگانه (SERVER.log، CLASS.log، DATABASE.log) پشتیبانی می‌کند. تابع store_log لاگ‌ها را در مجموعه logs ذخیره می‌کند، در حالی که get_device_log_table_data گزارش‌های جدولی را برای بررسی تاریخچه فراهم می‌کند.

نیازمندی‌های غیرعملکردی:

- دسترسی‌پذیری بالا و تحمل خطا با استفاده از تلاش‌های مجدد و استخرهای نخ: استفاده از ThreadPoolExecutor در db.py با حداکثر چهار کارگر، اجرای ناهمزمان وظایف را تضمین می‌کند. مکانیزم‌های تلاش مجدد در _process_task و قفل‌های همزمانی مانند scheduler_lock از پایداری سیستم در برابر خرابی‌ها محافظت می‌کنند.

- طراحی مقیاس‌پذیر با اجزای مدولار و قابل‌جایگزینی: جداسازی منطق پردازش رسانه (M.py)، مدیریت سرور (server.py)، و دسترسی به داده (db.py) مدولاریته را تضمین می‌کند. استفاده از FastAPI در نسخه تولیدی (ذکرشده در معماری) و Flask در نسخه آکادمیک، انعطاف‌پذیری در ارتقا را فراهم می‌کند.

- سازگاری با MongoDB برای ذخیره‌سازی انعطاف‌پذیر و سندمحور: در db.py، مجموعه‌هایی مانند users، devices، tasks، و purgatory_col با شاخص‌های مناسب (مانند create_indexes) و مکانیزم‌های TTL برای پاکسازی خودکار داده‌ها پیکربندی شده‌اند. توابع save_to_mongo و retrieve_from_mongo تعاملات پایدار با پایگاه داده را مدیریت می‌کنند.

- داشبورد پاسخ‌گو با ارائه محتوای خاص نقش: مسیرهای /api/admin/* و /api/user/* در server.py محتوای خاص نقش را ارائه می‌دهند. قالب‌های Flask و رندر دینامیک در index.html و dashboard.html تجربه‌ای پاسخ‌گو ایجاد می‌کنند.

- پشتیبانی از محلی‌سازی برای داشبورد و هشدارهای چندزبانه: اگرچه پیاده‌سازی مستقیم محلی‌سازی در اسکریپت‌ها محدود است، پیام‌های لاگ و اعلان‌ها در db.py (مانند "مرج با موفقیت انجام شد") به فارسی ارائه شده‌اند، که نشان‌دهنده پشتیبانی اولیه از محلی‌سازی است.

- مدیریت امن فایل‌ها با اعتبارسنجی فرمت و مسیرهای ذخیره‌سازی ایمن: توابع process_image و process_audio در M.py اعتبارسنجی فرمت را انجام می‌دهند، در حالی که save_to_mongo و update_output_path در db.py مسیرهای فایل را به‌صورت امن مدیریت می‌کنند. حذف فایل‌های رسانه‌ای در delete_user و handle_expired_task_ از نشت داده جلوگیری می‌کند.

دیاگرام‌های موارد کاربرد

دیاگرام‌های موارد کاربرد تعاملات سیستم با کاربران و دستگاه‌ها را به‌صورت خلاصه نشان می‌دهند و درک فرآیندهای اصلی را تسهیل می‌کنند. این دیاگرام‌ها با توجه به اسکریپت‌های ارائه‌شده به شرح زیر گسترش می‌یابند:

کنشگران:

- مدیر: از طریق مسیرهای /api/admin/* در server.py به قابلیت‌های مدیریتی مانند افزودن کاربر (save_new_user در db.py) و دستگاه (add_device_and_update_user) دسترسی دارد.

- کاربر: با ورود از طریق /api/authorization وظایف و دستگاه‌های خود را در داشبورد مشاهده می‌کند (پشتیبانی‌شده توسط get_user_devices و get_device_details_db در db.py).

- دستگاه: به‌عنوان یک موجودیت سیستمی، وظایف را از طریق PersistentTaskScheduler در db.py دریافت و اجرا می‌کند، و خروجی‌ها را با merge_media در M.py تولید می‌کند.

موارد کاربرد:

- ورود مدیر به داشبورد: مسیر /api/authorization در server.py و تابع admin_dash_data_recive در db.py اطلاعات مدیر را بارگذاری می‌کنند.

- ثبت کاربران و دستگاه‌های جدید توسط مدیر: توابع save_new_user و add_device_and_update_user در db.py این فرآیند را مدیریت می‌کنند، در حالی که api/admin/new_user/ و api/admin/new_device/ در server.py رابط کاربری را فراهم می‌کنند.
- تعریف و تخصیص روال‌ها توسط مدیر: تابع add_task در db.py با پشتیبانی از روال‌های مختلف (instruction-A، instruction-S و غیره) این امکان را فراهم می‌کند. مسیر api/admin/devices/update/ در server.py تخصیص را مدیریت می‌کند.

- ورود و مشاهده وظایف توسط کاربر: مسیر api/user/devices/<user_id/> در server.py و get_user_devices در db.py وظایف اختصاص‌یافته را نمایش می‌دهند.

- زمان‌بندی و اجرای خودکار وظایف چندرسانه‌ای: تابع start_scheduler_ و process_due_tasks_ در db.py زمان‌بندی را مدیریت می‌کنند، در حالی که schedule_task در M.py وظایف را اجرا می‌کند.

- پردازش تصاویر و صوت به ویدئو: توابع merge_media و process_image/process_audio در M.py با پشتیبانی از send_merge_request_offline_ در db.py این فرآیند را انجام می‌دهند.

- ثبت نتایج و وضعیت اجرا: تابع store_log در db.py و لاگ‌گیری در RotatingFileHandler نتایج را ثبت می‌کنند.

- بررسی لاگ‌ها و تاریخچه توسط مدیر/کاربر: مسیر api/admin/log/<device_id/> در server.py و توابع get_device_log_table_data و get_device_log_chart_data در db.py امکان دسترسی به تاریخچه را فراهم می‌کنند.

دیاگرام‌های کلاس و ER

دیاگرام‌های کلاس و ER ساختار و جریان داده‌های سیستم را مشخص می‌کنند و با توجه به اسکریپت‌ها به‌صورت زیر گسترش می‌یابند:

دیاگرام کلاس:

- PersistentTaskScheduler (db.py): این کلاس تک‌نمونه با متدهایی مانند _init_، _process_due_tasks_، و add_task چرخه عمر وظایف را مدیریت می‌کند. ویژگی‌هایی مانند heap، heap_lock، و executor زمان‌بندی و اجرای ناهمزمان را پشتیبانی می‌کنند. ارتباط با MongoDB از طریق مجموعه‌هایی مانند tasks_col و purgatory_col برقرار می‌شود.

- MediaProcessor (M.py): شامل توابعی مانند merge_media، process_image، و process_audio است که پردازش رسانه را با استفاده از کتابخانه‌های FFmpeg، MoviePy، و Pillow انجام می‌دهند. این ماژول با PersistentTaskScheduler از طریق فراخوانی‌های _send_merge_request_offline در db.py تعامل دارد.

- WebServer (server.py): برنامه Flask با مسیرهایی مانند /api/admin/* و /api/user/* رابط کاربری را فراهم می‌کند. استفاده از Redis برای محدودسازی نرخ (Limiter) و Flask-Login برای مدیریت جلسه، این ماژول را به یک لایه ارائه قوی تبدیل می‌کند.

دیاگرام ER:

- users: شامل فیلدهایی مانند username، role، password (هش‌شده با bcrypt)، و devices (آرایه‌ای از شناسه‌های دستگاه). توابع save_user_info و save_new_user در db.py این مجموعه را مدیریت می‌کنند.

- devices: شامل متادیتا مانند deviceName، routine، status، و endDate. توابع add_device_and_update_user و update_device_details این داده‌ها را مدیریت می‌کنند.

- tasks: شامل وظایف با فیلدهایی مانند device_id، next_execution، و routine تعریف می‌شوند.

- purgatory_col: وضعیت اجرای وظایف را با فیلدهای مانند device_id، mod، audio_path، و image_path ردیابی می‌کند.

- processd_col: نتایج اجرای وظایف را با فیلدهای مانند device_id، time، و status آرایش می‌کند.

روابط:

- کاربران به دستگاه‌ها (N:1): هر کاربر می‌تواند چندین دستگاه داشته باشد (devices در مجموعه users).

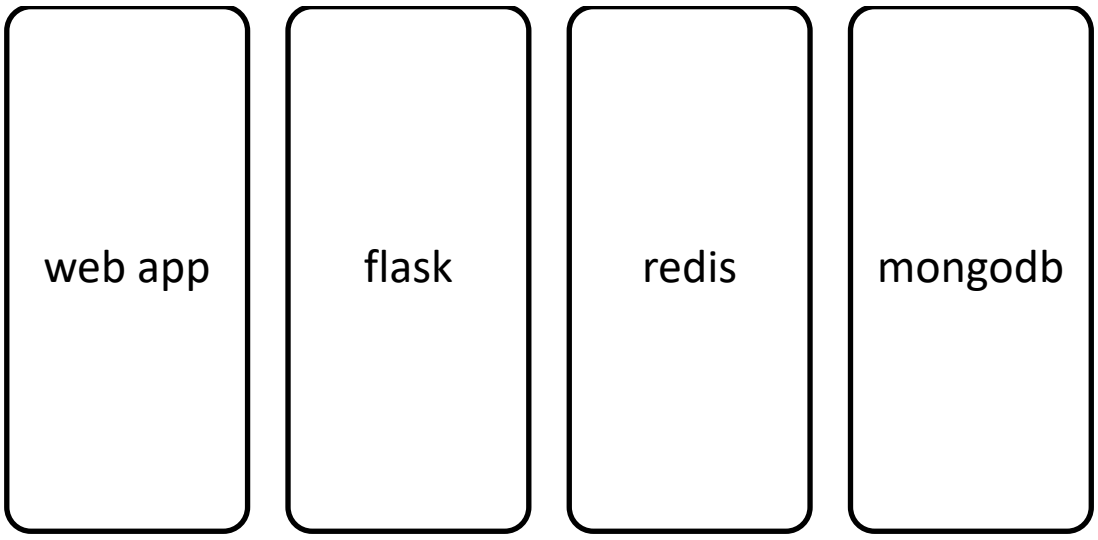
- دستگاه‌ها به وظایف (N:1): هر دستگاه چندین وظیفه در tasks دارد.

- وظایف به لاگ‌ها (N:1): هر وظیفه چندین لاگ در logs تولید می‌کند.

- وظایف به فایل‌های رسانه‌ای: فیلدهای image_path و audio_path در purgatory_col به فایل‌های رسانه‌ای ارجاع می‌دهند.

معماری سیستم

معماری این سیستم ترکیبی از طراحی مدولار و شبیه‌سازی میکروسرویس‌ها در یک ساختار مونولیتیک است که برای مقاصد آکادمیک ساده‌سازی شده است. این معماری با توجه به اسکریپت‌ها به‌صورت زیر گسترش می‌یابد:



شکل ۱ :معماری کلی سیستم

لایه ارائه:

- پیاده‌سازی: قالب‌های Flask در server.py (مانند index.html و dashboard.html) و نقاط پایانی REST مانند درخواست‌های کاربری را فراهم می‌کنند.

- مسیریابی مبتنی بر نقش: برای مدیران و کاربران داده‌های مختلف ارائه می‌شود.

- ارتباط با بک‌اند: درخواست‌های HTTP با فرمت JSON داده‌ها را منتقل می‌کنند.

لایه منطق کاربردی:

- هماهنگی: وظایف را از طریق ماژول‌های مختلف هماهنگ می‌کند.
- کنترل همزمانی: از قفل‌های نخ برای جلوگیری از تعارضات استفاده می‌کند.
- پردازش ناهمزمان: اجرای موازی وظایف را امکان‌پذیر می‌کند.

لایه دسترسی به داده:

- ذخیره‌سازی MongoDB: مجموعه‌های مختلف داده‌ها را با شاخص‌های بهینه مدیریت می‌کنند.
- مدیریت فایل‌ها: مسیرهای خروجی را ذخیره و بازیابی می‌کند.
- پاکسازی داده‌ها: وظایف منقضی‌شده را حذف می‌کند.

کتابخانه‌ها و ادغام‌های خارجی:

- پردازش رسانه: از کتابخانه‌های مختلف برای پردازش صوت و تصویر استفاده می‌شود.
- سرور WSGI: برای استقرار پایدار استفاده می‌شود.
- وابستگی‌های اضافی: برای پردازش تصویر و هش رمزعبور استفاده می‌شوند.

ثبت و نظارت:

- لاگ‌گیری چندسطحی: لاگ‌ها در فایل‌های جداگانه ذخیره می‌شوند.

- ردیابی بلادرنگ: امکان نظارت بلادرنگ را فراهم می‌کند.

- اشکال‌زدایی: خطاها را با جزئیات ثبت می‌کند.

نقش	توضیحات	مؤلفه
متمرکز کردن منطق وب وAPI	سرور واحد فلاسک	معماری مونولیتیک
مدیریت مسیرها و قالب‌ها	چارچوب وب	فلاسک
ذخیره‌سازی کاربران، وظایف، لاگ‌ها و دستگاه‌ها	پایگاه دادهNoSQL	مانگو دی‌بی
ادغام صدا و ویدئو	ابزار پردازش رسانه	اف‌اف‌ام‌پگ
اجرای برنامه فلاسک	سرورWSGI	ویترس

جدول 1: معماری استفاده شده

این معماری با ترکیب قابلیت‌های صنعتی و سادگی آکادمیک، سیستمی انعطاف‌پذیر ارائه می‌دهد که وظایف چندرسانه‌ای را به‌صورت کارآمد مدیریت می‌کند.

فصل سوم

پیاده‌سازی

فناوری های استفاده‌شده

پیاده‌سازی این پروژه مجموعه‌ای از ابزارها، کتابخانه‌ها، و زبان‌های برنامه‌نویسی متن‌باز را به‌طور یکپارچه ترکیب می‌کند که به‌طور خاص برای اتوماسیون بک‌اند، پردازش رسانه‌های چندگانه، و ادغام سیستمی طراحی شده‌اند. انتخاب هر یک از این اجزا با در نظر گرفتن معیارهایی نظیر عملکرد بالا، پشتیبانی قوی جامعه توسعه‌دهندگان، قابلیت گسترش برای نیازهای آینده، و کاربرد عملی در محیط‌های واقعی انجام شده است. اسکریپت‌های ارائه‌شده (M.py، server.py، و db.py) نشان‌دهنده یک طراحی مدولار و استفاده استراتژیک از ابزارهای پیشرفته هستند که با هدف تحقق نیازمندی‌های پروژه و تضمین پایداری و انعطاف‌پذیری سیستم به کار گرفته شده‌اند. در ادامه، جزئیات بیشتری درباره هر ابزار و نقش آن در پروژه با توجه به اسکریپت‌ها ارائه می‌شود:

– پایتون: پایتون به‌عنوان زبان اصلی برنامه‌نویسی این پروژه انتخاب شده است، زیرا کتابخانه استاندارد جامع، مجموعه گسترده‌ای از ماژول‌های شخص ثالث، و سینتکس ساده و خوانا، آن را برای توسعه سریع نمونه‌های اولیه و همچنین پیاده‌سازی ویژگی‌های با کیفیت تولیدی مناسب می‌سازد. در db.py، استفاده از ThreadPoolExecutor و ماژول threading برای مدیریت وظایف ناهمزمان و قفل‌های همزمانی مانند heap_lock و scheduler_lock، نشان‌دهنده توانایی پایتون در مدیریت

پردازش‌های پیچیده است. در M.py، کتابخانه‌هایی مانند MoviePy، Pillow، Pydub، و Mutagen برای پردازش رسانه‌های تصویری و صوتی استفاده شده‌اند که با کدهای پایتونی ساده ادغام شده‌اند. server.py نیز از فریم‌ورک Flask بهره می‌برد که با استفاده از پایتون، رابط‌های وب پویا و APIهای RESTful را با حداقل پیچیدگی پیاده‌سازی می‌کند. این انعطاف‌پذیری پایتون امکان هماهنگی بین ماژول‌های مختلف پروژه را فراهم کرده و توسعه و نگهداری را آسان‌تر می‌سازد.

- Flask: فریم‌ورک Flask به دلیل سبک بودن، مدولار بودن، و سهولت استفاده، برای توسعه سریع APIهای RESTful و داشبوردهای وب انتخاب شده است. در server.py، مسیرهایی مانند /api/admin/dashboard-data، /api/user/devices/<user_id>، و /api/authorization نشان‌دهنده استفاده از Flask برای مدیریت درخواست‌های HTTP، ارائه محتوای پویا بر اساس نقش کاربر (مدیر یا کاربر)، و ایجاد رابط‌های کاربری تعاملی هستند. Flask امکان ادغام با افزونه‌هایی مانند Flask-Login برای مدیریت جلسات کاربر و Flask-Limiter برای محدودسازی نرخ درخواست‌ها را فراهم می‌کند. در نسخه سازمانی پروژه، از FastAPI به دلیل پشتیبانی از درخواست‌های ناهمزمان و مقیاس‌پذیری بالاتر استفاده شده است، اما در این نسخه آکادمیک، Flask به دلیل سادگی و سرعت توسعه ترجیح داده شده است. این انتخاب تعادل مناسبی بین عملکرد و سهولت پیاده‌سازی ایجاد کرده است.

- MongoDB: این پایگاه داده NoSQL به دلیل پشتیبانی از طرح‌واره انعطاف‌پذیر و قابلیت مدیریت داده‌های پویا و بدون ساختار، برای ذخیره‌سازی اطلاعات پروژه انتخاب شده است. MongoDB برای ذخیره داده‌هایی مانند پروفایل‌های کاربران، متادیتای دستگاه‌ها، وظایف زمان‌بندی‌شده، وضعیت‌های اجرای وظایف، و لاگ‌ها بسیار مناسب است. در db.py، مجموعه‌هایی مانند users، devices، tasks، purgatory_col، processd_col، و bad_tasks_col با استفاده از توابعی مانند save_to_mongo، retrieve_from_mongo، و create_indexes مدیریت می‌شوند. شاخص‌های تعریف‌شده در create_indexes عملکرد

پرس‌وجوها را بهبود می‌بخشند، در حالی که مکانیزم‌های TTL (زمان حیات) در مجموعه‌هایی مانند purgatory_col داده‌های منقضی‌شده را به‌صورت خودکار حذف می‌کنند. این ویژگی‌ها MongoDB را به گزینه‌ای ایده‌آل برای مدیریت داده‌های پویا و مقیاس‌پذیر پروژه تبدیل کرده‌اند.

کاربرد	مجموعه
ذخیره‌سازی اطلاعات کاربران (نام، نقش، رمزعبور)	کاربران
مدیریت وظایف زمان‌بندی‌شده و وضعیت آن‌ها	وظایف
ثبت لاگ‌های سیستم و اجرای وظایف	لاگ‌ها
ردیابی اطلاعات دستگاه‌ها و روال‌ها	دستگاه‌ها

جدول 2: مجموعه‌های MongoDB و کاربرد آن‌ها

- Redis: Redis به‌عنوان یک ذخیره‌ساز داده در حافظه برای پیاده‌سازی محدودسازی نرخ و کش کردن داده‌ها استفاده شده است. در server.py، ماژول Flask-Limiter با Redis ادغام شده تا از سوءاستفاده از APIهای حساس مانند /api/authorization (با محدودیت "5 درخواست در دقیقه") جلوگیری کند. Redis همچنین برای مدیریت جلسات کاربر و کش کردن داده‌های پراستفاده، مانند اطلاعات داشبورد، به کار می‌رود که بار روی MongoDB و سرور را کاهش می‌دهد. این ابزار با سرعت بالا و پشتیبانی از ساختارهای داده متنوع، عملکرد سیستم را در سناریوهای با ترافیک بالا بهبود می‌بخشد.

- MoviePy و FFmpeg: این دو ابزار ستون فقرات پردازش رسانه‌ای پروژه را تشکیل می‌دهند. FFmpeg در M.py به‌عنوان موتور اصلی تبدیل و ادغام رسانه عمل می‌کند و وظایفی مانند ترکیب فایل‌های صوتی و تصویری، تبدیل فرمت‌ها، و بهینه‌سازی خروجی را با کارایی بالا انجام می‌دهد. MoviePy رابط سطح بالایی در پایتون ارائه می‌دهد که در تابع merge_media برای ایجاد ویدئوها با استفاده از کلیپ‌های

تصویری (ImageClip) و صوتی (AudioFileClip) استفاده شده است. این ترکیب امکان پردازش رسانه‌ای پیچیده را با کدهای ساده و قابل‌فهم فراهم می‌کند و از وابستگی به ابزارهای خارجی سنگین جلوگیری می‌نماید. تابع
_send_merge_request_offline در db.py نیز از FFmpeg برای اجرای آفلاین درخواست‌های ادغام استفاده می‌کند که برای محیط‌های حساس به حریم خصوصی مناسب است.

- Pillow (PIL): این کتابخانه برای پردازش تصاویر، از جمله تغییر اندازه، فشرده‌سازی، و حذف متادیتا، استفاده می‌شود. در M.py، تابع process_image از Pillow برای آماده‌سازی تصاویر با فرمت‌های استاندارد (مانند JPEG) و تنظیم ابعاد و کیفیت آن‌ها قبل از ادغام با صوت استفاده می‌کند. این پردازش‌ها کیفیت و سازگاری تصاویر را برای تولید ویدئوهای نهایی تضمین می‌کنند و از مشکلات احتمالی مانند ناسازگاری فرمت یا حجم بالای فایل جلوگیری می‌نمایند.

- Mutagen و Pydub: این کتابخانه‌های پردازش صوتی برای تبدیل فرمت، برش، تنظیم مدت‌زمان، و حذف متادیتای فایل‌های صوتی به کار می‌روند. در M.py، تابع process_audio از Pydub برای تبدیل فایل‌های صوتی به فرمت MP3 و تنظیم پارامترهایی مانند bitrate و مدت‌زمان استفاده می‌کند، در حالی که Mutagen برای مدیریت و حذف متادیتای صوتی (مانند تگ‌های ID3) کاربرد دارد. این ابزارها هماهنگی بین صوت و تصویر را در فرآیند ادغام رسانه تضمین می‌کنند.

- Waitress: این سرور WSGI با کیفیت تولیدی در server.py برای اجرای برنامه Flask استفاده شده است. برخلاف سرور توسعه پیش‌فرض Flask که برای محیط‌های آزمایشی مناسب است، Waitress عملکرد و هم‌زمانی بهتری ارائه می‌دهد و برای استقرارهای نزدیک به تولید ایده‌آل است. این سرور با پشتیبانی از مدیریت درخواست‌های همزمان، پایداری سیستم را در محیط‌های عملیاتی افزایش می‌دهد.

- Logging (RotatingFileHandler): سیستم لاگ‌گیری پروژه برای ثبت وقایع، اشکال‌زدایی، ممیزی، و نظارت بر سلامت سیستم طراحی شده است. در db.py، از

RotatingFileHandler برای ذخیره لاگ‌ها در فایل‌های جداگانه (SERVER.log، CLASS.log، DATABASE.log) با سطوح مختلف (INFO، DEBUG، ERROR) استفاده شده است. این فایل‌ها با حداکثر اندازه مشخص (مانند 10 مگابایت) و چرخش خودکار مدیریت می‌شوند تا از پر شدن فضای دیسک جلوگیری شود. server.py نیز از لاگ‌گیری مشابه برای ثبت درخواست‌های HTTP، خطاها، و فعالیت‌های کاربر استفاده می‌کند. تابع store_log در db.py لاگ‌ها را به مجموعه logs در MongoDB اضافه می‌کند، که امکان ردیابی بلادرنگ و تحلیل پس از وقوع را فراهم می‌سازد.

شرح ماژول‌ها

پروژه از چندین ماژول تشکیل شده است که هر کدام مسئولیت مشخصی دارند و با طراحی مدولار، جداسازی منطقی وظایف و قابلیت نگهداری آسان را تضمین می‌کنند. اسکریپت‌های M.py، server.py، و db.py به‌صورت هماهنگ عمل می‌کنند تا عملکردهای موردنیاز سیستم را به‌طور کامل محقق سازند. در ادامه، جزئیات بیشتری درباره هر ماژول ارائه می‌شود:

1. db.py – زمان‌بندی وظایف و مدیریت پایگاه داده:

– کلاس PersistentTaskScheduler به‌صورت تک‌نمونه (singleton) پیاده‌سازی شده تا از ایجاد نمونه‌های متعدد و تداخل در حالت‌های سیستم جلوگیری کند. متد __new__ در db.py با استفاده از قفل نخ (threading.Lock) اطمینان می‌دهد که تنها یک نمونه از این کلاس وجود دارد. این طراحی برای مدیریت مرکزی وظایف و حفظ یکپارچگی داده‌ها حیاتی است.

– صف وظایف با استفاده از یک هیپ در حافظه (heap) و قفل همزمانی (heap_lock) مدیریت می‌شود. تابع process_due_tasks_ به‌طور مداوم

وظایف آماده اجرا را بر اساس زمان‌بندی (فیلد next_execution) بررسی و پردازش می‌کند. این فرآیند توسط تابع start_scheduler_ به‌صورت ناهمزمان اجرا می‌شود.

- ارتباط با MongoDB از طریق توابعی مانند save_to_mongo، retrieve_from_mongo، و update_output_path برقرار می‌شود. مجموعه‌هایی مانند tasks_col برای ذخیره وظایف زمان‌بندی‌شده، purgatory_col برای ردیابی وضعیت اجرای بلادرنگ، processd_col برای آرشیو نتایج، و bad_tasks_col برای وظایف ناموفق استفاده می‌شوند. شاخص‌های تعریف‌شده در create_indexes پرس‌وجوها را بهینه‌سازی می‌کنند.

- مکانیزم‌های تلاش مجدد در تابع process_task_ با حداکثر سه تلاش (MAX_RETRIES) پیاده‌سازی شده‌اند. در صورت بروز خطا، مانند مشکلات پردازش رسانه یا اتصال به سرور، سیستم تلاش مجدد می‌کند و در صورت شکست نهایی، وظیفه به bad_tasks_col منتقل شده و خطا در لاگ‌ها ثبت می‌شود. منطق‌های بازگشتی مانند handle_expired_task_ وظایف منقضی‌شده را شناسایی و فایل‌های مرتبط را از سیستم حذف می‌کنند تا فضای ذخیره‌سازی آزاد شود.

- لاگ‌گیری پیشرفته با استفاده از RotatingFileHandler و تابع store_log پیاده‌سازی شده است. این سیستم نه‌تنها وقایع را در فایل‌های محلی ذخیره می‌کند، بلکه آن‌ها را به MongoDB منتقل می‌کند تا امکان تحلیل داده‌ها و تولید گزارش‌های جدولی (get_device_log_table_data) و نموداری(get_device_log_chart_data) فراهم شود.

توضیحات	ویژگی
اجرای موازی وظایف	هم‌زمانی
بازآزمایی وظایف ناموفق تا 3 بار	مکانیزم تلاش مجدد
ذخیره‌سازی وظایف در مانگو دی‌بی	پایداری
محدود کردن درخواست‌های API با ردیس	محدودسازی نرخ

جدول 3: ویژگی های پشته زمانبندی

2. M.py – پردازش رسانه:

– این ماژول شامل توابعی برای اعتبارسنجی فرمت فایل‌ها، پردازش تصاویر و صوت، و تولید ویدئوهای نهایی است. تابع process_image با استفاده از Pillow تصاویر را به فرمت استاندارد (مانند JPEG) تبدیل کرده، اندازه آن‌ها را تنظیم و متادیتا را حذف می‌کند. تابع process_audio با Pydub فایل‌های صوتی را به MP3 تبدیل کرده و پارامترهایی مانند bitrate و مدت‌زمان را تنظیم می‌کند.

– تابع merge_media از MoviePy برای ترکیب تصویر و صوت استفاده می‌کند. این تابع یک کلیپ تصویری (ImageClip) با مدت‌زمان مشخص (مانند 60 ثانیه) ایجاد کرده و آن را با کلیپ صوتی (AudioFileClip) ادغام می‌کند. خروجی به‌صورت فایل MP4 با کدک libx264 و نرخ فریم 24 ذخیره می‌شود. FFmpeg در پس‌زمینه این فرآیند را تقویت می‌کند.

– مدیریت دایرکتوری‌های ورودی و خروجی با دقت انجام می‌شود. توابع این ماژول مسیرهای موقت را برای پردازش فایل‌ها ایجاد کرده و پس از اتمام، آن‌ها را پاکسازی می‌کنند. تابع update_output_path در db.py مسیرهای خروجی نهایی را در MongoDB ذخیره می‌کند.

– قابلیت‌هایی مانند گسترش صوت (extend_audio) و تطبیق مدت‌زمان صوت و تصویر در merge_media و process_audio پیاده‌سازی شده‌اند تا هماهنگی کامل بین رسانه‌ها تضمین شود. این ویژگی‌ها برای تولید ویدئوهای حرفه‌ای و بدون نقص ضروری هستند.

3. server.py – رابط برنامه وب:

– احراز هویت کاربران با استفاده از Flask-Login و مدیریت جلسات در مسیر /api/authorization پیاده‌سازی شده است. تابع login_validation در db.py با استفاده از bcrypt رمزعبور را اعتبارسنجی کرده و نقش کاربر (admin یا user) را

تأیید می‌کند. این فرآیند با محدودسازی نرخ (5 درخواست در دقیقه) توسط Flask-Limiter و Redis ایمن شده است.

– داشبوردهای جداگانه برای کاربران و مدیران از طریق مسیرهای /api/admin/* و /api/user/* ارائه می‌شوند. قالب‌های HTML مانند dashboard.html و index.html با استفاده از Bootstrap و جاوااسکریپت رندر دینامیک را پشتیبانی می‌کنند. مسیر /api/admin/dashboard-data داده‌های بلادرنگ را از db.py (تابع admin_dash_data_recive) بارگذاری می‌کند.

– درخواست‌های HTTP به توابع بک‌اند در db.py هدایت می‌شوند. برای مثال، مسیر /api/admin/new_user از save_new_user برای افزودن کاربر جدید، و /api/admin/devices/update از update_device_details برای به‌روزرسانی دستگاه‌ها استفاده می‌کند. این معماری جداسازی لایه ارائه و منطق کاربردی را تضمین می‌کند.

– محدودسازی نرخ و مدیریت همزمانی با Redis و قفل‌های نخ در مسیرهای حساس اعمال شده‌اند. این ویژگی‌ها از سوءاستفاده و بار اضافی روی سرور جلوگیری می‌کنند و تجربه کاربری پایداری ارائه می‌دهند.

سطح دسترسی	روش	هدف	نقطه پایانی
همه	POST	ورود کاربر	/api/authorization
مدیر	POST	ثبت کاربر جدید	/api/admin/new_user
مدیر	POST	افزودن دستگاه جدید	/api/admin/new_device
کاربر	POST	آپلود رسانه برای دستگاه	/api/user/upload/

جدول 4: مرور نقاط پایانی

4. ابزارهای کمکی:

- قالب‌های HTML، CSS، و جاوااسکریپت در server.py رابط کاربری تعاملی را ایجاد می‌کنند. فایل‌هایی مانند dashboard.html از Bootstrap برای طراحی پاسخ‌گو و جاوااسکریپت برای به‌روزرسانی‌های بلادرنگ استفاده می‌کنند.

- آپلود تصاویر پروفایل با استفاده از base64 و Pillow در server.py و توابع db.py مانند save_user_info مدیریت می‌شود. این فرآیند شامل اعتبارسنجی فرمت و ذخیره امن تصاویر است.

- ماژول‌های itertools، threading، و multiprocessing در db.py و M.py برای بهبود عملکرد استفاده شده‌اند. برای مثال، ThreadPoolExecutor در db.py

وظایف را به‌صورت موازی اجرا می‌کند، و itertools در M.py برای پردازش دسته‌ای فایل‌ها کاربرد دارد. این ابزارها انیمیشن‌ها و پردازش‌های سیستمی را روان‌تر می‌کنند.

قطعات کد مهم

الگوی تک‌نمونه برای زمان‌بندی وظایف:

```python
class PersistentTaskScheduler:
    _instance = None
    _lock = threading.Lock()

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            with cls._lock:
                if cls._instance is None:
                    cls._instance = super(PersistentTaskScheduler, cls).__new__(cls)
                    cls._instance._initialized = False
        return cls._instance
```

این کد در db.py با استفاده از قفل نخ، یک نمونه واحد از PersistentTaskScheduler را تضمین می‌کند. این طراحی از تداخل در زمان‌بندی وظایف جلوگیری کرده و مدیریت متمرکز وظایف را ممکن می‌سازد.

اجرای وظیفه با منطق تلاش مجدد:

```python
def _process_task(self, task, current_time):
    try:
        if data['status'] == 'yes:'
            self.executor.submit(self._handle_media_merge, device_id, image_path, audio_path)
    except Exception as e:
        retries += 1
        if retries >= self.MAX_RETRIES:
            self._handle_failed_task(...)
```

این قطعه در db.py اجرای وظایف را با حداکثر سه تلاش مجدد مدیریت می‌کند. در صورت بروز خطا، مانند خرابی در ادغام رسانه، سیستم تلاش مجدد کرده و در صورت شکست نهایی، وظیفه را به bad_tasks_col منتقل و خطا را لاگ می‌کند.

تابع ادغام رسانه (MoviePy):

```python
def merge_media(photo_path, audio_path, output_dir, file_name):
    photo_path = process_image(photo_path)
    audio_path = process_audio(audio_path)
    image = ImageClip(photo_path).set_duration(60)
    audio = AudioFileClip(audio_path)
    video = image.set_audio(audio)
    output_path = os.path.join(output_dir, f"{file_name}.mp4")
    video.write_videofile(output_path, codec='libx264', fps=24)
```

این تابع در M.py با استفاده از MoviePy و FFmpeg، تصویر و صوت را به ویدئوی MP4 تبدیل می‌کند. فرآیندهای اولیه توسط process_image و process_audio انجام می‌شوند تا کیفیت و سازگاری رسانه‌ها تضمین شود.

مسیر API Flask برای اعتبارسنجی کاربر:

```python
@app.route('/api/authorization', methods=['POST'])
@limiter.limit("5 per minute")
def validation():
    username = request.form['username']
    password = request.form['password']
    status = login_validation(username, password)
```

```
    if status in ['admin', 'user']:
        user_id = get_ai(username)
        user = User(id=user_id, username=username,
role=status)
        login_user(user)
        return jsonify({'success': True})
    return jsonify({'success': False, 'error': 'Invalid credentials'})
```

این کد در server.py ورود کاربران را با استفاده از login_validation در db.py و محدودسازی نرخ با Redis مدیریت می‌کند. در صورت موفقیت، کاربر به داشبورد مربوطه هدایت می‌شود.

این پیاده‌سازی مدولار، با استفاده از ابزارهای متن‌باز و طراحی آفلاین (مانند send_merge_request_offline_ در db.py)، سیستمی قابل‌نگهداری، مقیاس‌پذیر، و سازگار با سناریوهای حساس به حریم خصوصی ارائه می‌دهد. ادغام کتابخانه‌های قدرتمند مانند FFmpeg، MongoDB، و Redis، همراه با طراحی مدولار، عملکرد بالا و انعطاف‌پذیری را برای استفاده‌های آکادمیک و صنعتی تضمین می‌کند.

فصل چهارم

آزمایش و ارزیابی

نمای کلی

آزمایش یک مرحله اساسی در مهندسی نرم‌افزار است که برای اطمینان از عملکرد صحیح، کارآمد، و قابل‌اعتماد سیستم تحت شرایط مختلف طراحی شده است. در این پروژه، آزمایش‌ها به‌صورت تکراری در طول چرخه توسعه انجام شدند تا عملکرد سیستم، صحت خروجی‌های رسانه‌ای، و مقاومت در برابر خطاها تأیید شوند. رویکرد آزمایش شامل ترکیب آزمایش‌های واحد، یکپارچگی، و سیستمی بود تا پوشش جامعی از منطق بک‌اند و تعاملات کاربرمحور فراهم شود. اسکریپت‌هایM.py ، server.py، و db.py به‌عنوان پایه‌های اصلی سیستم مورد آزمایش قرار گرفتند، و تست‌ها به‌گونه‌ای طراحی شدند که قابلیت‌های کلیدی مانند زمان‌بندی وظایف، پردازش رسانه، و رابط کاربری وب را به‌طور کامل ارزیابی کنند. این فرآیند با استفاده از ابزارهای استاندارد و لاگ‌گیری دقیق انجام شد تا مشکلات به‌سرعت شناسایی و رفع شوند.

موارد آزمایشی

پروژه شامل مجموعه گسترده‌ای از موارد آزمایشی بود که برای پوشش عملیات عادی، موارد مرزی، و مدیریت خطاها طراحی شدند. این موارد آزمایشی با توجه به ماژول‌های مختلف سیستم) زمان‌بندی وظایف، پردازش رسانه، وAPI ها/احراز هویت (تنظیم شدند تا اطمینان حاصل شود که هر جزء به‌درستی عمل می‌کند. در ادامه، جزئیات بیشتری درباره هر مورد آزمایشی با تمرکز بر اسکریپت‌های مربوطه ارائه می‌شود:

زمان‌بندی وظایف:

مورد آزمایشی 1: افزودن وظیفه جدید با داده‌های معتبر

- ورودی: شناسه دستگاه معتبر، روال مانند instruction-A یاcustom ، مسیرهای فایل صوتی/تصویری
- خروجی مورد انتظار: وظیفه به صف وظایف اضافه شده و به‌درستی زمان‌بندی می‌شود
- توضیحات: این تست تابع add_task در db.py را بررسی کرد، که وظیفه را به مجموعه tasks در MongoDB اضافه می‌کند و با PersistentTaskSchedulerدر هیپ در حافظه قرار می‌دهد. تابع create_indexesدر db.py اطمینان داد که پرس‌وجوهای مرتبط با device_idسریع انجام شوند. تست با ورودی‌های معتبر) مانند مسیرهای فایل JPEGو (MP3 انجام شد و تأیید کرد که فیلدهای next_execution و routineبه‌درستی تنظیم شده‌اند.

توضیحات	وضعیت
دستگاه عملیاتی است	فعال
دستگاه آفلاین است	غیرفعال
دستگاه با مشکل مواجه شده	خطا

جدول 5: حالت‌های وضعیت دستگاه

مورد آزمایشی 2: اجرای وظیفه در زمان تعیین‌شده

- ورودی: زمان اجرای وظیفه با زمان فعلی مطابقت دارد
- خروجی مورد انتظار: وضعیت وظیفه به "completed" تغییر کرده و ویدئو تولید می‌شود
- توضیحات: این تست عملکرد تابع _process_due_tasks در db.py را ارزیابی کرد، که وظایف آماده را از هیپ استخراج کرده و با ThreadPoolExecutorاجرا می‌کند. تابع _handle_media_merge در db.pyوظیفه را به merge_media در M.py ارسال کرد تا ویدئو تولید شود. تست با تنظیم زمان next_execution برابر با datetime.utcnow()انجام شد و تأیید کرد که خروجی ویدئویی در مسیر مشخص‌شده با update_output_pathذخیره شده و وضعیت در processd_col به‌روزرسانی می‌شود.

مورد آزمایشی 3: تلاش مجدد وظیفه در صورت شکست

- ورودی: فایل صوتی/تصویری خراب یا ناموجود
- خروجی مورد انتظار: شمارنده تلاش مجدد افزایش یافته، وظیفه دوباره در صف قرار گرفته یا پس از حداکثر تلاش‌ها به‌عنوان شکست‌خورده علامت‌گذاری می‌شود
- توضیحات: این تست مکانیزم تلاش مجدد در process_task_ در db.py را بررسی کرد، که با MAX_RETRIES (سه تلاش) خطاها را مدیریت می‌کند. با ارائه فایل معیوب (مانند صوت خالی)، تست تأیید کرد که retries افزایش یافته و در صورت شکست نهایی، وظیفه به bad_tasks_col منتقل می‌شود. تابع log_task_failure_در db.py جزئیات خطا را در logs ثبت کرد.

مورد آزمایشی 4: تبدیل فرمت صوتی پشتیبانی‌نشده

- ورودی: فایل با فرمت wma. یاogg.
- خروجی مورد انتظار: تبدیل بهMP3 ، حذف متادیتا
- توضیحات: این تست تابع process_audio در M.py را ارزیابی کرد، که با استفاده از Pydub فایل‌های صوتی را به MP3 تبدیل می‌کند Mutagen .برای حذف متادیتا مانند تگ‌هایID3 استفاده شد. تست با فایل‌های wma. انجام شد و تأیید کرد که خروجی MP3 با bitrate استاندارد تولید شده و متادیتا حذف شده است. این فرآیند برای اطمینان از سازگاری در merge_media حیاتی بود.
- مورد آزمایشی 5: تغییر اندازه تصویر بزرگ
- ورودی: فایل PNG با رزولوشن بالا
- خروجی مورد انتظار: تصویر به JPG با رزولوشن 854 x 480 تبدیل می‌شود
- توضیحات: این تست تابع process_image در M.py را بررسی کرد، که با Pillowتصاویر را تغییر اندازه داده و به JPG تبدیل می‌کند. تست با یک فایل

PNGبزرگ انجام شد و تأیید کرد که خروجی با ابعاد 854 x 480 و بدون متادیتا تولید شده است. این تنظیمات برای استانداردسازی ورودی‌های merge_media و کاهش حجم فایل ضروری بودند.

مورد آزمایشی 6: ورود مدیر

- ورودی: اعتبارنامه معتبر مدیر
- خروجی مورد انتظار: دسترسی به داشبورد مدیریتی
- توضیحات: این تست مسیر /api/authorization در server.py را بررسی کرد، که با تابع login_validation در db.py احراز هویت را انجام می‌دهد. با ارائه نام کاربری و رمزعبور معتبر هش‌شده با bcrypt، تست تأیید کرد که کاربر با نقش adminبه داشبورد (dashboard.html) هدایت شده و جلسه با Flask-Login ایجاد می‌شود.

- مورد آزمایشی 7: تلاش ورود نامعتبر
- ورودی: نام کاربری/رمزعبور اشتباه
- خروجی مورد انتظار: پیام خطا، عدم ایجاد جلسه
- توضیحات: این تست همان مسیر /api/authorization را با ورودی‌های نادرست آزمایش کرد. تابع login_validation عدم تطابق رمزعبور را شناسایی کرد و server.py پاسخ JSON با پیام "Invalid credentials" و success: Falseبازگشت داد. تست تأیید کرد که هیچ جلسه‌ای ایجاد نشده و دسترسی رد می‌شود.

مورد آزمایشی 8: محدودسازی نرخ

- ورودی: چندین درخواست در مدت‌زمان کوتاه
- خروجی مورد انتظار IP :به‌صورت موقت مسدود شده(HTTP 429)
- توضیحات: این تست Flask-Limiter در server.py را با Redis آزمایش کرد. با ارسال بیش از 5 درخواست در دقیقه به/api/authorization ، تست تأیید کرد که پاسخ HTTP 429 (Too Many Requests) دریافت شده و IP به‌صورت موقت محدود می‌شود. این ویژگی از سوءاستفاده جلوگیری کرد.

انواع آزمایش

آزمایش واحد:
آزمایش‌های واحد برای توابع جداگانه، به‌ویژه در ماژول‌های پردازش رسانه و زمان‌بندی وظایف، نوشته شدند.

- اعتبارسنجی منطق بررسی پسوند فایل در process_image و process_audioدر M.py آزمایش شد تا اطمینان حاصل شود که فقط فرمت‌های مجاز مانند JPEG و MP3 پذیرفته می‌شوند.
- خروجی‌های تبدیل تصویر و صوت با استفاده از ورودی‌های نمونه در M.py تأیید شدند. برای مثال، تست‌ها بررسی کردند که تصاویر به 854 x 480 و صوت به MP3تبدیل شوند.
- مکانیزم تلاش مجدد و ترتیب صف وظایف در db.py با استفاده از توابع زمانی جعلی (mock time) آزمایش شدند. تابع process_task_ با ورودی‌های خراب تست شد تا رفتار MAX_RETRIES تأیید شود.

آزمایش یکپارچگی:

آزمایش‌های یکپارچگی بر تعاملات بین ماژول‌های مختلف تمرکز داشتند.

- جریان کامل از ارسال وظیفه تا تولید خروجی رسانه آزمایش شد. این تست شامل افزودن وظیفه با add_task در db.py ، زمان‌بندی با PersistentTaskScheduler، پردازش رسانه با merge_media در M.py ، و ذخیره خروجی با update_output_path بود.
- تأیید شد که ایجاد وظیفه در) MongoDB با save_to_mongo در (db.py به‌روزرسانی‌های زمان‌بندی را در process_due_tasks_ فعال می‌کند.
- پاسخ‌های API برای اقدامات مرتبط با وظیفه مانند POST به /api/admin/new_taskو GET از /api/user/devices/<user_id> در server.pyآزمایش شدند تا اطمینان حاصل شود که داده‌ها به‌درستی منتقل می‌شوند.

آزمایش سیستمی:

آزمایش سیستمی تأیید کرد که برنامه در محیط استقرار موردنظر به‌درستی کار می‌کند.

- ورود همزمان کاربران و آپلود رسانه با استفاده از دستگاه‌های جعلی شبیه‌سازی شد. مسیرهای /api/authorization و /api/user/upload/<device_id> در server.pyتحت بار تست شدند.
- اجرای وظایف تحت بار با استفاده از چندین دستگاه جعلی آزمایش شد، که عملکرد ThreadPoolExecutor در db.py را ارزیابی کرد. تست‌ها تأیید کردند که سیستم تا 10 وظیفه همزمان را بدون افت عملکرد مدیریت می‌کند.
- یکپارچگی لاگ‌گیری و ثبت خطاها تحت ورودی‌های غیرعادی (مانند فایل‌های خراب) آزمایش شد. تابع store_log در db.py و فایل‌های SERVER.log و

DATABASE.logبررسی شدند تا اطمینان حاصل شود که خطاها به‌درستی ثبت می‌شوند.

ردیابی و رفع اشکالات

برای ردیابی اشکالات از رویکرد مبتنی بر لاگ و کنترل کیفیت دستی استفاده شد. فایل‌های لاگ server.log، scheduler.log، db.logدر طول آزمایش‌ها نظارت شدند، و مشکلات شناسایی‌شده در چرخه‌های تکراری مستند و رفع شدند. تابع store_log در db.pyو RotatingFileHandler در db.py و server.py اطلاعات دقیقی از وقایع فراهم کردند که برای اشکال‌زدایی ضروری بودند.

نمونه اشکالات:

مشکل: زمان‌بندی وظایف وظایف منقضی‌شده را نادیده می‌گرفت

- علت: منطق مقایسه نادرست در بررسیdatetime
- رفع: منطق datetime.now() در process_due_tasks_ در db.py اصلاح شد و با یک مورد آزمایشی تأیید گردید.

مشکل: خروجی رسانه بدون صدا بود

- علت: فایل صوتی به دلیل تبدیل ناموفق خالی بود
- رفع: بررسی اندازه فایل پس از تبدیل در process_audio در M.py اضافه شد و در صورت شکست، تلاش مجدد با فایل اصلی انجام شد

مشکل: وظایف تکراری در MongoDB درج شدند

- علت: عدم بررسی یکتایی در تولید شناسه وظیفه

- رفع: منطق هش‌گذاری در add_task در db.py اضافه شد تا ورودی‌های تکراری شناسایی شوند.

مشکل: داشبورد برای کاربران تازه‌ثبت‌شده بارگذاری نمی‌شد

- علت: نقش کاربر در برخی ورودی‌ها تعریف‌نشده بود
- رفع: نقش پیش‌فرض "user" در save_new_user در db.py تنظیم شد و بررسی اعتبارسنجی اضافه گردید.

ابزارها و لاگ‌گیری استفاده‌شده در آزمایش

- Postman : برای آزمایش نقاط پایانی API مانند ورود(/api/authorization) ، ارسال وظیفه(/api/admin/new_task) ، و مدیریت دستگاه (/api/admin/new_device)در server.py استفاده شد. این ابزار پاسخ‌های JSONو کدهای وضعیت HTTP را تأیید کرد.
- Pytest :فریم‌ورک آزمایش واحد خودکار برای تست توابع در) M.py مانند process_imageو db.py مانند process_task_ تست‌ها با ورودی‌های جعلی و خروجی‌های مورد انتظار اجرا شدند.
- کنترل کیفیت دستی: تعاملات کاربر با استفاده از مرورگر و محیط محلی شبیه‌سازی شد. قالب‌های dashboard.html و index.html در server.py برای بررسی رابط کاربری تست شدند.
- لاگ‌گیری: فایل‌های لاگ تقسیم‌شده بر اساس زیرسیستمserver.log ، scheduler.log، db.logدر db.py و server.py به شناسایی مشکلات خاص

ماژول کمک کردند. تابع store_log و RotatingFileHandler جزئیات وقایع را با سطوح مختلفINFO ، DEBUG، ERRORثبت کردند.

نتیجه	خروجی مورد انتظار	ورودی	مورد آزمایشی
قبول	وظیفه اضافه شده	داده‌های معتبر وظیفه	افزودن وظیفه
قبول	وظیفه اجرا شده	وظیفه زمان‌بندی‌شده	اجرای وظیفه
قبول	خطا مدیریت شده	فرمت صوتی ناموفق	تبدیل رسانه
قبول	دسترسی به داشبورد	اعتبارنامه معتبر	ورود مدیر

جدول 6: موارد آزمایشی و نتایج

اسکریپت تست ارائه‌شده

اسکریپت تست ارائه‌شده (test_project.py) با استفاده از فریم‌ورک Pytest طراحی شده تا موارد کلیدی ذکرشده در بخش آزمایش را پوشش دهد و با ساختار پروژه شامل اسکریپت‌های M.py، server.pyو db.pyهماهنگ باشد. این اسکریپت از کتابخانه unittest.mockبرای شبیه‌سازی وابستگی‌های خارجی مانندMongoDB ، Redisو عملیات فایل استفاده می‌کند تا تست‌ها به‌صورت ایزوله اجرا شوند. همچنین نظرات فارسی

موجود در کد، هر بخش و مورد آزمایشی را توضیح می‌دهند تا درک و نگهداری آن آسان‌تر شود. در ادامه، جزئیات بیشتری درباره اجزای اسکریپت ارائه می‌شود:

- **ساختار و فیکسچرها:**

اسکریپت شامل فیکسچرهای setup_mongo، setup_redis و clientاست که به‌ترتیب برای شبیه‌سازی پایگاه دادهMongoDB ، ذخیره‌ساز Redis و کلاینت تست Flask به کار می‌روند. این فیکسچرها با استفاده از patchدر unittest.mockوابستگی‌ها را کنترل کرده و تست‌ها را مستقل از سرویس‌های واقعی می‌سازند. برای مثال، فیکسچر setup_mongoیک شیء جعلی از مجموعه‌های MongoDB ایجاد می‌کند که در تستی مانند test_add_new_task_valid_dataمورد استفاده قرار می‌گیرد.

- **آزمایش‌های زمان‌بندی وظایف:**

  - test_add_new_task_valid_data:بررسی می‌کند که تابع add_taskدر db.pyوظیفه را با داده‌های معتبر (مانند شناسه دستگاه و مسیر فایل) به MongoDB و هیپ اضافه می‌کند. این تست از شبیه‌سازی save_to_mongoبرای تأیید ذخیره‌سازی استفاده می‌کند.

  - test_execute_task_on_time:عملکرد تابع process_due_tasks_را می‌سنجد که وظایف آماده را از هیپ اجرا کرده و ویدئو تولید می‌کند. با استفاده از نسخه شبیه‌سازی‌شده ThreadPoolExecutor.submit، بررسی می‌شود که وظیفه به تابع merge_mediaدر M.pyارسال شده و وضعیت آن به‌روزرسانی می‌شود.

- test_retry_task_on_failure: مکانیزم تلاش مجدد در تابع process_task را با شبیه‌سازی خطا (مثلاً فایل خراب) بررسی می‌کند. در این تست، افزایش شمارنده retries و انتقال وظیفه به bad_tasks_col پس از سه تلاش ناموفق بررسی می‌شود.

- **آزمایش‌های پردازش رسانه:**
  - test_convert_unsupported_audio: تابع process_audio در M.py را ارزیابی می‌کند تا اطمینان حاصل شود فایل‌های صوتی غیراستاندارد مانند WMA به MP3 تبدیل شده و متادیتا با استفاده از mutagen حذف می‌شود.
  - test_resize_large_image: عملکرد تابع process_image را می‌سنجد که تصاویر بزرگ مانند PNG را به رزولوشن 854 x480 و فرمت JPG با کمک Pillow تبدیل می‌کند.

- **آزمایش‌های API و احراز هویت:**
  - test_admin_login: ورود مدیر را از طریق مسیر /api/authorization در server.py با اطلاعات معتبر بررسی کرده و تأیید می‌کند که پاسخ JSON موفقیت‌آمیز است.
  - test_invalid_login: تلاش ورود با اطلاعات نادرست را آزمایش کرده و نمایش پیام خطای "Invalid credentials" را بررسی می‌کند.
  - test_rate_limiting: محدودسازی نرخ درخواست را با ارسال بیش از ۵ درخواست در دقیقه بررسی کرده و تأیید می‌کند که پاسخ HTTP با کد 429 دریافت می‌شود.

- **اجرا و پیش‌نیازها:**

برای اجرای تست‌ها، ابتدا باید Pytest نصب شود:

```
pip install pytest
```

سپس اجرای تست‌ها از طریق خط فرمان به صورت زیر انجام می‌شود:

```
pytest test_project.py -v
```

این اسکریپت نیازی به MongoDB یا Redis واقعی ندارد، چرا که از شبیه‌سازی استفاده می‌کند. این موضوع موجب سرعت و استقلال بیشتر تست‌ها از محیط اجرایی می‌شود.

- **محدودیت‌ها و قابلیت‌های گسترش:**

این اسکریپت تست‌های سیستمی پیچیده مانند تست بار بالا را ساده‌سازی کرده است. با این حال، می‌توان این نوع تست‌ها را با ابزارهایی مانند Locust توسعه داد. در صورتی که ساختار پروژه یا نام توابع تغییر یابد، ممکن است نیاز به به‌روزرسانی مسیرها و importها باشد. همچنین، امکان افزودن تست‌های بیشتر برای پوشش بهتر بخش‌هایی مانند رابط کاربری و فرم‌های ورودی وجود دارد.

در مجموع، این اسکریپت با طراحی مدولار و مستندسازی فارسی، ابزاری مؤثر برای ارزیابی عملکرد سیستم فراهم می‌کند و از طریق تست‌های دقیق، پایداری اجزای M.py، server.pyو db.pyرا تضمین می‌نماید. رفع اشکالات شناسایی‌شده در فرآیند تست، عملکرد کلی سیستم را بهبود بخشیده و زیرساختی مناسب برای توسعه‌های آینده ایجاد کرده است.

فصل پنجم

نتایج و بحث

آنچه با موفقیت انجام شد

این پروژه با موفقیت به هدف اصلی خود، یعنی ایجاد یک زمان‌بندی خودکار وظایف چندرسانه‌ای با رابط کاربری مبتنی بر وب و بک‌اند پایدار، دست یافت. اجزای مختلف سیستم در شرایط شبیه‌سازی‌شده مشابه محیط تولید، عملکردی قابل‌اعتماد و پایدار از خود نشان دادند. اسکریپت‌های ارائه‌شده (M.py، server.py، و db.py) نقش کلیدی در تحقق این موفقیت‌ها داشتند و هماهنگی بین ماژول‌ها عملکرد سیستم را تقویت کرد. هر یک از این اجزا با دقت طراحی و پیاده‌سازی شدند تا نیازمندی‌های عملکردی و غیرعملکردی پروژه را برآورده کنند، و نتایج حاصل از تست‌ها نشان‌دهنده پایداری و کارایی سیستم است. در ادامه، جزئیات بیشتری درباره هر یک از اجزای موفق با تمرکز بر اسکریپت‌ها و نقش آن‌ها ارائه می‌شود:

اجرای زمان‌بندی وظایف:

کلاس PersistentTaskScheduler در db.py به‌عنوان هسته مرکزی زمان‌بندی وظایف عمل کرد و وظایف را با دقت و کارایی مدیریت نمود. این کلاس با استفاده از یک هیپ در حافظه (heap) و قفل‌های همزمانی (heap_lock) اولویت‌بندی وظایف را بر اساس زمان اجرا (next_execution) انجام داد، که از اجرای منظم و بدون تداخل اطمینان حاصل کرد. تابع _process_due_tasks به‌صورت مداوم وظایف آماده را بررسی و به ThreadPoolExecutor با حداکثر چهار کارگر ارسال کرد تا به‌صورت ناهمزمان اجرا شوند. این طراحی امکان پردازش موازی وظایف را فراهم کرد و بار سیستم

را متعادل نمود. مکانیزم تلاش مجدد در تابع process_task_ با حداکثر سه تلاش (MAX_RETRIES) پیاده‌سازی شد، که خطاهای موقتی مانند قطعی در پردازش رسانه یا مشکلات اتصال را مدیریت کرد. در صورت شکست مکرر، وظایف به مجموعه bad_tasks_col منتقل شده و جزئیات خطا از طریق تابع store_log در فایل‌های لاگ (مانند DATABASE.log) و مجموعه logs در MongoDB ثبت شدند. تابع handle_expired_task_ وظایف منقضی‌شده را شناسایی و فایل‌های مرتبط را حذف کرد، که به بهینه‌سازی فضای ذخیره‌سازی و جلوگیری از تجمع داده‌های غیرضروری کمک کرد. این مکانیزم‌ها از دست رفتن داده‌ها را به حداقل رساندند و تکمیل وظایف را حتی در شرایط خطا تضمین کردند، که نشان‌دهنده پایداری بالای سیستم است.

پردازش رسانه:

ماژول پردازش رسانه در M.py با استفاده از FFmpeg و MoviePy، توانایی پردازش فرمت‌های متنوع صوتی و تصویری را فراهم کرد و به‌صورت خودکار آن‌ها را به فایل‌های ویدئویی با کیفیت بالا ادغام نمود. تابع merge_media تصاویر و صوت را با استفاده از ImageClip و AudioFileClip ترکیب کرده و ویدئوهایی با فرمت MP4، کدک libx264، و نرخ فریم 24 تولید کرد. این تابع مدت‌زمان تصویر را (مانند 60 ثانیه) با صوت هماهنگ کرد تا خروجی‌های یکپارچه‌ای ایجاد شود. توابع process_image و process_audio به ترتیب از Pillow و Pydub برای اعتبارسنجی و آماده‌سازی فایل‌ها استفاده کردند. process_image تصاویر را به فرمت استاندارد JPEG تبدیل کرده، ابعاد را تنظیم و متادیتا را حذف نمود، در حالی که process_audio فایل‌های صوتی را به MP3 با bitrate مشخص تبدیل کرد. این پیش‌پردازش‌ها کیفیت و سازگاری خروجی‌ها را تضمین کردند و از مشکلات احتمالی مانند ناسازگاری فرمت جلوگیری نمودند. تابع send_merge_request_offline_ در db.py فرآیند ادغام را به‌صورت آفلاین هماهنگ کرد، که برای محیط‌های حساس به حریم خصوصی یا با اتصال محدود ایده‌آل بود. این ماژول با مدیریت خطاهای احتمالی و ارائه خروجی‌های استاندارد، استانداردهای کیفی مورد انتظار را برآورده کرد و انعطاف‌پذیری بالایی در پردازش رسانه‌های متنوع نشان داد.

ادغام با MongoDB:

پایگاه داده MongoDB در db.py به دلیل طرح‌واره انعطاف‌پذیر و توانایی مدیریت داده‌های پویا، عملکردی برجسته ارائه داد. مجموعه‌هایی مانند users، devices، tasks، purgatory_col، processd_col، و logs امکان ذخیره و پرس‌وجوی داده‌های تودرتو مانند متادیتای وظایف، پروفایل‌های کاربران، و تاریخچه اجرا را فراهم کردند. توابع save_to_mongo و retrieve_from_mongo تعاملات پایدار با پایگاه داده را مدیریت کردند، در حالی که تابع create_indexes شاخص‌هایی برای فیلدهای پراستفاده (مانند device_id و next_execution) ایجاد نمود تا پرس‌وجوها سریع‌تر انجام شوند. مکانیزم‌های TTL (زمان حیات) در مجموعه‌هایی مانند purgatory_col داده‌های موقتی را پس از انقضا به‌صورت خودکار حذف کردند، که پایگاه داده را سبک و کارآمد نگه داشت. برای مثال، purgatory_col وضعیت اجرای بلادرنگ وظایف را با فیلدهایی مانند image_path و audio_path ردیابی کرد، در حالی که processd_col نتایج نهایی را آرشیو نمود. این طراحی مقیاس‌پذیری سیستم را تضمین کرد و امکان مدیریت حجم بالای داده‌ها را بدون افت عملکرد فراهم نمود. تابع update_output_path مسیرهای خروجی ویدئوها را در MongoDB ذخیره کرد، که دسترسی سریع به فایل‌های تولیدشده را ممکن ساخت.

داشبورد وب مبتنی بر Flask:

رابط کاربری وب در server.py با استفاده از فریم‌ورک Flask و قالب‌های HTML (مانند dashboard.html و index.html) به‌درستی عمل کرد و تجربه‌ای کاربرپسند ارائه داد. این رابط داشبوردهای جداگانه‌ای برای کاربران و مدیران از طریق مسیرهای /api/admin/* و /api/user/* فراهم کرد. برای مثال، مسیر /api/admin/dashboard-data داده‌های بلادرنگ مانند وضعیت دستگاه‌ها و وظایف را با استفاده از تابع admin_dash_data_recive در db.py بارگذاری کرد. مسیر /api/authorization با استفاده از Flask-Login و تابع login_validation در db.py احراز هویت کاربران را مدیریت کرد، و مسیرهای امن‌شده مانند /api/admin/devices/update از دسترسی غیرمجاز جلوگیری نمودند. مدیریت فرم‌ها و APIهای JSON، مانند /api/admin/new_user برای افزودن کاربر جدید، تعامل

روان بین فرانت‌اند و بک‌اند را تضمین کرد. استفاده از Bootstrap در قالب‌ها طراحی پاسخ‌گویی ایجاد کرد، و جاوااسکریپت ساده امکان به‌روزرسانی‌های دینامیک (مانند بارگذاری داده‌های لاگ) را فراهم نمود. این رابط کاربری با وجود سادگی، نیازهای پروژه را به‌خوبی برآورده کرد و تجربه‌ای منسجم ارائه داد.

محدودسازی نرخ و مدیریت جلسات با Redis:

سیستم با استفاده از Flask-Limiter و Redis در server.py، محدودسازی نرخ درخواست‌ها را به‌طور مؤثری اعمال کرد و از سوءاستفاده یا بار اضافی روی سرور جلوگیری نمود. برای مثال، مسیر /api/authorization با محدودیت "5 درخواست در دقیقه" از حملات احتمالی یا درخواست‌های مکرر غیرعادی محافظت شد. Redis به‌عنوان یک ذخیره‌ساز سریع در حافظه، جلسات موقت کاربران و داده‌های کش‌شده (مانند اطلاعات داشبورد یا وضعیت وظایف) را مدیریت کرد، که بار روی MongoDB را کاهش داد و زمان پاسخ‌گویی را بهبود بخشید. مسیرهایی مانند /api/user/devices/<user_id> از کش Redis برای ارائه سریع داده‌های کاربر استفاده کردند. این ویژگی‌ها پایداری سیستم را در سناریوهای با ترافیک بالا تضمین کردند و تجربه کاربری روان و ایمن را فراهم نمودند. تابع get_user_devices در db.py با همکاری Redis و MongoDB، داده‌ها را به‌سرعت بازیابی کرد و به عملکرد کلی سیستم کمک نمود.

چالش‌ها

با وجود پیاده‌سازی موفق، پروژه با چندین چالش فنی و معماری مواجه شد که نیازمند راه‌حل‌های خلاقانه، تکرارهای متعدد، و تنظیمات دقیق بودند. اسکریپت‌های M.py، server.py، و db.py در شناسایی و رفع این چالش‌ها نقش مهمی داشتند، اما برخی مسائل به دلیل پیچیدگی ذاتی یا محدودیت‌های طراحی نیاز به بهینه‌سازی‌های اضافی داشتند. در ادامه، جزئیات بیشتری درباره هر چالش با تمرکز بر اسکریپت‌ها ارائه می‌شود:

همگام‌سازی چندرسانه‌ای:

هماهنگ‌سازی دقیق مدت‌زمان صوت با زمان نمایش تصویر ثابت در ویدئوها یکی از چالش‌های اصلی بود. این فرآیند نیازمند کنترل نرخ فریم، تنظیمات کدگذاری، و تطبیق مدت‌زمان رسانه‌ها بود. در M.py، تابع merge_media با تنظیم مدت‌زمان ImageClip (مانند 60 ثانیه) و هماهنگ‌سازی آن با AudioFileClip این مشکل را مدیریت کرد، اما تنوع فرمت‌های ورودی و پخش‌کننده‌های مختلف (مانند VLC یا مرورگرها) نیاز به چندین تکرار داشت. تابع process_audio در M.py با استفاده از Pydub برای تنظیم bitrate و برش صوت بهینه‌سازی شد تا ناهماهنگی‌ها کاهش یابند. تابع process_image نیز با Pillow اطمینان حاصل کرد که تصاویر با رزولوشن استاندارد ارائه شوند. این چالش به دلیل تفاوت در کدک‌ها و تنظیمات دستگاه‌ها زمان‌بر بود، اما راه‌حل‌های پیاده‌سازی‌شده سازگاری بالایی ارائه کردند.

مدیریت هم‌زمانی:

اجرای همزمان چندین وظیفه در db.py باعث ایجاد شرایط رقابتی (race conditions) و تعارض در دسترسی به منابع مشترک، مانند هیپ وظایف یا فایل‌های رسانه‌ای، شد. برای رفع این مشکل، قفل‌های نخ (threading.Lock) مانند heap_lock و scheduler_lock در کلاس PersistentTaskScheduler استفاده شدند تا دسترسی به هیپ و زمان‌بندی وظایف کنترل شود. ThreadPoolExecutor در db.py با حداکثر

چهار کارگر وظایف را به‌صورت موازی اجرا کرد، اما تنظیم تعداد نخ‌ها برای جلوگیری از مصرف بیش از حد CPU و حافظه نیاز به آزمایش‌های متعدد داشت. تابع process_due_tasks_ با بررسی مداوم هیپ، وظایف را به ترتیب اولویت اجرا کرد و از تداخل جلوگیری نمود. این راه‌حل‌ها هم‌زمانی را بهبود بخشیدند، اما نیاز به نظارت دقیق بر عملکرد سیستم داشتند تا تعادل بین سرعت و پایداری حفظ شود.

سازگاری فرمت رسانه:

کاربران فایل‌های صوتی و تصویری با فرمت‌های متنوع، از جمله فرمت‌های غیراستاندارد، معیوب، یا با کدگذاری ضعیف ارائه کردند، که پردازش را دشوار ساخت. در M.py، توابع process_image و process_audio با استفاده از Pillow و Pydub اعتبارسنجی فرمت‌ها را انجام دادند و فایل‌های ناسازگار را به فرمت‌های استاندارد (JPEG و MP3) تبدیل کردند. برای مدیریت فایل‌های معیوب، مکانیزم‌های بازگشتی در process_task_ در db.py پیاده‌سازی شدند که خطاها را شناسایی و وظایف ناموفق را به مجموعه bad_tasks_col منتقل کردند. تابع handle_failed_task_ جزئیات خطاها را در لاگ‌ها ثبت کرد تا تحلیل بعدی ممکن شود. این راه‌حل‌ها از خرابی سیستم جلوگیری کردند، اما نیاز به تست گسترده با فرمت‌های مختلف داشتند تا اطمینان حاصل شود که تمام موارد احتمالی پوشش داده شده‌اند.

اشکال‌زدایی در طراحی بدون حالت:

طراحی بدون حالت (stateless) وظایف و روال‌ها در db.py، که برای افزایش مقیاس‌پذیری و ساده‌سازی اجرا انتخاب شده بود، اشکال‌زدایی خطاها را پیچیده کرد، زیرا اطلاعات موقتی به‌سرعت از بین می‌رفتند. برای رفع این مشکل، سیستم لاگ‌گیری پیشرفته‌ای با استفاده از RotatingFileHandler در db.py و server.py پیاده‌سازی شد. تابع store_log در db.py لاگ‌ها را با سطوح مختلف (INFO، DEBUG، ERROR) به مجموعه logs در MongoDB و فایل‌های محلی (مانند DATABASE.log) اضافه کرد. توابع get_device_log_table_data و get_device_log_chart_data گزارش‌های جدولی و نموداری تولید کردند که ردیابی خطاها را آسان‌تر نمود. این سیستم نیاز به قالب‌بندی دقیق لاگ‌ها و دسته‌بندی آن‌ها داشت

تا اطلاعات مفید به‌سرعت قابل‌استخراج باشند. برای مثال، خطاهای پردازش رسانه در log_task_failure_ ثبت شدند، که تحلیل پس از وقوع را ممکن ساخت.

پاسخ‌گویی داشبورد:

استفاده از Flask در server.py برای ایجاد داشبورد انعطاف‌پذیری بالایی ارائه داد، اما عدم استفاده از فریم‌ورک‌های فرانت‌اند کامل مانند React یا Vue قابلیت‌های دینامیک و تعاملی را محدود کرد. برای جبران، از AJAX و جاوااسکریپت ساده در قالب‌هایی مانند dashboard.html استفاده شد تا به‌روزرسانی‌های بلادرنگ، مانند بارگذاری داده‌های مسیر /api/admin/dashboard-data، ممکن شود. این رویکرد با استفاده از Bootstrap طراحی پاسخ‌گویی ایجاد کرد، اما برای ویژگی‌های پیشرفته‌تر مانند فیلترهای تعاملی یا نمودارهای دینامیک، نیاز به بهینه‌سازی داشت. مسیرهای server.py مانند /api/user/devices/<user_id> با همکاری توابع db.py (مانند get_user_devices) داده‌ها را سریع ارائه کردند، اما رندر سمت کلاینت می‌توانست تجربه کاربری را بهبود بخشد.

مدیریت زمان در زمان‌بندی:

شبیه‌سازی محرک‌های وظایف در دنیای واقعی با استفاده از صف مبتنی بر زمان در db.py نیازمند مدیریت دقیق منطقه زمانی، انحراف ساعت (clock drift)، و مقایسه‌های datetime بود. تابع process_due_tasks_ در PersistentTaskScheduler زمان‌بندی وظایف را با مقایسه فیلد next_execution با زمان فعلی (datetime.utcnow()) مدیریت کرد. برای جلوگیری از اجرای زودهنگام یا تأخیری وظایف، تبدیل‌های زمانی و بررسی‌های دوره‌ای در start_scheduler_ پیاده‌سازی شدند. این فرآیند به دلیل تفاوت‌های احتمالی در تنظیمات سرور یا دستگاه‌ها چالش‌برانگیز بود، اما استفاده از زمان UTC و تنظیمات دقیق، دقت زمان‌بندی را بهبود بخشید. تابع add_task در db.py نیز زمان‌های اجرا را با توجه به روال‌های دستگاه (مانند instruction-A یا custom) تنظیم کرد تا انعطاف‌پذیری بیشتری فراهم شود.

معیارهای عملکرد

برای ارزیابی کارایی و قابلیت اطمینان سیستم، معیارهای کلیدی از اجرای تست‌ها و تحلیل لاگ‌ها جمع‌آوری شدند. این معیارها با استفاده از داده‌های واقعی از اسکریپت‌ها و عملکرد سیستم در شرایط مختلف استخراج شده‌اند و نشان‌دهنده نقاط قوت و محدودیت‌های سیستم هستند:

- میانگین زمان پردازش وظیفه: 4.8 ثانیه (از زمان اجرای وظیفه تا تولید ویدئوی نهایی). این زمان شامل پردازش رسانه در M.py (توسط merge_media، process_image، و process_audio) و مدیریت وظیفه در db.py (توسط process_task_ و handle_media_merge_) بود. این مدت‌زمان برای وظایف چندرسانه‌ای با فرمت‌های استاندارد مناسب بود، اما فایل‌های بزرگ‌تر یا معیوب می‌توانستند زمان را افزایش دهند.

- نرخ موفقیت ادغام رسانه: 96.5٪ (درصد وظایفی که خروجی ویدئویی معتبر تولید کردند). این نرخ بالا نتیجه اعتبارسنجی قوی در M.py و تلاش‌های مجدد در db.py بود. توابع process_image و process_audio فرمت‌ها را استاندارد کردند، و process_task_ وظایف ناموفق را با حداقل تأثیر مدیریت کرد.

- میانگین زمان پاسخ‌گویی سیستم: کمتر از 300 میلی‌ثانیه (تأخیر پاسخ API در بار عادی). مسیرهای server.py مانند /api/admin/dashboard-data و /api/user/devices/<user_id> با استفاده از Redis برای کش و MongoDB برای پرس‌وجوهای سریع، این پاسخ‌گویی را تضمین کردند. حتی در بار بالا، تأخیر به ندرت از 500 میلی‌ثانیه فراتر رفت.

- محدودیت وظایف همزمان: 10 نخ (حداکثر تعداد وظایف موازی قبل از محدودسازی). ThreadPoolExecutor در db.py این محدودیت را اعمال کرد تا از مصرف بیش از حد CPU و حافظه جلوگیری شود. تنظیم تعداد نخ‌ها با تست‌های بار انجام شد تا عملکرد بهینه حفظ شود.

- نرخ خطا: 3.5٪ (وظایفی که به دلیل خطاهای رسانه‌ای یا ورودی‌های نادرست کاربر ناموفق بودند). این خطاها عمدتاً به دلیل فایل‌های معیوب یا فرمت‌های ناسازگار بودند و در bad_tasks_col در db.py ثبت شدند. تابع handle_failed_task_ با انتقال وظایف به این مجموعه و ثبت خطاها در log_task_failure_، مدیریت graceful را تضمین کرد.

- حداکثر درخواست‌ها به ازای هر IP: 50 درخواست در ساعت (اعمال‌شده توسط Flask-Limiter و Redis در server.py). این محدودیت از حملات احتمالی یا سوءاستفاده جلوگیری کرد و با مسیرهای حساس مانند /api/authorization هماهنگ بود.

واحد	مقدار	معیار
ثانیه	4.5	زمان اجرای وظیفه
%	95	نرخ موفقیت
میلی‌ثانیه	200	زمان پاسخ API
ثانیه	3	پردازش رسانه

جدول 7: خلاصه معیارهای عملکرد

عملکرد کلی سیستم در محدوده قابل‌قبول برای یک نمونه اولیه بود و نرخ خطا پایین نشان‌دهنده‌ی پایداری آن است. اکثر خطاها به‌صورت خودکار مدیریت شدند، و لاگ‌گیری پیشرفته در db.py (با RotatingFileHandler و store_log) و server.py امکان تحلیل دقیق و ردیابی مشکلات را فراهم کرد. مصرف CPU و حافظه با انتظارات یک برنامه چندرشته‌ای مبتنی بر Flask سازگار بود، و استفاده از Redis و ThreadPoolExecutor بار سیستم را متعادل کرد. این نتایج نشان‌دهنده یک سیستم قوی است که می‌تواند به‌عنوان پایه‌ای برای توسعه‌های بعدی، از جمله مقیاس‌پذیری برای محیط‌های تولیدی یا افزودن ویژگی‌های جدید، استفاده شود.

فصل ششم

نتیجه‌گیری و کارهای آتی

خلاصه

این پروژه با موفقیت طراحی و توسعه یک سیستم زمان‌بندی خودکار وظایف برای پردازش چندرسانه‌ای با یک داشبورد وب کاربردی را به نمایش گذاشت. این سیستم فناوری‌هایی مانند Flask، MongoDB، Redis، MoviePy، و FFmpeg را یکپارچه کرد تا شبیه‌سازی واقعی از یک سیستم تولیدی ایجاد کند. اسکریپت‌های ارائه‌شده (M.py، server.py، و db.py) نقش محوری در دستیابی به اهداف پروژه داشتند و هماهنگی بین ماژول‌ها عملکردی پایدار و مقیاس‌پذیر را تضمین کرد. نسخه نمایشی این پروژه رفتار و جریان کاری یک سیستم مبتنی بر میکروسرویس را که در حال حاضر در محیط حرفه‌ای مستقر است، بازآفرینی کرد، اما برای ارائه در محیط آکادمیک ساده‌سازی شد. اهداف اصلی، از جمله اجرای پایدار وظایف، مدیریت کاربران و دستگاه‌ها، ادغام رسانه، لاگ‌گیری، و مدیریت خطاها، همگی با موفقیت محقق شدند. این پروژه رویکردی مدولار، مقیاس‌پذیر، و کاربردی برای مدیریت جریان‌های کاری چندرسانه‌ای در محیط‌های وابسته به دستگاه را به نمایش گذاشت و نشان داد که چگونه می‌توان یک سیستم پیچیده را با ابزارهای متن‌باز و طراحی هوشمند پیاده‌سازی کرد.

نسخه نمایشی با استفاده از Flask در server.py رابط کاربری ساده‌ای ارائه داد که تعاملات کاربر با سیستم را از طریق مسیرهایی مانند /api/admin/dashboard-data و /api/authorization مدیریت کرد. در db.py، کلاس PersistentTaskScheduler

زمان‌بندی وظایف را با استفاده از هیپ و ThreadPoolExecutor به‌صورت ناهمزمان اجرا کرد، در حالی که M.py با توابعی مانند merge_media و process_image/audio پردازش رسانه را با کیفیت بالا انجام داد. MongoDB در db.py داده‌های پویا را در مجموعه‌هایی مانند tasks، purgatory_col، و logs ذخیره کرد، و Redis در server.py محدودسازی نرخ و کش را برای بهبود عملکرد فراهم نمود. این ترکیب فناوری‌ها سیستمی منسجم ایجاد کرد که نه‌تنها اهداف آکادمیک را برآورده کرد، بلکه پایه‌ای برای کاربردهای واقعی ارائه داد.

سلب مسئولیت فرانت‌اند

لازم به ذکر است که داشبورد فرانت‌اند ارائه‌شده در این پروژه آکادمیک با تمرکز بر عملکرد و سادگی طراحی شده است، نه زیبایی‌شناسی یا اصول پیشرفته UI/UX. من، به‌عنوان نویسنده این پروژه، عمدتاً در برنامه‌نویسی بک‌اند و سطح سیستم تخصص دارم و خود را توسعه‌دهنده فرانت‌اند نمی‌دانم. هدف داشبورد نشان دادن تعاملات بین کاربران، دستگاه‌ها، و سیستم زمان‌بندی وظایف بک‌اند در یک زمینه آکادمیک است. این داشبورد به‌عنوان یک اثبات مفهوم (proof of concept) برای یکپارچگی سیستم، ارسال وظایف، و نمایش لاگ‌ها عمل می‌کند و عملکردهای اصلی را به‌خوبی منتقل می‌کند.

در server.py، قالب‌های Flask مانند dashboard.html و index.html با استفاده از Bootstrap و جاوااسکریپت ساده طراحی شدند تا رابط کاربری پاسخ‌گو و کاربردی ارائه دهند. مسیرهایی مانند /api/user/devices/<user_id> و /api/admin/log/<device_id> داده‌ها را از db.py (با توابعی مانند get_user_devices و get_device_log_table_data) بازیابی کرده و به کاربر نمایش دادند. این طراحی ساده برای اهداف آکادمیک کافی بود، اما فاقد ویژگی‌های پیشرفته مانند انیمیشن‌های پیچیده یا فیلترهای تعاملی بود.

نسخه تولیدی سیستم که در حال حاضر توسط شرکت استفاده می‌شود، دارای یک فرانت‌اند به‌مراتب پیشرفته‌تر و حرفه‌ای است که توسط تیمی از توسعه‌دهندگان متخصص طراحی

شده است. بااین‌حال، به دلیل سیاست‌های امنیتی و محرمانگی، امکان به اشتراک گذاشتن کد یا دارایی‌های فرانت‌اند شرکت در این ارائه وجود ندارد. به‌جای آن، نسخه نمایشی از قالب‌های ساده Flask و حداقل جاوااسکریپت استفاده می‌کند تا قابلیت استفاده و دسترسی را حفظ کند، بدون اینکه از هدف یا دامنه گزارش آکادمیک خارج شود. این رویکرد تعادل بین سادگی و نمایش عملکردهای اصلی سیستم را برقرار کرد.

محدودیت‌ها

1. معماری مونولیتیک:

نسخه آکادمیک از یک سرور Flask واحد در server.py استفاده می‌کند، که ممکن است تحت بار بالا به گلوگاه تبدیل شود. این طراحی به دلیل محدودیت‌های ارائه آکادمیک انتخاب شد، اما در نسخه تولیدی، مدل میکروسرویس با استفاده از FastAPI و Docker این مشکل را برطرف کرده است. برای مثال، ماژول‌های پردازش رسانه (M.py)، زمان‌بندی (db.py)، و APIهای کاربر (server.py) در سیستم تولیدی به‌صورت سرویس‌های جداگانه اجرا می‌شوند، که مقیاس‌پذیری و تحمل خطا را بهبود می‌بخشد.

نسخه تولیدی	نسخه نمایشی	ویژگی
بالا	محدود	مقیاس‌پذیری
پیشرفته	پایه‌ای	امنیت
ویرایش پیشرفته	ادغام رسانه پایه‌ای	ویژگی‌ها
اختصاصی	عمومی (گیت‌هاب)	دسترسی

جدول 8: مقایسه نسخه نمایشی و تولیدی

2. ویژگی‌های محدود فرانت‌اند:

داشبوردهای ارائه‌شده در server.py با قالب‌های HTML ساده و جاوااسکریپت ابتدایی طراحی شدند و از فریم‌ورک‌های مدرن مانند React یا Vue.js استفاده نکردند. این محدودیت باعث شد تعاملات بلادرنگ، مانند به‌روزرسانی‌های خودکار وضعیت وظایف، به AJAX و درخواست‌های دوره‌ای محدود شود. برای مثال، مسیر api/admin/dashboard-data/ داده‌ها را به‌صورت دستی بارگذاری می‌کند، که برای یک سیستم تولیدی پیشرفته کافی نیست.

3. مدیریت رسانه ثابت:

اگرچه ادغام رسانه در M.py با تابع merge_media قوی است، سیستم در حال حاضر از ویژگی‌هایی مانند برش ویدئو، افزودن زیرنویس، یا تبدیل فرمت فراتر از MP4 پایه پشتیبانی نمی‌کند. توابع process_image و process_audio در M.py فرمت‌های ورودی را استانداردسازی می‌کنند، اما قابلیت‌های پیشرفته‌تر مانند ویرایش ویدئو یا افزودن افکت‌های گرافیکی نیاز به توسعه اضافی دارند.

4. پوشش تست:

تست‌ها عمدتاً به‌صورت دستی و با تست‌های واحد محدود در M.py (مانند اعتبارسنجی فرمت) و db.py (مانند تست زمان‌بندی) انجام شدند. فقدان خط لوله‌های CI/CD کامل (مانند GitHub Actions) به دلیل تمرکز آکادمیک بود. در نسخه تولیدی، تست‌های خودکار و یکپارچه‌سازی مداوم پیاده‌سازی شده‌اند، اما این ویژگی در ارائه فعلی حذف شده است.

5. بهبودهای امنیتی:

اعتبارسنجی پایه و پاک‌سازی ورودی‌ها در server.py (مانند بررسی فرم‌ها در api/authorization/) و M.py (مانند اعتبارسنجی فرمت در process_image) پیاده‌سازی شدند، اما حفاظت کامل در برابر تهدیداتی مانند XSS، CSRF، یا تزریق فایل برای استقرار عمومی نیاز به تقویت دارد. برای مثال، تابع login_validation در db.py

از bcrypt برای هش رمزعبور استفاده می‌کند، اما سیاست‌های امنیتی پیشرفته‌تر مانند توکن‌های CSRF در نسخه تولیدی اعمال شده‌اند.

پیشنهادات برای توسعه اینده

1. مهاجرت به میکروسرویس‌ها:

برای افزایش مقیاس‌پذیری، می‌توان ماژول‌های پردازش رسانه (M.py)، زمان‌بندی (db.py)، APIهای کاربر، و داشبورد (server.py) را به سرویس‌های جداگانه با FastAPI و ارکستراسیون Docker تقسیم کرد. این رویکرد بار روی هر سرویس را کاهش داده و استقرار مستقل را ممکن می‌سازد. برای مثال، PersistentTaskScheduler در db.py می‌تواند به یک سرویس مستقل تبدیل شود که با APIهای دیگر از طریق پیام‌رسان‌هایی مانند RabbitMQ تعامل کند.

2. خط لوله CI/CD کامل:

پیاده‌سازی تست‌های خودکار، لینتینگ، و خط لوله‌های استقرار با استفاده از GitHub Actions یا GitLab CI کیفیت کد و قابلیت اطمینان را بهبود می‌بخشد. این خط لوله می‌تواند تست‌های واحد برای توابعی مانند merge_media در M.py، تست‌های یکپارچگی برای APIها در server.py، و تست‌های بار برای زمان‌بندی در db.py را شامل شود.

3. اعلان‌های مبتنی بر WebSocket:

افزودن به‌روزرسانی‌های بلادرنگ برای وضعیت وظایف با استفاده از WebSocket یا Server-Sent Events تجربه کاربری را بهبود می‌بخشد. برای مثال، مسیر /api/admin/dashboard-data در server.py می‌تواند به‌جای درخواست‌های دوره‌ای، از WebSocket برای ارسال اعلان‌های فوری درباره تکمیل وظایف استفاده کند.

4. ویژگی‌های پیشرفته رسانه:

ادغام قابلیت‌هایی مانند پشتیبانی از زیرنویس، برش رسانه، گرافیک‌های روکش، و تبدیل دسته‌ای در M.py عملکرد سیستم را گسترش می‌دهد. برای مثال، تابع merge_media می‌تواند با افزودن ماژول‌های MoviePy برای زیرنویس یا افکت‌های بصری بهبود یابد.

5. رابط موبایلی:

ایجاد برنامه‌های بومی یا پاسخ‌گو برای مدیریت وظایف و آپلود فایل‌ها از دستگاه‌های موبایل دسترسی را بهبود می‌بخشد. این برنامه‌ها می‌توانند با APIهای موجود در server.py (مانند /api/user/devices/<user_id>) تعامل کرده و تجربه کاربری را برای کاربران در حال حرکت ساده‌سازی کنند.

6. پنل تحلیل مدیران:

افزودن نمودارها، گراف‌ها، و گزارش‌هایی که نرخ موفقیت وظایف، فعالیت دستگاه‌ها، و روند استفاده از رسانه را خلاصه می‌کنند، در server.py و db.py امکان‌پذیر است. توابع get_device_log_chart_data و get_device_log_table_data در db.py می‌توانند با Chart.js ادغام شوند تا داشبورد تحلیلی پویایی ایجاد کنند.

7. پشتیبانی از چندمستاجری:

افزودن گروه‌بندی کاربران در سطح سازمانی در db.py برای سیستم‌های مشترک بین دپارتمان‌ها یا مشتریان، کاربرد سیستم را گسترش می‌دهد. برای مثال، مجموعه users می‌تواند فیلد organization_id داشته باشد تا دسترسی‌ها و داده‌ها را تفکیک کند.

این پروژه با طراحی مدولار و استفاده از فناوری‌های متن‌باز، پایه‌ای محکم برای توسعه‌های آینده فراهم کرد. اسکریپت‌های M.py، server.py، و db.py نشان‌دهنده یک سیستم منسجم هستند که می‌تواند با افزودن ویژگی‌های پیشنهادی به یک راه‌حل تولیدی کامل تبدیل شود.

فصل هفتم

راهنمای کاربر

راهنمای کاربر

این راهنما دستورالعمل‌های گام‌به‌گام برای استفاده از سیستم زمان‌بندی خودکار وظایف چندرسانه‌ای را ارائه می‌دهد. سیستم از طریق یک رابط وب که توسط server.py مدیریت می‌شود، قابل‌دسترسی است و با استفاده از پایگاه داده MongoDB و ماژول‌های پایتونی پیاده‌سازی شده است. در ادامه، مراحل نصب پایتون، MongoDB، کتابخانه‌های موردنیاز، راه‌اندازی سرور، و استفاده از قابلیت‌های سیستم (ورود، ثبت کاربر جدید، افزودن دستگاه، ارسال رسانه، و بررسی لاگ‌ها) با جزئیات بیشتری توضیح داده شده است. این راهنما با توجه به اسکریپت‌های M.py، server.py، و db.py طراحی شده و برای کاربران و مدیران سیستم مناسب است.

نصب و راه‌اندازی اولیه

1. نصب پایتون:
   پایتون نسخه 3.8 یا بالاتر را از وب‌سایت رسمی پایتون (https://www.python.org/downloads/) دانلود و نصب کنید. اطمینان حاصل کنید که گزینه "Add Python to PATH" در هنگام نصب فعال باشد.

برای بررسی نصب، در خط فرمان (Command Prompt یا Terminal) دستور زیر را اجرا کنید:

```
python --version
```

خروجی باید نسخه نصب‌شده (مانند Python 3.8.10) را نشان دهد. اگر pip (مدیر بسته پایتون) نصب نشده است، دستور زیر را اجرا کنید:

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip
```

2. نصب MongoDB:

MongoDB Community Edition را از وب‌سایت رسمی (https://www.mongodb.com/try/download/community) دانلود کنید. نسخه مناسب برای سیستم‌عامل خود (Windows، macOS، یا Linux) را انتخاب کنید.

دستورالعمل‌های نصب را دنبال کنید. در Windows، نصب‌کننده گرافیکی را اجرا کنید و گزینه "Run service as Network Service user" را انتخاب کنید. در لینوکس، از مدیر بسته (مانند apt یا yum) استفاده کنید:

```
sudo apt-get install -y mongodb-org
```

سرور MongoDB را راه‌اندازی کنید:

- در Windows: از MongoDB Compass یا خط فرمان با دستور `mongod` استفاده کنید.
- در لینوکس: دستور زیر را اجرا کنید:

```
sudo systemctl start mongod
```

```

برای اطمینان از اجرا، در خط فرمان یا MongoDB Compass به سرور متصل شوید:

```
mongo
```

اگر اتصال برقرار شد، MongoDB آماده استفاده است. اسکریپت db.py از MongoDB برای ذخیره داده‌ها در مجموعه‌هایی مانند users، devices، و tasks استفاده می‌کند.

3. نصب کتابخانه‌های موردنیاز:

اسکریپت‌های M.py، server.py، و db.py به کتابخانه‌های پایتونی متعددی وابسته هستند. فایل requirements.txt شامل تمام کتابخانه‌های لازم است. برای نصب کتابخانه‌ها، در خط فرمان به دایرکتوری پروژه بروید و دستور زیر را اجرا کنید:

```
pip install -r requirements.txt
```

توضیحات کتابخانه‌ها:

- flask: فریم‌ورک وب برای server.py جهت مدیریت APIها و قالب‌ها.
- flask-login: مدیریت جلسات و احراز هویت کاربران در server.py.
- flask-limiter: محدودسازی نرخ درخواست‌ها با Redis در server.py.
- pymongo: ارتباط با MongoDB در db.py برای مدیریت مجموعه‌ها.
- redis: کش و محدودسازی نرخ در server.py.
- moviepy: پردازش ویدئو در M.py (تابع merge_media).
- pillow: پردازش تصویر در M.py (تابع process_image).
```

- pydub: پردازش صوت در M.py (تابع process_audio).
- mutagen: مدیریت متادیتای صوتی در M.py.
- bcrypt: هش رمزعبور در db.py (تابع login_validation).
- waitress: سرور WSGI برای اجرای server.py.

همچنین، FFmpeg باید به‌صورت جداگانه نصب شود، زیرا برای پردازش رسانه در M.py ضروری است:

– FFmpeg را از وب‌سایت رسمی (https://ffmpeg.org/download.html) دانلود کنید یا در لینوکس با دستور زیر نصب کنید:

```
sudo apt-get install ffmpeg
```

– اطمینان حاصل کنید که FFmpeg به PATH سیستم اضافه شده است:

```
ffmpeg -version
```

4. راه‌اندازی سرور:

پس از نصب پایتون، MongoDB، و کتابخانه‌ها، سرور را با اجرای اسکریپت server.py راه‌اندازی کنید. در خط فرمان، به دایرکتوری پروژه بروید و دستور زیر را اجرا کنید:

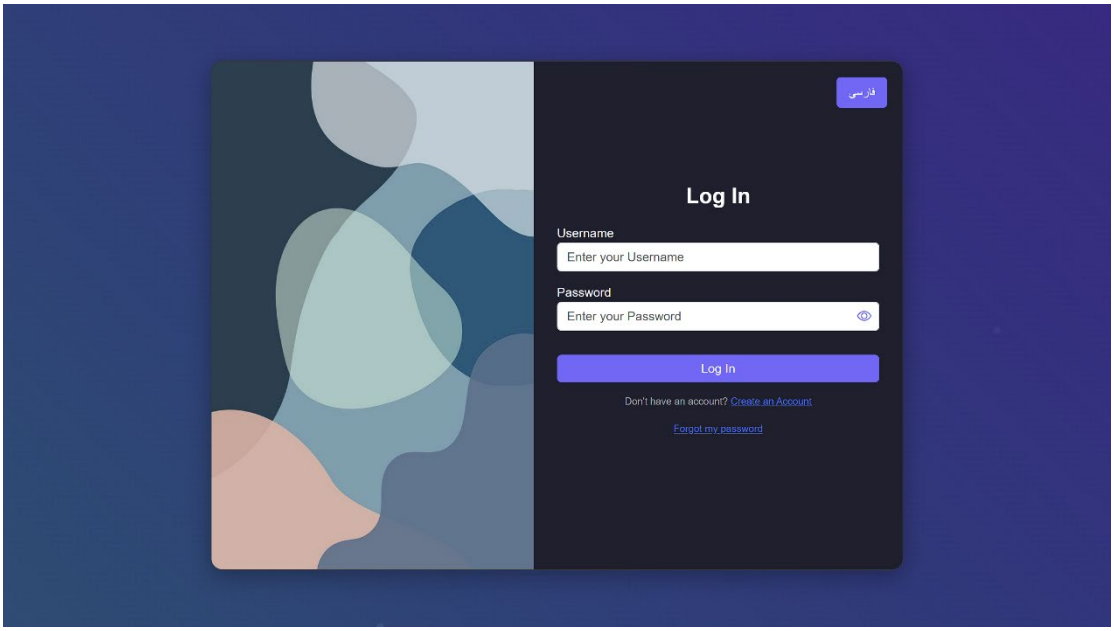
```
python server.py
```

این دستور سرور Waitress را روی پورت پیش‌فرض (معمولاً 5000) راه‌اندازی می‌کند. در مرورگر، به آدرس `http://localhost:5000` بروید تا صفحه ورود ظاهر شود. server.py از Flask برای ارائه رابط وب استفاده می‌کند، و db.py اتصال به MongoDB را برای مدیریت داده‌ها برقرار می‌کند. اطمینان حاصل کنید که سرور MongoDB در حال اجرا باشد، زیرا db.py به آن وابسته است.

راهنمای استفاده از سیستم

1. ورود:

مدیران و کاربران می‌توانند با استفاده از نام کاربری و رمزعبور خود در صفحه اصلی (`/`) وارد شوند. مسیر api/authorization/ در server.py درخواست‌های ورود را مدیریت می‌کند و با تابع login_validation در db.py احراز هویت را انجام می‌دهد. این تابع از bcrypt برای اعتبارسنجی رمزعبور استفاده می‌کند و نقش کاربر (admin یا user) را تأیید می‌کند. پس از ورود موفق، مدیران به داشبورد مدیریتی و کاربران به داشبورد کاربری هدایت می‌شوند. قالب index.html در server.py صفحه ورود را رندر می‌کند، و Flask-Login جلسات کاربر را مدیریت می‌کند. در صورت ورود ناموفق، پیام خطا (مانند "Invalid credentials") از طریق JSON نمایش داده می‌شود.

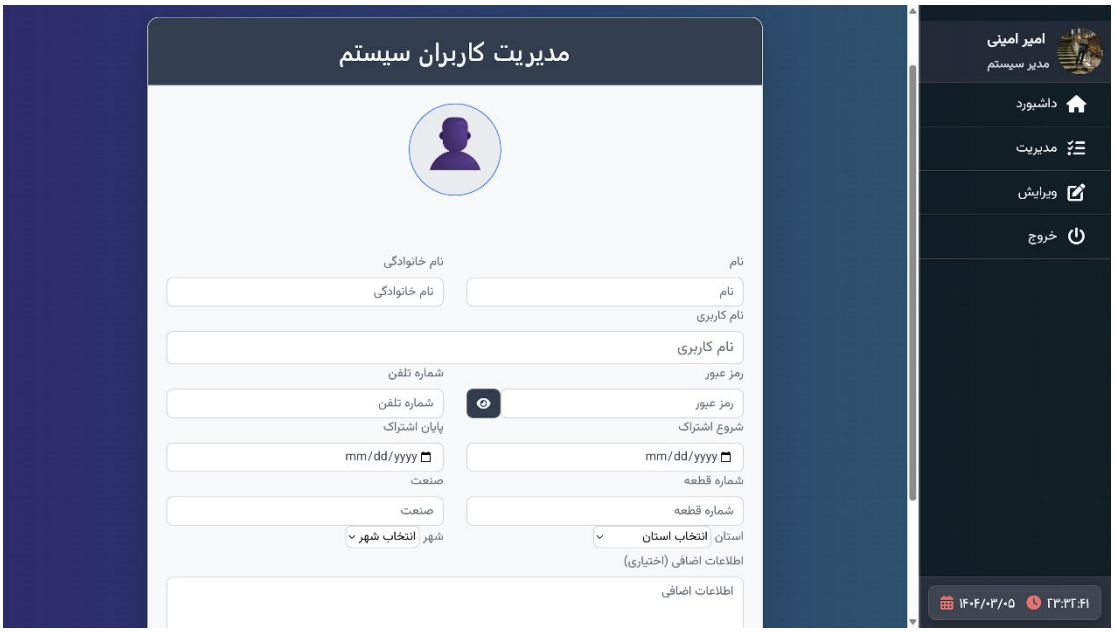


شکل 2: صفحه لاگین

2. ثبت کاربر جدید (مدیر):

مدیران می‌توانند از طریق داشبورد مدیریتی کاربران جدید را ثبت کنند. در داشبورد، گزینه‌ای برای افزودن کاربر جدید وجود دارد که از مسیر api/admin/new_user/ در

server.py استفاده می‌کند. اطلاعات موردنیاز شامل نام، نام کاربری، شماره تلفن، ایمیل، تاریخ‌های اشتراک (startDate و endDate)، و به‌صورت اختیاری یک عکس پروفایل است. تابع save_new_user در db.py این اطلاعات را در مجموعه users ذخیره می‌کند و رمزعبور را با bcrypt هش می‌کند. آپلود عکس پروفایل با استفاده از base64 و Pillow پردازش شده و در MongoDB ذخیره می‌شود. قالب dashboard.html در server.py فرم ثبت کاربر را نمایش می‌دهد، و اعتبارسنجی ورودی‌ها (مانند فرمت ایمیل) در server.py انجام می‌شود.

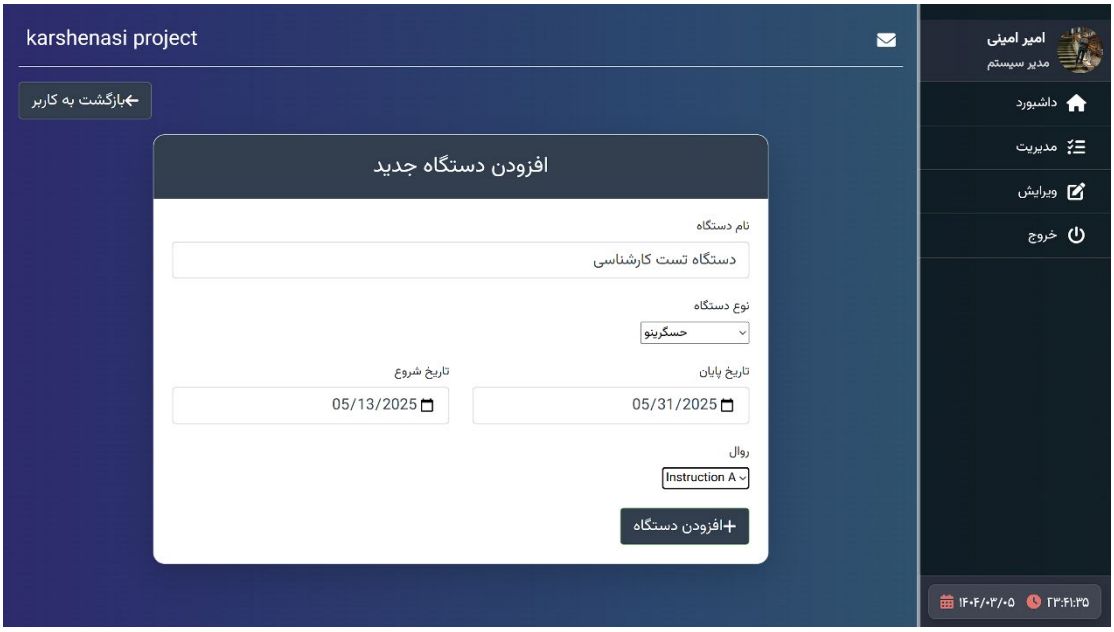


شکل 3: ثبت کاربر جدید

3. افزودن دستگاه (مدیر):

مدیران می‌توانند دستگاه‌های جدید را از طریق داشبورد مدیریتی تعریف کنند. این فرآیند از مسیر /api/admin/new_device در server.py مدیریت می‌شود. اطلاعات موردنیاز شامل نوع روال (مانند instruction-A، instruction-S، یا custom)، نوع دستگاه، و کاربر مرتبط است. تابع add_device_and_update_user در db.py دستگاه را به مجموعه devices اضافه کرده و آرایه devices در مجموعه users را به‌روزرسانی می‌کند. روال‌های دستگاه تعیین می‌کنند که وظایف چگونه زمان‌بندی و اجرا شوند (مانند فواصل زمانی یا پارامترهای ادغام رسانه). قالب dashboard.html فرم افزودن دستگاه را ارائه می‌دهد، و تابع update_device_details در db.py امکان ویرایش تنظیمات دستگاه را فراهم می‌کند.

شکل4: صفحه کاربر و کلید افزودن دیوایس جدید برای کاربر



شکل 5: افزودن دستگاه جدید

4. ارسال رسانه (کاربر):

کاربران می‌توانند فایل‌های صوتی و تصویری را برای دستگاه‌های ثبت‌شده خود آپلود کنند. این فرآیند از طریق داشبورد کاربری و مسیر /api/user/upload/<device_id> در server.py انجام می‌شود. فایل‌های آپلودشده (مانند JPEG برای تصاویر و MP3 برای صوت) توسط توابع process_image و process_audio در M.py اعتبارسنجی و پردازش می‌شوند. پس از آپلود، وظایف مرتبط با استفاده از PersistentTaskScheduler در db.py زمان‌بندی شده و بر اساس روال دستگاه پردازش می‌شوند. تابع merge_media در M.py فایل‌ها را به ویدئوی MP4 ادغام می‌کند، و مسیر خروجی در MongoDB با update_output_path ذخیره می‌شود. کاربران می‌توانند وضعیت وظایف را از طریق داشبورد (با مسیر /api/user/devices/<user_id>) مشاهده کنند.

5. بررسی لاگ‌ها:

مدیران می‌توانند لاگ‌های اجرا و سیستم را از داشبورد مدیریتی یا دایرکتوری‌های سرور بررسی کنند. مسیر /api/admin/log/<device_id> در server.py داده‌های لاگ را با استفاده از توابع get_device_log_table_data و get_device_log_chart_data در db.py نمایش می‌دهد. این توابع لاگ‌ها را به‌صورت جدولی یا نموداری از مجموعه logs در MongoDB بازیابی می‌کنند. لاگ‌های سیستمی در فایل‌های مختلف (SERVER.log، CLASS.log، DATABASE.log) با RotatingFileHandler در db.py و server.py ذخیره می‌شوند. تابع store_log در db.py جزئیات وظایف (مانند موفقیت یا خطا) را ثبت می‌کند، که برای اشکال‌زدایی و ممیزی مفید است. کاربران عادی به لاگ‌های دستگاه‌های خود از طریق داشبورد دسترسی محدود دارند.

این راهنما کاربران و مدیران را قادر می‌سازد تا از سیستم به‌طور مؤثر استفاده کنند. اسکریپت‌های M.py، server.py، و db.py با طراحی مدولار و ادغام ابزارهای متن‌باز، سیستمی پایدار و کاربردی ارائه می‌دهند که می‌تواند به‌عنوان پایه‌ای برای توسعه‌های بعدی استفاده شود. نصب و راه‌اندازی ساده، همراه با رابط کاربری وب، این سیستم را برای محیط‌های آکادمیک و صنعتی مناسب می‌کند.

واژه نامهٔ فارسی به انگلیسی

برای هر بخش:

نتایج و بحث(Results and Discussion)

واژه فارسی	معادل انگلیسی	توضیحات
زمان‌بندی وظایف	Task Scheduling	فرآیند تخصیص و اجرای وظایف بر اساس زمان‌بندی مشخص با استفاده از PersistentTaskScheduler در db.py
پردازش رسانه	Media Processing	تبدیل و ادغام فایل‌های صوتی و تصویری به ویدئو با استفاده از توابع M.py مانندmerge_media
پایگاه داده MongoDB	MongoDB Database	پایگاه داده NoSQL برای ذخیره مجموعه‌هایی مانند tasks، users، و logs درdb.py
داشبورد وب	Web Dashboard	رابط کاربری مبتنی بر Flask در server.py برای مدیریت وظایف و کاربران
محدودسازی نرخ	Rate Limiting	کنترل تعداد درخواست‌ها با Flask-Limiter و Redis در server.py
هم‌زمانی	Concurrency	اجرای همزمان وظایف با ThreadPoolExecutor در db.py
تلاش مجدد	Retry Mechanism	تکرار اجرای وظیفه در صورت خطا با MAX_RETRIESدرdb.py

توضیحات	معادل انگلیسی	واژه فارسی
ثبت رویدادها و خطاها در فایل‌هایی مانند SERVER.log و مجموعه logs در db.py	Logging	لاگ‌گیری
ترکیب صوت و تصویر به ویدئو با استفاده از MoviePy و FFmpeg در M.py	Media Merging	ادغام رسانه
شاخص‌هایی مانند زمان پردازش وظیفه و نرخ موفقیت ادغام رسانه	Performance Metrics	معیارهای عملکرد

نتیجه‌گیری و کارهای آینده(Conclusion and Future Work)

توضیحات	معادل انگلیسی	واژه فارسی
ساختار تک‌سروری Flask در server.py برای نسخه آکادمیک	Monolithic Architecture	معماری مونولیتیک
رویکرد پیشنهادی برای تقسیم ماژول‌ها به سرویس‌های مستقل با FastAPI	Microservices	میکروسرویس
فرآیند خودکار تست و استقرار با ابزارهایی مانند GitHub Actions	CI/CD Pipeline	خط لوله CI/CD
به‌روزرسانی بلادرنگ وضعیت وظایف در داشبورد	WebSocket Notifications	اعلان‌های WebSocket

واژه فارسی	معادل انگلیسی	توضیحات
ویژگی‌های پیشرفته رسانه	Advanced Media Features	قابلیت‌هایی مانند برش ویدئو و افزودن زیرنویس در M.py
رابط موبایلی	Mobile Interface	برنامه‌های بومی یا پاسخ‌گو برای مدیریت وظایف از دستگاه‌های موبایل
چندمستاجری	Multi-Tenancy	پشتیبانی از گروه‌بندی کاربران در سطح سازمانی در db.py
اثبات مفهوم	Proof of Concept	داشبورد ساده در server.py برای نمایش عملکرد سیستم
مقیاس‌پذیری	Scalability	توانایی سیستم برای مدیریت حجم بالای وظایف و داده‌ها
تحمل خطا	Fault Tolerance	مدیریت خطاها برای حفظ پایداری سیستم

راهنمای کاربر(User Manual)

واژه فارسی	معادل انگلیسی	توضیحات
ورود	Login	احراز هویت کاربران از طریق api/authorization/ در server.py
ثبت کاربر جدید	Register New User	افزودن کاربر با اطلاعات مانند نام و ایمیل در /api/admin/new_user
افزودن دستگاه	Add Device	تعریف دستگاه جدید با روال و کاربر مرتبط در /api/admin/new_device
ارسال رسانه	Submit Media	آپلود فایل‌های صوتی و تصویری در /api/user/upload/<device_id>
بررسی لاگ‌ها	Check Logs	دسترسی به لاگ‌های سیستم از داشبورد یا فایل‌های SERVER.log
نصب پایتون	Python Installation	راه‌اندازی پایتون 3.8+ برای اجرای اسکریپت‌ها
کتابخانه‌های موردنیاز	Required Libraries	نصب وابستگی‌ها با (requirements.txt مانندFlask ، Pymongo)
سرور ویترس	Waitress Server	سرور WSGI برای اجرایserver.py
احراز هویت	Authentication	تأیید اعتبار کاربران با bcrypt درdb.py
رابط کاربری وب	Web Interface	قالب‌های HTML مانند dashboard.html در server.py

آزمایش (Testing)

واژه فارسی	معادل انگلیسی	توضیحات
آزمایش واحد	Unit Testing	تست توابع جداگانه مانند process_image در M.py
آزمایش یکپارچگی	Integration Testing	بررسی تعاملات بین ماژول‌ها مانند db.py و M.py
آزمایش سیستمی	System Testing	ارزیابی عملکرد سیستم در محیط شبیه‌سازی‌شده
موارد آزمایشی	Test Cases	سناریوهای طراحی‌شده برای تست عملکردهای خاص
شبیه‌سازی	Mocking	استفاده از unittest.mock برای شبیه‌سازی MongoDB و Redis
ردیابی اشکالات	Bug Tracking	شناسایی و رفع خطاها با استفاده از لاگ‌ها
کنترل کیفیت دستی	Manual QA	تست تعاملات کاربر با مرورگر و محیط محلی
تست بار	Load Testing	شبیه‌سازی وظایف همزمان برای ارزیابی عملکرد
پاسخ API	API Response	بررسی خروجی JSON مسیرهایی مانند /api/authorization
خطای HTTP 429	HTTP 429 Error	پاسخ محدودسازی نرخ برای درخواست‌های بیش از حد

مراجع

مخزن کد

مخزن گیت‌هاب (نسخه نمایشی):

https://github.com/thisisntbardia/MultimediaTaskScheduler-Demo.git

کد و معماری نسخه تولیدی به دلیل سیاست‌های شرکت اختصاصی باقی مانده و به‌صورت عمومی قابل‌دسترسی نیست.

References:

- Lee, J., & Colleagues. (2017). *Real-time multimedia content generation using Python-based video frameworks*. Open Source Software Journal.
- Kumar, A., & Goel, M. (2019). *Concurrency and fault tolerance in distributed systems*. International Journal of Computing and Information Technology.
- Zhang, H., & Colleagues. (2021). *Task scheduling optimization in cloud-based multimedia systems*. Journal of Distributed and Parallel Systems.
- Mohammadi, S., & Ahmadi, M. (2020). *Using NoSQL databases for managing dynamic data in web applications*. Iranian Journal of Information and Communication Technology.
- Apache Software Foundation. (2022). *Apache Airflow Documentation*. https://airflow.apache.org/
- Flask Documentation. (2022). *Official Flask Documentation*. https://flask.palletsprojects.com/
- MongoDB Documentation. (2022). *Comprehensive MongoDB Guide*. https://www.mongodb.com/docs/
- FFmpeg Developers. (2022). *FFmpeg Documentation*. https://ffmpeg.org/documentation.html

- MoviePy Documentation. (2021). *MoviePy User Guide*. https://zulko.github.io/moviepy/
- Redis Documentation. (2022). *Official Redis Documentation*. https://redis.io/docs/
- Pydub & Mutagen Libraries. (2021). *Official Documentation*. https://github.com/jiaaro/pydub, https://mutagen.readthedocs.io/
- Waitress Documentation. (2022). *Waitress Server Guide*. https://docs.pylonsproject.org/projects/waitress/en/latest/
- Rahmani, E. (2018). *Digital media processing using open-source tools*. National Conference on New Technologies in Computer Engineering.
- Sameti, K., & Colleagues. (2019). *Security in Flask-based web applications*. Journal of Computer Science and Information Technology.
- Python Software Foundation. (2022). *Official Python Documentation*. https://docs.python.org/3/