
بسم الله الرحمن الرحيم

دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

صنعت حمل و نقل درون شهری

پروژه درس معماری نرم‌افزار

سید سجاد میرزابابایی ـ امیرحسین کارگران خوزانی
(۹۹۲۰۱۱۱۹ـ۹۹۲۱۰۱۴۲)

استاد درس

دکتر جعفر حبیبی

تحت نظارت

مهدی لشکری

نیمسال دوم تحصیلی ۱۴۰۰ـ۱۳۹۹

فهرست مطالب

عنوان **صفحه**

فصل سوم : قابلیت همکاری

فصل چهارم : اصلاح‌پذیری

فصل پنجم : کارایی

فصل ششم : امنیت

فصل هفتم : قابلیت‌آزمون

چکیده

در این گزارش به بررسی حوزه صنعت حمل و نقل درون شهری از دیدگاه تاکسی‌رانی آنلاین، اشتراک‌گذاری خودرو و پارکینگ‌های آنلاین پرداخته شده است. در گام اول به شرح این حوزه پژوهشی به همراه بررسی نمونه کار های بین المللی پرداخته شده است. در این بررسی نیازمندی‌های عملکردی هر یک از نمونه کارها به صورت مجزا بیان شد و سپس نیازمندی‌های غیرعملکردی به صورت مفصل برای کل حوزه حمل و نقل درون شهری و هر یک از نمونه‌کارها به طور اختصاصی مورد بررسی قرار گرفت. سپس در فصل‌های بعدی معماری هر یک از زیر حوزه‌های تاکسی‌رانی آنلاین، اشتراک‌گذاری خودرو و پارکینگ‌های آنلاین به صورت مجزا تجزیه و تحلیل شده است و در گام آخر برای هر یک از آن‌ها ارزیابی معماری نمونه‌کارها انجام شده است.

کلمات کلیدی: حمل‌ونقل درون‌شهری، تاکسی‌های برخط، پلترفرم های اشتراک خودرو، زیرساخت پارکینگ‌های عمومی

فصل اول

مقدمه

در این فصل به معرفی صنعت حمل و نقل درون‌شهری پرداخته می‌شود و ۳ مطالعه موردی از آن با جزئیات بیشتری مورد بررسی قرار خواهند گرفت. در ادامه در فصل‌های بعد به بررسی نیازهای غیر عملکردی این ۳ مطالعه موردی پرداخته می‌شود. ترتیب و محتوای ارائه این نیازهای غیر عملکردی بر اساس کتاب [۴۸] انتخاب شده است.

۱ـ۱ معرفی صنعت حمل و نقل درون شهری

برای چندین دهه ، برنامه ریزی و حمل و نقل شهری در اطراف خودرو شخصی تکامل یافته است که منجر به مشکلاتی از جمله ازدحام ، صدا ، آلودگی و غیره شده است. همانطور که وارد دهه جدیدی می شویم، تحول دیجیتال همچنان باعث رشد صنعت می شود. معرفی فن آوری های نوظهور و فرآیندهای تجاری، شیوه لجستیک[1] و حمل و نقل را تغییر ساختار داده است و این روند احتمالاً در سال های آتی نیز ادامه خواهد داشت ، خصوصاً که نوآوری در فن آوری منجر به رشد پایدار خواهد شد.

صنعت حمل و نقل امروزه همانند زنجیر صنایع مختلف را بهم متصل می کند؛شرکت ها و سازمان های

[1] Logestic

بزرگی بر پایه ی پیشرفت های اخیر صنعت حمل‌ونقل فرصت ظهور و خودنمایی پیدا کرده اند و سازمان های بزرگی نظیر Uber در آمریکا، cab OLA در هند و DiDi در چین فراتر از مرز های ملی ظاهر شده و در سراسر جهان شروع به ارائه ی خدمات نموده اند.

در این فصل ابتدا به تاکسی‌رانی آنلاین و سهم آن از حمل‌ونقل عمومی از کل صنعت حمل‌ونقل عمومی درون شهری خواهیم پرداخت؛سپس روش های اشتراک خودرو[1] که به تازگی در حال پیدا کردن جایگاه خود در صنعت حمل‌ونقل هستند را مورد بررسی قرار خواهیم داد و در پایان این فصل پیرامون یکی از محیط ترین زیرساخت های شهری یعنی پارکینگ های عمومی و ارتباط آن ها به عنوان یک زیرساخت شهری با صنعت حمل‌ونقل عمومی درون شهری صحبت خواهیم کرد.

۱ـ۲ تاکسی‌رانی و حمل‌ونقل درون شهری

تاکسی‌رانی برخط [2] امروزه در بسیاری از کشور ها سهم زیادی از بازار را در اختیار خود گرفته اند؛برای مثال Uber تا کنون فعالیت خود را به ۷۰ کشور گسترش داده است و میلیون ها راننده برای این غول بزرگ حمل‌ونقل درون شهری فعالیت می کنند و روزانه میلیون ها سفر درون شهری بر بستر این شرکت انجام می شود.

شاید استفاده ی راحت مهم ترین نیازی باشد که شرکت هایی نظیر Uber به آن پاسخ داده اند؛تنها کافی است برنامه ای را بر روی تلفن همراه خود نصب داشته باشید و در کمتر از چند دقیقه نزدیک ترین راننده در حال حرکت به سمت شما برای خدمت رسانی خواهد بود.همچنین راه حل هایی که این شرکت ها در زمینه ی پرداخت هزینه ی سفر،تضمین امنیت سفر و کاهش هزینه های سفر ارائه داده اند بر استقبال هر چه بیشتر جامعه از این پلتفرم [3] های تاکسی برخط[4] افزوده است.

در کنار Uber ، از برنامه های جهانی تاکسی‌رانی برخط می توان به DiDi که عمده فعالیت آن در چین است و هچنین OLA cab که در هند فعالیت می کند اشاره کرد.

یکی از موارد مطالعه در این پژوهش، پروژه MyWay است.MyWay یک پلتفرم مدیریت منابع در صنعت حمل‌ونقل هوشمند در شهر های اروپایی است.هدف این پروژه این است که به طور یکپارچه به ادغام کارآمد و منسجم سرویسهای حمل‌ونقل مکمل، در کل زنجیره سفر های درون شهری بپردازد. MyWay یک اپ عمومی در حوزه حمل و نقل است تلاش کرده که چندین ویژگی در رابطه تاکسی‌رانی آنلاین، اشتراک‌گذاری خودرو یا استفاده از اتوبوس را در کنار هم برای برنامه‌ریزی سفر استفاده کند که بیشتر در این گزارش ما بر روی قسمت

[1] CarPool
[2] Ride sharing
[3] Platform
[4] Online

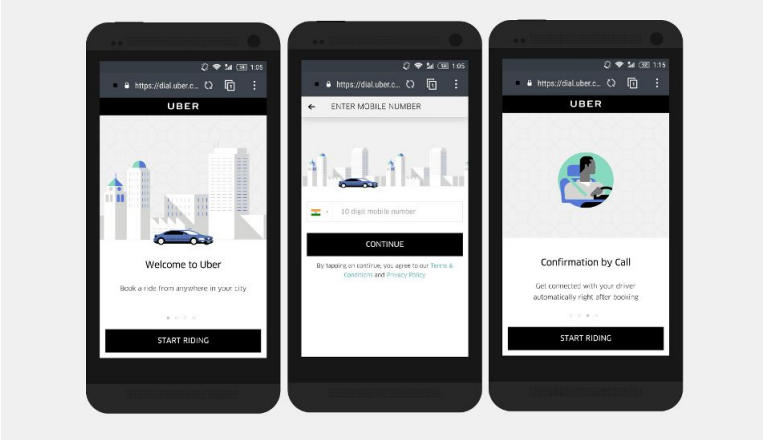
تاکسی‌رانی آنلاین آن تمرکز می‌کنیم.

در پایان این بخش لازم است به این نکته اشاره شود که با ورود شرکت های بزرگ تاکسی‌رانی آنلاین به شهر ها، کسب‌وکار های حمل‌ونقل سنتی آسیب می بینند و همچنین استفاده از وسایل حمل‌ونقل عمومی نیز کاهش خواهد یافت که آثار مخرب زیست محیطی و اجتماعی از جمله معایب گسترش این سیستم هاست.

۱ـ۲ـ۱ نیازهای عملکردی Uber

مهمترین نیازهای عملکردی در اپ Uber را می‌توان به شرح زیر داشت:

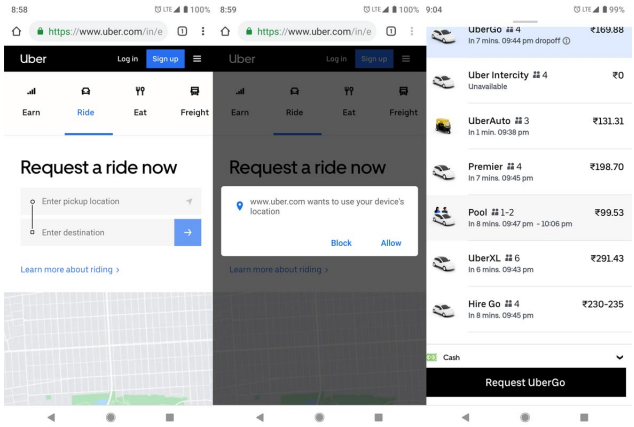
- کاربران باید بتوانند با نام و شماره تلفن و زبان مورد نظر ثبت نام کنند در شکل ۱ـ۱ روند این ثبت نام نشان داده شده است که یک SMS برای تایید به شماره تلفنفرستاده می‌شود و سپس نحوه پرداخت پیشنهادی انتخاب می‌شود.



شکل ۱ـ۱: ثبت نام در اوبر، عکس از themobileindian.com

- قابلیت رزرو یک سرویس تاکسی: مبدا و مقصد را وارد کرده و به کاربر انواع سرویس‌ها و مبلغ آن‌ها را پیشنهاد می‌دهد، این موارد در شکل ۱ـ۲ نمایش داده شده است. رانندگان نزدیک‌تر نیز اطلاعات سفر کاربران را دریافت می‌کنند و آن را قبول یا رد می‌کنند.

- دسترسی به موقعیت جغرافیایی و پیدا کردن آن بر روی نقشه در تمامی دستگاه‌های پشتیبانی شده
- نشان دادن نقشه مسیر هم به کاربر و هم به راننده و راهنماییی گام به گام
- فرستادن پوش‌نوتیفیکیشن مواقعی که رارننده ای درخواست را اکسپت می‌کند یا به نزدیکی محل می‌رسد یا به هر دلیلی سفر لغو می‌شود.



شکل ۱ـ۲: رزرو تاکسی در اوبر، عکس از androidinfotech.com

- محاسبه قیمت دقیق بر اساس مبدا و مقصد و ساعت حرکت و نوع ماشین
- قابلیت اشتراک سفر و مسیر خود با دیگران و دنبال کردن در زمان واقعی
- قابلیت اضافه کردن نقاط توقف در مسیر
- استفاده از سرویس‌های متنوع پرداخت نظیر paypal ، card credit و ...
- قابلیت برنامه‌ریزی سفر تا ۳۰ روز مانده
- امکان همگام‌سازی تقویم قرارهای ملاقات با برنامه

۱ـ۲ـ۲ نیازهای عملکردی MyWay

طبق مستندات [۲۳]برای MyWay می‌توان ۱۰۰ نیازمندی عملکردی بیان کرد که این نیازمندی‌ها را می‌توان در ۴ دسته کلی زیر طبقه بندی کرد

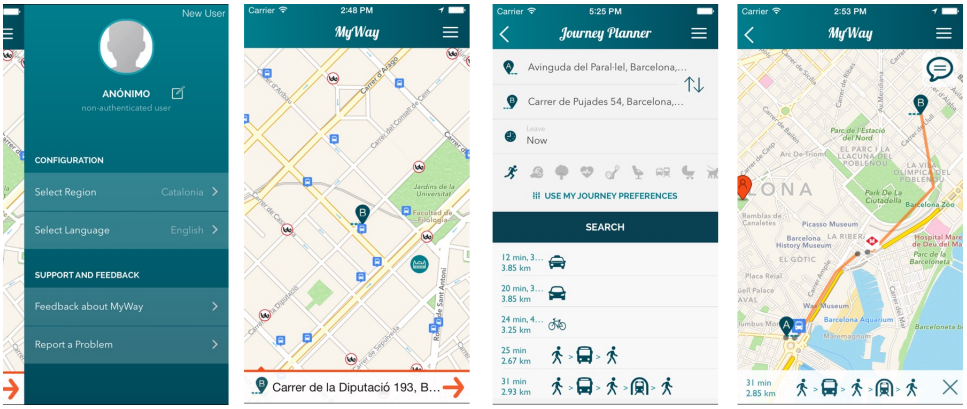
- برنامه‌ریزی سفر و دنبال کردن [1]
- خدمات تحرک[2] و آگاهی از منابع
- نمایه کردن [3] کاربران
- مدیریت سیستم

از مهمترین نیازمندی‌های این برنامه نیز می‌توان به موارد زیر اشاره کرد:

[1] Tracking
[2] Mobility
[3] Profiling

- برنامه‌ریزی سفر
- رزرو یک سرویس
- اطلاعات شهر
- نمایه کردن کاربر

در شکل ۱ـ۳ تعدادی از صفحات نسخه موبایل نمایش داده شده است. از آنجا که نوشتن ریز جزئیات



شکل ۱ـ۳: نمایی از نسخه موبایل اپ Myway تصویر از appadvice.com

نیازمندی‌های عملکردی هدف این گزارش نیست بنابراین ۴ دسته کلی نیازهای عملکردی را به ۱۴ دسته تقسیم می‌کنیم و آن ۱۴ دسته را همراه با توضیحات گزارش می‌کنیم.

برنامه‌ریزی سفر و دنبال کردن:

- برنامه‌ریزی سفر: نیاز است که کاربر با استفاده از موقعیت مکانی خود، آدرس‌ها، انتخاب نقطه بر روی نقشه، نقاط مورد علاقه یک سفر را برنامه‌ریزی کند و برنامه نیز مسیری بین مبدا و مقصد پیشنهاد دهد و اطلاعات ترافیک را بازتاب دهد و آن سفر را به پروفایل کاربر اضافه کند و ...
- رزرو کردن یک سرویس: رزو کردن تاکسی، قطار، اتوبوس، اسکوتر، جای پارک
- دنبال کردن یک سفر: دوباره فیدبک‌دهی و محاسبه به کاربر در رابطه با سفر با استفاده از دنبال کردن آن با GPS
- فیدبک‌دهی آنلاین از شرایط جاده و محدودیت‌ها
- فیدبک دادن به سفر: کاربر می‌تواند فیدبک خود را ارائه دهد.

خدمات تحرک و آگاهی از منابع:

- اطلاعات شهر: ذخیره نقاط مورد علاقه در شهر و اطلاعات دادن در مورد رستوران‌ها و ارزیابی‌آن‌ها و سرویس‌های زمان‌بندی شده تحرک و اطلاعات در مورد توزیع آن‌ها و ...
- فیدبک دادن به Myway در مورد سرویس‌ها

نمایه کردن کاربر:

- نمایه کاربر: اضافه کردن اطلاعات عمومی در نمایه کارب رو تغییر آن‌ها، علایق و ترجیحات و ...
- گروه کردن کاربران: بیشتر برای اشتراک‌گذاری خودرو استفاده می‌شود و کاربر می‌تواند یک جامعه اشتراک‌گذاری خودرو درست کند و به آن اضافه و حذف شود و با افراد آن جامعه برنامه سفر خود را به اشتراک بگذارد.

مدیریت سیستم:

- تحلیل: جمع‌اوری تاریخچه‌ها و انجام عملیات‌های آماری و تحلیل داده‌ها
- احراز هویت کاربر: تعیین نقش و احراز هویت کاربر با استفاده از سیستم احراز هویت
- مدیریت داده: مدیریت شهرها و تنظیمات و تغییر داده ها توسط ادمین‌های Myway
- درگاه برای اپراتورهای سرویس‌ها

۱ـ۳ اشتراک خودرو و حمل‌ونقل درون شهری

در سال های اخیر دغدغه های اقتصادی،زیست محیطی و اجتماعی جوامع را به سمت استفاده مشترک از منابع در دسترس سوق داده است؛صنعت حمل‌ونقل نیز از این قاعده مستثنا نبوده و در جهت استفاده مشترک از زیرساخت های موجود گام‌های موثری را برداشته است.

اشتراک خودرو[1] به معنی به اشتراک گذاشتن خودرو توسط افراد در سفر هایی با مقاصد نزدیک به هم است و در نتیجه سبب می شود تا افراد کمتری برای رفتن به مقاصد خود از خودرو های شخصی استفاده کنند.

با استفاده بیشتر از یک فرد از وسیله نقلیه، هزینه های سفر هر فرد نظیر هزینه های سوخت، عوارض رانندگی و استرس رانندگی کاهش خواهد یافت.

رانندگان و مسافران سفرها را از طریق یکی از چندین رسانه موجود ارائه می دهند و جستجو می کنند. آنها پس از یافتن افراد با سفر های مشابه با یکدیگر تماس می گیرند تا جزئیات سفر را ترتیب دهند. هزینه ها،نقاط

[1] Car Pooling

ملاقات و سایر جزئیات مانند فضای وسایل اضافه همراه افراد نیز از پیش توافق شده است.سپس افراد طبق برنامه ریزی از پیش انجام شده با یکدیگر ملاقات نموده و سفر را انجام می دهند.

هم اکنون استارت آپ های زیادی در زمینه ی اشتراک گذاری خودرو در سرتاسر دنیا شروع به فعالیت کرده اند که می توان برای مثال از Waze Car ، BlaBla Car و Ride Connect نام برد.

از آنجا که استفاده از اشتراک خودرو تعداد خودروهای مورد نیاز مسافران را کاهش می دهد ، اغلب با مزایای متعددی در جامعه همراه است از جمله:

- کاهش در مصرف انرژی و انتشار گازهای گلخانه ای
- کاهش ترافیک سطح شهری
- کاهش تقاضای زیرساخت پارکینگ

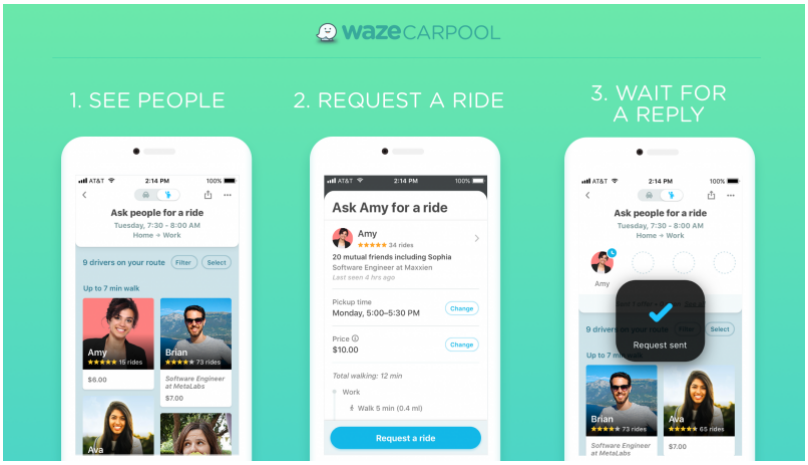
۱ـ۳ـ۱ نیازهای عملکردی Waze Carpool

اشتراک‌گذاری خودرو به دو صورت در شرکت waze انجام می‌شود. یک آن که در خود اپلکیشن اصلی waze یک گزینه جدا برای اشتراک‌گذاری خودرو وجود دارد. دو، برنامه مجزا این شرکت برای اشتراک‌گذاری خودرو است [۴۴] .

هر یک از افراد می‌توانند در نقش راننده یا مسافر ظاهر شوند. برای مسافر تنها کافی است زمان اشتراک‌گذاری خودرو را تنظیم کنند، مشاهده کنند که چه کسی در آن زمان رانندگی می‌کند و از او درخواست عضویت کنند و سپس با یکدیگر تایید کنند. برای رانندگی نیز کافیست که زمان، تعداد صندلی‌ها و مسیر خود را مشخص کنید از مسافران دعوت کنید و اشتراک‌گذاری خودرو را شروع کنید. نحوه درخواست از راننده برای برنامه موبایل در تصویر ۱ـ۴ نمایش داده شده است.

از مهمترین نیازمندی‌های این برنامه می‌توان به موارد زیر اشاره کرد:

- ثبت نام و ورود به برنامه
- وارد کردن اطلاعات در نمایه و تغییر انها
- تنظیم زمان اشتراک گذاری خودرو و مسیر برای راننده و مسافر
- درخواست به رانندگان
- مشاهده درخواست مسافران و قبول و رد انها



شکل ۱ـ۴: درخواست خودرو برای هم‌اشتراک‌گذاری، عکس از sourceWaze

- چت و گرفتن تایید بین راننده و مسافر
- پرداخت با روش‌های مختلف
- تایید رانندگان با سرویس‌های ثالث

۱ـ۴ پارکنیگ های عمومی و حمل‌ونقل درون شهری

برای افرادی که در شهر های بزرگ زندگی می کنند یافتن پارکینگ مقرون به صرفه می تواند یک چالش باشد؛ طی تحقیق که توسط IBM انجام شد محققان به این واقعیت پی ردند که نزدیک به ۳۰٪ از ترافیك شهر به رانندگانی که بدنبال پارکینگ هستند، نسبت داده می شود. خوشبختانه، برنامه های مختلفی برای کمک به افراد در پیدا کردن و مقایسه نقاط پارک مناسب وجود دارند که برخی از آن ها حتی امکان رزرو پارکینگ را در اختیار کاربران خود قرار می دهند.

از جمله برنامه هایی که در زمینه‌ی پارکنیگ های درون شهری در آمریکا فعالیت می کنند می توان به Best Parking و یا ParkWhiz اشاره کرد.

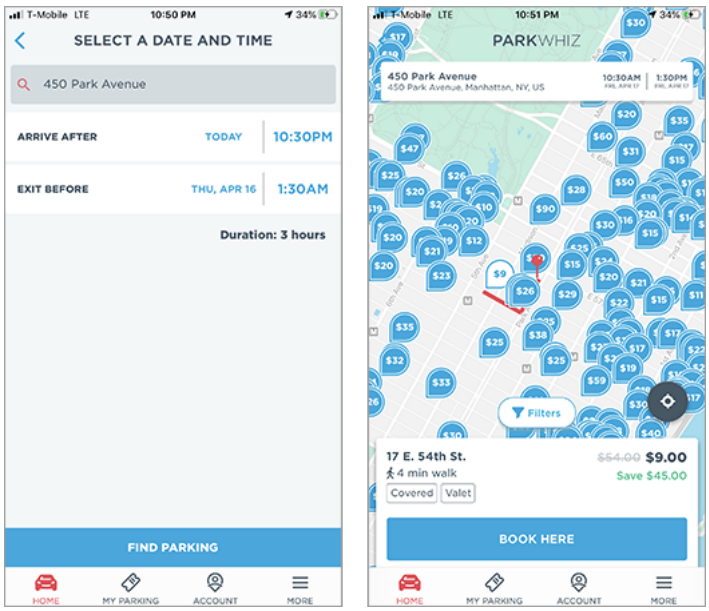
۱ـ۴ـ۱ نیازهای عملکردی ParkWhiz

از مهمترین نیازمندی‌های این برنامه می‌توان به موارد زیر اشاره کرد:

- ثبت نام و ورود به برنامه
- وارد کردن اطلاعات در نمایه و تغییر انها
- پیدا کردن پارکینگ‌ها برای الان یا برای بعد ، مقایسه قیمت آن‌ها و فاصله آن‌ها تا مقصد اصلی کاربر

- رزرو کردن پارکینگ در ساعت دلخواه
- اتصال به سرویس‌های مسیریابی برای رسیدن به پارکینگ
- پرداخت راحت بوسیله کارت‌های اعتباری یا دروازه‌های پرداخت دیجیتال
- خروجی گرفتن از رزرو انجام شده و مشاهده اطلاعات رزرو
- پنل ادمین برای مدیریت پارکینگ و ایجاد تغییرات و عملیات‌های مدیریت و گزارش گیری

در شکل ۱ـ۵ نحوه پیدا کردن پارکینگ و مقایسه آن‌ها بر اساس قیمت و فاصله نمایش داده شده است.



شکل ۱ـ۵: جستجو پارکینگ و مقایسه، عکس از [۲۵]

فصل دوم

دسترسی‌پذیری

۲ـ۱ تعریف دسترسی پذیری

دسترسی پذیری[1] به خاصیتی از نرم افزار اطلاق می‌شود که نرم افزار در لحظه، حاضر باشد و از آمادگی انجام وظایف خود برخوردار باشد [۴۵] . در واقع دسترسی پذیری بر پایه ی مفهوم قابلیت اطمینان[2] بنا شده با این تفاوت که مفاهیم ترمیم [3] نیز به آن افزوده شده است.

به طور دقیق‌تر، قابلیت دسترس‌پذیری به توانایی سیستم برای پنهان کردن یا اصلاح شکست‌ها اشاره دارد؛ به طوری که مجموع مدت زمان قطع خدمات از یک مقدار لازم در یک بازه زمانی مشخص بیشتر نشود. شکست از زاویه دید سیستم یا ناظر انسانی در محیط تعریف می شود و به معنای انحراف سیستم از مشخصات[4] آن است به گونه ای که این انحراف قابل مشاهده باشد. آنچه به شکست منجر می شود، خطاست و معمار نرم افزار همواره سعی می کند تا با درک ریشه های شکست سیستمی مقاوت در برابر خطا[5] را بنا کند.

از جمله مواردی که دغدغه ی معماری سیستم در زمینه ی دسترسی‌پذیری است می توان به موارد زیر اشاره کرد:

---

[1] Availability
[2] Reliability
[3] Recovery
[4] Specification
[5] Resilient

- چگونه خطاهای سیستم باید شناسایی شوند؟
- چقدر ممکن است خطاهای سیستم رخ دهند و عواقب رخداد یک خطا چیست؟
- یک سیستم چقدر می تواند خارج از دسترس باشد؟
- چگونه می توان از خطا و در نتیجه شکست نرم افزار جلوگیری کرد؟
- در زمان رخداد خطا چه اقداماتی لازم هستند؟

۲ـ۲ سناریوی عمومی دسترسی‌پذیری

سناریوی عمومی دسترسی‌پذیری به صورت زیر است:

- منبع محرک[1] : چه درون و چه بیرون: افراد، سخت‌افزار، نرم‌افزار، محیط فیزیکی
- محرک[2] : خطا: حذف، خرابی، زمان بندی نادرست، پاسخ نادرست
- محصول[3]: پروسسورهای سیستم، کانال‌های ارتباطی، ذخیره‌سازی مداوم، فرآیندها
- محیط[4]: عملکرد عادی، راه اندازی، خاموش کردن، حالت تعمیر، عملکرد تخریب شده، عملکرد بیش از حد
- پاسخ[5] : جلوگیری از این که یک خطا منجر به یک شکست شود. به این منظور نیاز است که ابتدا خطا شناخته شود. در صورت وجود خطا تاریخچه آن حفظ شود و سیستم‌ها و آدم‌های مرتبط اطلاع داده شود. همچنین نیاز است که بازیابی پس از انجام خطا نیز انجام شود که شامل غیرفعال کردن منبعی که باعث خطا شده است، غیرفعال کردن تا زمانی که تعمیر تکمیل شود، تعمیر خطا و شکست و عمل کردن در مد کاهش‌داده شده[6] تا زمانی که تعمیر تکمیل شود.
- اندازه گیری پاسخ[7]: موارد بسیاری را می‌توان در نظر گرفت که به مقدار زمانی که سیستم دردسترس است اشاره دارد. از جمله آن‌ها می‌توان به درصد در دسترسی‌پذیری، زمانی که تا تشخیص خطا طول می‌کشد، زمانی که تا تعمیر طول می‌کشد، مقدار زمانی که سیستم در حالت کاهش یافته عمل می‌کند اشاره کرد.

---
[1] Source of stimulus
[2] Stimulus
[3] Artifact
[4] Enviroment
[5] Response
[6] degraded mode
[7] Response Measure

۲ـ۳ تاکتیک ها در دسترسی‌پذیری

تاکتیک‌ها در دسترس‌پذیری به گونه ای طراحی شده است که سیستم را قادر می‌سازد تا خطاهای سیستم را تحمل کند تا سرویس ارائه شده توسط سیستم با مشخصات آن مطابقت داشته باشد و به شکست ختم نشود. تاکتیک های دسترسی‌پذیری را می توان به سه دسته‌ی تشخیص خطا[۱] ،بازیابی از خطا[۲] و جلوگیری از خطا[۳] تقسیم بندی کرد.

۲ـ۳ـ۱ تشخیص خطا

اولین قدم برای جلوگیری از شکست تشخیص خطاست.به منظور تشخیص خطا سیستم می تواند از روش های پینگ [۴]، اکو [۵] ، نظارت[۶] Heartbeat، Timestamp، عقل سنجی[۷]، نظارت بر شرایط[۸]، رای دادن [۹]، شناسایی استثنا[۱۰]، خودآزمونی [۱۱] استفاده کرد [۴۷] .

۲ـ۳ـ۲ بازیابی از خطا

تاکتیک‌های بازیابی از خطاها در قالب روش‌های آماده‌سازی و ترمیم و تکنیک‌های بازآفرینی سیستم پیاده سازی می شوند. روش‌های بازآفرینی دغدغه‌ی راه اندازی مجدد یک سیستم شکست خورده را دارند. روش‌های ترمیم افزونگی فعال [۱۲]، افزونگی منفعل [۱۳]، پشتیبان [۱۴]، کنترل استثنا [۱۵]، بازگشت به عقب [۱۶]، به روزرسانی نرم‌افزار [۱۷]، امتحان مجدد [۱۸]، نادیده گرفتن رفتار دارای خطا، تخریب [۱۹]، پیکربندی ،[۲۰] می‌باشند.

و روش های بازآفرینی سیستمی که پس از شکست از اصلاح شده است، شامل سایه [۲۱]، هماهنگ سازی مجدد [۲۲]، راه اندازی مجدد افزایشی [۲۳] است.

---

[1] fault detection
[2] fault recovery
[3] fault prevention
[4] Ping
[5] Echo
[6] Monitor
[7] Sanity Checking
[8] Condition Monitoring
[9] Voting
[10] Exception Detection
[11] self-test
[12] Active redundancy
[13] Passive redundancy
[14] Spare
[15] Exception handling
[16] Rollback
[17] Software Upgrade
[18] Try
[19] The degradation
[20] Reconfiguration
[21] The Shadow
[22] State resynchronization
[23] Escalating restart

۲-۳-۳ جلوگیری از خطا

به جای شناسایی خطاها و سپس تلاش برای بازیابی آن‌ها، در این تاکتیک سیستم سعی می کند تا از خطا جلوگیری کند .هر چند انجام چنین کاری در نگاه اول دشوار بنظر می رسد اما در بسیاری از موارد انجام چنین کاری ممکن است. از روش‌های حذف از سرویس [۱]، تراکنش ها [۲]، مدل پیش‌بینی [۳]، پیشگیری از استثنا [۴]، افزایش مجموعه‌ی صلاحیت‌ها [۵] به این منظور می‌توان استفاده کرد.

۲-۴ طراحی فهرست بازبینی برای دسترسی‌پذیری

فهرست بازبینی زیر برای پشتیبانی از فرایند طراحی و تجزیه و تحلیل نیازکیفی دردسترس بودن طراحی شده است.

۲-۴-۱ تخصیص مسئولیت ها

در زمان تخصیص مسئولیت ها باید [۶] به موارد زیر توجه شود:

- مسئولیت های سیستم را که باید بسیار دردسترس باشند،باید به طور دقیق مشخص شوند.
- اطمینان حاصل شود که مسئولیت‌های اضافی برای تشخیص حذف، خرابی، زمان‌بندی نادرست یا پاسخ نادرست اختصاص داده شده است.
- اطمینان حاصل شود وظائف زیر به بخش‌هایی از سیستم تخصیص داده شده است:
  - ثبت[۷] خطاها
  - اطلاع‌رسانی به موجودیت های موجود در سیستم
  - غیرفعال‌سازی منشا خطا
  - خارج شدن از دسترس به صورت موقت
  - اصلاح یا پوشاندن[۸] خطا
  - فعالیت سیستم به حالت کاهش‌یافته [۹] برده شود

---

[1] Removal from service
[2] Transactions
[3] Predictive Model
[4] Exception prevention
[5] Increase competence set
[6] Allocation of Responsibilities
[7] Log
[8] mask
[9] Degraded

۲-۴-۲ مدل هماهنگی

پس از تعیین مهم‌ترین قسمت‌های سیستم که باید بسیار دردسترس باشند، می بایست موارد زیر در نظر گرفته شوند:

- اطمینان حاصل شود که مکانیسم‌های هماهنگی می توانند حذف، خرابی، زمان‌بندی نادرست یا پاسخ نادرست را تشخیص دهند. برای مثال در نظر بگیرید که آیا هماهنگی نرم‌افزار تاکسی آنلاین در شرایط ارتباطات ضعیف کار خواهد کرد؟
- اطمینان حاصل شود که مکانیزم های هماهنگی امکان ثبت گزارش از خطا، اطلاع رسانی به موجودیت های مناسب، غیرفعال کردن منبع ایجاد کننده خطا، رفع یا پوشاندن خطا و یا عملکرد در حالت کاهش یافته را دارند.
- اطمینان حاصل شود که مدل هماهنگی از جایگزینی مصنوعات استفاده شده مثل پردازنده ها، کانال های ارتباطی، سخت افزار های ذخیره سازی و فرآیندها پشتیبانی می کند. به عنوان مثال، آیا جایگزینی سرور در یک سیستم پارکینگ عمومی اجازه می دهد سیستم به کار خود ادامه دهد؟
- تعیین کنید که آیا این هماهنگی در شرایط ارتباطات کاهش یافته، هنگام راه اندازی[۱]/ خاموش شدن[۲]، در حالت تعمیر[۳] یا تحت کار بیش از حد[۴] به درستی فعالیت می کند. به عنوان مثال، مدل هماهنگی در نرم‌افزار اشتراک خودرو چقدر اطلاعات از دست رفته[۵] را تحمل می کند و با چه عواقبی؟

۲-۴-۳ مدل داده

لازم است ابتدا مشخص شود که کدام قسمت از سیستم باید همواره در دسترس باشد سپس، در این قسمت ها، تعیین شود که انتزاع داده ها، همراه با عملیات[۶] هایی که برای آن ها تعریف شده اند، می توانند باعث کدوم یک از خطاهای حذف، خرابی[۷]، رفتار نادرست در زمان بندی یا پاسخ نادرست شوند. همچنین برای این داده‌های انتزاعی و عملکردها و خصوصیات آن‌ها، اطمینان حاصل شود که در صورت لزوم می توانند به طور موقت غیرفعال شوند؛ به عنوان مثال، اطمینان حاصل شود زمانی که سرور مدیریت پارکینگ عمومی موقتا در دسترس نیست، درخواست های نوشتن بر روی دیسک در حافظه ی نهان ذخیره می شوند تا پس از بازگشت

[1] Startup
[2] Shutdown
[3] Repair mode
[4] OverLoaded
[5] Lost information
[6] Operation
[7] crash

مجدد سرور، پردازش فرآیند آن‌ها ادامه پیدا کند.

۲ـ۴ـ۴ نگاشت در میان عناصر معماری

مشخص شود کدام بخش از سیستم نظیر پردازنده‌ها، کانال‌های ارتباطی و یا سایر بخش‌ها می تواند منجر به ایجاد خطا در سیستم شود. همچنین اطمینان حاصل شود نگاشت میان عناصر معماری از انعطاف کافی جهت بهبود[1] پس از شکست برخوردار باشد.

۲ـ۴ـ۵ مدیریت منابع

در این بخش باید مشخص شود چه منابع حیاتی برای ادامه کار در صورت وجود خطا لازم است و باید از وجود منابع کافی برای ثبت[2] خطا، اطلاع‌رسانی خطا به سایر موجودیت‌ها، غیرفعال‌سازی منشا خطا و تعمیر و بازیابی سیستم از آن، اطمینان حاصل شود.

زمان دردسترس بودن منابع حیاتی مشخص شود و تعیین شود چه منابعی باید در شرایط خاص همچون زمانی که سیستم با خطا مواجه شده و یا در حالت کاهش یافته در حال فعالیت است، باید دردسترس باشند. به عنوان مثال، اطمینان حاصل کنید که صفهای ورودی در سیستم اشتراک خودرو به اندازه کافی بزرگ هستند تا در صورت خرابی سرور پیامهای پیش بینی شده را بافر[3] کند تا پیامها برای همیشه از بین نروند.

۲ـ۴ـ۶ زمان اتصال

نحوه و زمان اتصال عناصر معماری مشخص شود. اگر از اتصال دیرهنگام برای جایگزینی بین اجزایی استفاده شود که خود می توانند منبع خطا در سیستم باشد، اطمینان حاصل شود که استراتژی در دسترس بودن تعیین شده برای پوشش خطاهای احتمالی تولیدشده توسط همه‌ی منابع کافی است. به عنوان مثال، اگر از اتصال دیرهنگام در سیستم تاکسی آنلاین برای جابجایی بین قطعاتی مانند پردازنده هایی که ممکن است در معرض خطا قرار گیرند، استفاده شود ، آیا مکانیزم های تشخیص و بازیابی خطا تعیین شده برای همه اتصال های احتمالی به درستی عمل می کنند؟

۲ـ۴ـ۷ انتخاب فناوری

فناوری های موجود به گونه‌ای تعیین شود که بتوانند به شناسایی خطاها، بازیابی از آن‌ها و یا بازآفرینی مجدد بخش هایی از سیستم که با خطا متوقف شده اند، کمک کند.

همچنین ویژگیهای در دسترس بودن خود فناوری‌های انتخاب شده تعیین شود: پس از برخورد با چه خطاهایی

---

[1] Recovery
[2] Log
[3] buffer

قادر به بازیابی مجدد خواهند بود؟ چه نقص هایی ممکن است توسط این فناوری ها در سیستم وارد شود؟

۲ـ۵ مطالعات موردی

در اینجا مطالعات موردی در ۳ حوزه حمل و نقل درون شهری برای نیاز دسترس‌پذیری انجام شده است.

۲ـ۵ـ۱ تاکسیرانی و حمل‌ونقل درون شهری

برنامه ی تاکسی‌رانی برخط نیازی است تا تقریبا همواره در دسترس کاربران خود باشد؛ اگر برنامه نیاز به بروزرسانی یا تعمیرات داشته باشد، باید سعی شود در طول به عملیات خدمات حیاتی برنامه حفظ شوند و حتما پیش از آغاز بروزرسانی کاربران را از زمان آغاز و تخمین از طول زمان عملیات باخبر سازد.

سرویس هایی نظیر Uber با کمک کافکا از تکنیک هایی نظیر افزونگی داده و پردازش برای بالابردن ضریب دردسترس‌پذیری خود عمل می‌کنند [۴].

در این رابطه می‌توان سناریوهای زیر را نوشت:

سناریو: یک عامل خارجی پیام هایی که توسط سیستم مورد انتظار نیستند را به سیستم ارسال می کند.

- منبع محرک: عامل خارجی
- محرک: پیامی که سیستم انتظار آن را ندارد
- محصول: فرآیند
- محیط: محیط اجرای عادی
- پاسخ: ثبت اطلاعات پیام دریافتی و عدم پذیرش پیام های بیشتر از عامل خارجی مذکور
- اندازه گیری پاسخ: درصد دسترس‌پذیری

سناریو: در اوبر اگر یکی از دیتاسنتر ها fail شود تمامی امکانات آن توسط یک دیتاسنتر دیگر موقتا ارائه می‌شود.

- منبع محرک: heartbeat monitor
- محرک: دیتاسنتر پاسخ‌گو نیست.
- محصول: مدل هیبریدی ابری
- محیط: نرمال، زمان اجرا

- پاسخ: اوبر از هیچ دیتاسنتر بک آپی برای یک location استفاده نمی‌کند و در هنگام شکست با استفاده از ابزار Terraform بار آن دیتا سنتر را به سایرین انتقال می‌دهد.
- اندازه گیری پاسخ: درصد در دسترس‌پذیری

۲-۵-۲ اشتراک خودرو و حمل‌ونقل درون شهری

در برنامه‌ی اشتراک گذاری خودرو دردسترس پذیری از اهمیت بالایی برخوردار است، زیرا کاربران در صورتی که برنامه در هنگام برنامه ریزی برای سفر و یا در هنگام سفر دردسترس نباشد با مشکلات جدی ای مواجه خواهند شد.در صورتی که برنامه نیاز به بروزرسانی یا تعمیر دارد، پیش از انجام عملیات باید از وجود نقطه ی بازگشت[1] مناسب اطمینان حاصل شود تا در صورت نیاز وضعیت سیستم به پیش از زمان بروزرسانی بازگشت داده شود. سرویسی نظیر waze carpool علاوه بر نیز خط مشی‌های دیگری نظیر این که یک کاربر فقط می‌تواند دو سفر در روز داشته باشد را نیز اجرا می‌کند، این خط مشی علاوه بر جلوگیری از در درآمدزایی می‌تواند به دسترس‌پذیری بیشتر این سرویس نیز کمک کند [۴۳].

در صورت نیاز به بروزرسانی یا تعمیر باید حداقل از یک روز قبل به کاربران در مورد زمان آغاز و طول بازه‌ی عملیات آگاهی داده شود و همچنین امکان استفاده از برخی امکانات برنامه نظیر برنامه ریزی و اشتراک سفر با استفاده از سرور های پشتیبان حفظ شود.

در این رابطه می‌توان سناریوهای زیر را نوشت:

سناریو: کاربر نهایی زمانی که سرور ها در حال به روزرسانی هستند، درخواست ارسال می کند.

- منبع محرک: کاربر نهایی
- محرک: سرور قادر به پردازش درخواست نیست
- محصول: سرور
- محیط: در دست تعمیر
- پاسخ: سرور وضعیت تعمیر را به کاربر اعلام کند.
- اندازه گیری پاسخ: زمان لازم برای انجام به روزرسانی

سناریو: وقتی که لود بر روی سرور پردازش درخواست‌های برنامه‌ریزی زیاد است، در حالت کاهشی سرور عمل کند تا سرور از دسترس خارج نشود.

[1] Restore Point

- منبع محرک: کاربر نهایی
- محرک: درخواست برنامه‌ریزی سفر
- محصول: سرور
- محیط: overloaded
- پاسخ: به اشخاص مناسب (افراد یا سیستم‌ها) اطلاع رسانی شود. و در حالت degraded عمل کند تا تمهیدات برای بازگشت به حالت نرمال انجام شود.
- اندازه گیری پاسخ: زمان تا بازگشت به حالت نرمال، درصد دسترس پذیری

۲-۵-۳ پارکنیگ های عمومی و حمل‌ونقل درون شهری

اختلال در برنامه ی مدیریت پارکینگ زمانی که فرهنگ استفاده از برنامه در شهری جا افتاده است، می تواند عواقب بدی را به دنبال داشته باشد. بهتر است به روزرسانی سرور تا جایی که ممکن است منطقه به منطقه و در زمانی که استفاده از خیابان ها و پارکینگ ها به حداقل می رسد صورت پذیرد. برنامه‌هایی مثل Parkwhiz که به صورت گسترده در جهان استفاده می‌شوند نیاز به دسترسی‌پذیری تمام مدت دارند مخصوصا در شهرهای شلوغی مثل نیویورک که پارکینگ‌ها ۲۴ ساعته هستند [۱] . همچنین اگر پس از کامپایل مولفه ای به برنامه اضافه می شود، همانند اضافه شدن پرداخت با تلفن همراه، معماری باید از اینکه اضافه شدن مولفه ی جدید خللی در نیاز دردسترس پذیری ایجاد نمی کند، اطمینان حاصل کند. برای مثال، فرض کنید اضافه کردن یک درگاه پرداخت جدید سبب بروز خطا در سیستم شده و استفاده از آن را دچار مشکل سازد و در نتیجه سیستم از دسترس کاربران خارج شود.

در این رابطه می‌توان سناریوهای زیر را نوشت:

سناریو: در اوبر اگر یکی از دیتاسنتر ها fail شود تمامی امکانات آن توسط یک دیتاسنتر دیگر موقتا ارائه می‌شود.

- منبع محرک: heartbeat monitor
- محرک: سرویس پاسخ‌گو نیست.
- محصول: سرور
- محیط: نرمال، زمان اجرا

- پاسخ: مطلع کردن اپراتور
- اندازه گیری پاسخ: درصد در دسترس‌پذیری

سناریو: کاربر نهایی زمانی که سرور ها در حال به روزرسانی هستند، درخواست ارسال می کند.

- منبع محرک: کاربر نهایی
- محرک: سرور قادر به پردازش درخواست نیست
- محصول: سرور
- محیط: در دست تعمیر
- پاسخ: سرور وضعیت تعمیر را به کاربر اعلام کند.
- اندازه گیری پاسخ: زمان لازم برای انجام به روزرسانی

فصل سوم

قابلیت همکاری

۳ـ۱ تعریف قابلیت همکاری

قابلیت همکاری [1] نیازمندی‌ای است که به میزان توانایی دو یا بیشتر سیستم، در انتقال معنادار اطلاعات از طریق رابط [2] هایشان در یک زمینه ی خاص می پردازد. این تعریف علاوه بر توانایی مبادله‌ی صحیح اطلاعات که به قابلیت همکاری نحوی [3] منجر می شود به بعد فهم صحیح اطلاعات مبادله شده توسط طرفین (قابلیت همکاری معنایی [4]) نیز تاکید دارد [۵۳] . زمانی که از قابلیت همکاری صحبت به میان می آید بررسی یک سیستم در انزوا معنا نمی دهد فلذا باید در مورد سیستم‌های همکار، نحوه ی همکاری و شرایط همکاری صحبت کرد.

از جمله دلایلی که قابلیت همکاری در میان چندین سیستم را دارای اهمیت می سازد، دو مورد زیر مطرح هستند:

۱. سیستم طراحی شده قرار است خدماتی را در اختیار سایر سیستم های از پیش شناخته شده یا نشده قرار دهد. برای مثال Google Maps سیستمی است که خدمات زیادی را به سایر سیستم‌ها در حوزه حمل و نقل درون شهری از طریق رابط‌های خود ارائه می‌دهد که در هر ۳ سیستم تاکسیرانی آنلاین، اشتراک خودرو

[1] Interoperability
[2] Interface
[3] Syntatic Interoperability
[4] semantic Interoperability

و پارکینگ عمومی کاربرد دارد[۳۲].

۲. سیستم متشکل از زیر سیستم‌هایی است که هر یک وظایف مشخصی را بر عهده دارند و همچنین این زیر سیستم‌ها به یک دیگر وابسته بوده و خدماتی را به یکدیگر ارائه می دهند. برای مثال تک تک سیستم‌های حوزه حمل و نقل درون شهری که دارای که شامل چندین زیر سیستم است.

فرآیند همکاری دو مهم کشف[۱] و بکارگیری پاسخ ها [۲] را در بر دارد. مشتریان یک سیستم پیش از و یا در زمان همکاری با یک سیستم نیاز دارند تا از مکان[۳]،اطلاعات هویتی[۴]و همچنین رابط های در دسترس سیستمی که قصد همکاری با آن را دارند اطلاع داشته باشند. در زمان ایجاد همکاری و دریافت پیام از سایر سیستم‌ها، سیستم می‌تواند یکی از سه رویکرد زیر را در پیش گیرد:

۱. حالت اول این است که سیستم پاسخ را به سیستمی که درخواست را ارسال نموده است بازگرداند. همانند بسیار از سیستم های Server-Client : که می‌توان به درخواست تاکسی آنلاین اشاره کرد که درخواست کاربر به سرور ارسال می‌شود.

۲. گاهی ممکن است پاسخ به درخواست کننده‌ی سرویس بازگشت داده نشود و در عوض پاسخ به سیستمی دیگر ارسال گردد.

۳. گاهی سیستم، پاسخ را برای تمامی سیستم هایی که ممکن است علاقه‌مند باشد ارسال می کند.

۳ـ۲ سناریوی عمومی قابلیت همکاری

سناریوی عمومی قابلیت همکاری به صورت زیر است:

- منبع محرک[۵] : سیستمی که شروع کننده‌ی همکاری است.
- محرک[۶] : درخواست تبادل اطلاعات میان سیستم‌ها
- محصول[۷]: سیستمی که قصد تبادل اطلاعات با آن وجود دارد: برای مثال هر سرویس ثالثی که سرویسی را در حوزه حمل و نقل ارائه می‌دهد، مانند سرویس پیدا کردن مسیر بین دو نقطه
- محیط[۸]:مجموعه سیستم‌هایی که قصد تبادل اطلاعاتی میان آن‌ها وجود دارد و در هنگام اجرا و یا پیش

[1] Discovery
[2] Handling of the response
[3] Location
[4] Identity
[5] Source of stimulus
[6] Stimulus
[7] Artifact
[8] Enviroment

از آن شناخته می‌شوند.

- پاسخ[1] : درخواست برای همکاری باعث تبادل اطلاعات می‌شود. اطلاعات توسط طرف گیرنده از نظر نحوی و معنایی قابل درک است. همچنین، این درخواست رد می شود و نهادهای مربوطه از آن مطلع می شوند. در هر حال ممکن است درخواست های انجام شده توسط سیستم‌ها ثبت[2] شود.
- اندازه گیری پاسخ[3]: درصدی از اطلاعات که به درستی مبادله شده‌اند و یا درصدی از اطلاعات مبادله شده که به درستی پذیرفته نشده‌اند، می توانند روش های اندازه گیری پاسخ باشند.

برای دست یابی به هر نیازمندی مجموعه ای از تکنیک‌ها را به کار می بندیم که در مورد قابلیت همکاری این تکنیک ها شامل مکان‌یابی[4] و مدیریت ارتباط‌ها [5] است.

در دسته بندی مکان یابی روش کشف سرویس [6] در زمان اجرا اقدام به کشف سیستمی که قصد تبادل پیام با آن را داریم می کند. یک سرویس می تواند با استفاده از نوع سرویس، نام، مکان و سایر خصیصه‌ها مکان‌یابی شود.

در مقابل در دسته بندی مدیریت ارتباط ها دو روش معمول هماهنگ سازی[7] و درخورسازی رابط[8] از جمله روش های معمول به شمار می روند. روش هماهنگ‌سازی با استفاده از مکانیزم کنترلی می‌تواند امکان هماهنگی و مدیریت توالی فراخوانی سرویس‌های خاص را فراهم سازد و در کنار آن روش درخورسازی ارتباط به افزودن یا کاهش امکانات به یک رابط توجه دارد.

در زمینه ی قابلیت همکاری به دلیل وجود چالش های مشترک در معماری های متفاوت استانداردهای از پیش تعریف شده ی زیادی وجود دارند اما به دلیل سیر تکاملی که استاندارد ها معمولا طی می کنند، نمی توان معماری سیستم را بر پایه ی استاندارد ها طراحی کرد. پیشنهاد می شود که همواره ابتدا با توجه به سایر فاکتور های تاثیر گذار معماری انتخاب شود و سپس از میان استاندارد های منطبق با معماری، به انتخاب استاندارد های مناسب با معماری پرداخته شود.

۳ـ۳ طراحی فهرست بازبینی برای قابلیت همکاری

در ادامه فهرستی را برای پشتیبانی از روند طراحی و تجزیه و تحلیل برای قابلیت همکاری ارائه می‌شود.

---
[1] Response
[2] logged
[3] Response Measure
[4] Locate
[5] Manage Interface
[6] Discover System
[7] Orchestrate
[8] Tailor interface

۳-۳-۱ تخصیص مسئولیت ها

در زمان تخصیص مسئولیت‌ها باید [1] به موارد زیر توجه شود:

- معمار سیستم باید مشخص کند که انجام کدام یک از وظائف سیستم نیازمند همکاری با سایر سیستم‌هاست.
- اطمینان حاصل شود که مسئولیت هایی به جهت تشخیص درخواست‌های همکاری با سیستم‌های خارجی شناخته شده و یا ناشناخته اختصاص یافته است.
- معماری باید مسئولیت پاسخ به وظائف زیر را در سیستم لحاظ کرده باشد:
  - پذیرش درخواست همکاری
  - تبادل اطلاعات
  - عدم پذیرش درخواست همکاری
  - اعلان و آگاهی سازی هویت های مرتبط با سیستم از همکاری با سایر سیستم ها
  - ثبت درخواست

۳-۳-۲ مدل هماهنگی

مدل هماهنگی[2] باید با در نظر گرفتن دغدغه و نیاز های معماری به آن‌ها پاسخ دهد. مواردی که برای پاسخ دهی به نیاز کارایی[3] باید توسط مدل هماهنگی مورد توجه قرار گیرند شامل موارد زیر است:

- حجم ترافیکی که به صورت مستقیم توسط سیستم‌های تحت کنترل و حتی سیستم‌هایی که کنترلی بر روی ترافیک آن ها ندارد ایجاد می شود.
- ارسال به موقع پیام‌ها از سمت سیستم
- هم زمانی[4] ارسال پیام از سمت سیستم
- و شاید مهم‌تر از همه این است که اطمینان حاصل شود سیستم‌های تحت کنترل فرضیات قابل انطابقی در همکاری با سایر سیستم هایی که کنترلی بر روی آن ها وجود ندارد، در نظر دارد.

---
[1] Allocation of Responsibilities
[2] Coordination Model
[3] Performance
[4] Currency

۳-۳-۳ مدل داده

مدل داده[1] از مهم ترین بخش هایی است که اگر به درستی بر آن نظارت صورت نپذیرد می تواند عواقب سنگینی را به همراه داشته باشد.در مدل داده ای باید از برداشت یکسان طرفین همکاری از اطلاعات اطمینان حاصل کنید.به این منظورم لازم است تا انتزاعات اصلی داده های مبادله شده از نظر نحو و معناشناسی مورد بررسی دقیق قرار بگیرند.

۳-۳-۴ نگاشت در میان عناصر معماری

جدا از ملاحظاتی که در نگاشت در میان عناصر معماری[2] درباره امنیت، دسترس‌پذیری و کارائی وجود دارد و در سایر فصل های این مستند به آن ها پرداخته خواهد شد؛ در رابطه با قابلیت همکاری مساله ی مهم نگاشت درست از اعضای سازنده ی پردازنده ها است.

۳-۳-۵ مدیریت منابع

لازم است معمار نرم افزار اطمینان حاصل کند منابعی که سیستم به جهت همکاری با سایر سیستم ها نیاز دارد هیچ گاه سیستم را تحت فشار غیر قابل تحمل قرار نخواهد داد و بار منابع تحمیل شده توسط الزامات همکاری همواره قابل قبول است.

در ضمن نیاز یک سیستم ناظر با هدف تخصیص منابع به صورت منصفانه و بر پایه ی سیاست های تبیین شده در معماری باید حتما دیده شود.

۳-۳-۶ زمان اتصال

لازم است سیستم هایی که ممکن است با یکدیگر همکاری داشته باشند شناسایی شده و از رعایت موارد زیر در مورد آن‌ها اطمینان حاصل شود.

- از وجود یک سیاست مشخص به منظور همکاری با سیستم‌های شناخته شده و ناشناخته خارجی اطمینان حاصل کنید.
- از وجود قوانینی برای عدم پذیرش درخواست‌ها و ثبت درخواست‌هایی که پذیرفته نشده‌اند اطمینان حاصل کنید.
- در صورت تاخیر در اتصال، اطمینان حاصل شود که مکانیزم‌هایی از کشف خدمات یا پروتکل‌های جدید مرتبط یا ارسال اطلاعات با استفاده از پروتکل‌های انتخاب شده پشتیبانی می کنند.

[1] Data Model
[2] Mapping among Architectural Elements

۳ـ۳ـ۷ انتخاب فناوری

در انتخاب فناوری باید روی این مساله تمرکز کرد که انتخاب هر فناوری چه تاثیراتی بر روی رابط‌ها و همکاری سیستم با دیگر سیستم‌ها خواهد داشت. بررسی شود آیا فناوری انتخاب شده توانایی پاسخ گویی به نیاز هایی که توسط قابلیت همکاری مطرح می شوند را دارا هستند و یا خیر؟

۳ـ۴ مطالعات موردی

در اینجا مطالعات موردی در ۳ حوزه حمل و نقل درون شهری برای نیاز قابلیت همکاری انجام شده است.

۳ـ۴ـ۱ تاکسیرانی و حمل‌ونقل درون شهری

انجام پرداخت‌ها زمانی که کاربر یک سرویس تاکسی را استفاده می‌کند معمولا با استفاده از یک نهاد سوم انجام می‌شود، چرا که امروزه بیشتر تبادلات دیگر به صورت وجه کاغذی نقد انجام نمی‌شود و با پیشرفت تکنولوژی نیز گوشی‌ها توانسته‌اند به ابزار خوبی برای این کار تبدیل شوند. در تمامی اپ‌های مورد بررسی در حوزه تاکسی‌رانی همگی از یک نهاد ثالث برای پرداخت استفاده می‌کنند [۱۴] و بنابراین نیاز است که درک مشترکی از نظر نحوی و معنایی در مورد اطلاعات مبادله شده داشته وجود داشته باشد. همچنین از آنجا که نیاز است کاربر موقعیت جغرافیایی خود را به سیستم اعلام کند، در این اپ‌ها معمولا از یک ادغام‌سازی با آنچه که گوگل سرویس می‌دهد نیز انجام می‌شود تا تاکسی بتواند درست به کاربر برسد و مسیریابی مناسبی تا مقصد او نیز انجام دهد. جزئیات این موضوع کاملا مشخص نیست و هر کدام از آن‌ها در یک سطح خاصی از این سرویس استفاده می‌کنند. در مستندات پروژه MyWay اعلام شده است که معماری اجازه دارد از یک سرویس ثالث جستجو برای انتخاب مسیر بین دو نقطه استفاده کند. ماژول رابط به نحوی انعطاف‌پذیر است که می‌توان داده جدید را به ازای هر دسته از داده‌ها اضافه کرد. هر سیستم ثالثی اجازه دارد فرانت‌اند[1] خود را با استفاده از بک‌اند[2] برنامه پیاده‌سازی کند. مدل داده به راحتی قابل توسعه دادن است تا اجازه دهد که پیشرفت‌های آتی در پروژه قابل اجرا کردن باشد. برنامه از تاریخچه داده‌های واسط برای ماژول ارزیابی استفاده می‌کند. فرانت های برنامه شامل مولفه های برنامه‌ی iOS، اندروید و وب است؛که این مولفه به واسطه رابطه های تعیین شده از سمت بک‌اند اقتدام به مبادله ی داده های خود می‌کنند. برنامه برای تبلت‌های با سایز مختلف نیز سازگار باشد. همچنین برنامه وب نیز باید وجود داشته باشد که توسط مرورگرهای فایرفاکس[3] و کروم[4] و سافاری[5] به خوبی

[1] Front-End
[2] Back-End
[3] Firefox
[4] Chrome
[5] Safari

پشتیبانی شود. نوتیفیکشن‌ها نیز به صورت پوش[1] ارسال شود. قابلیت استفاده و ادغام دادگان نقشه گوگل و نقشه‌های محلی با فرمت‌های به خصوصی نیز توسط اپلیکیشن فراهم باشد [۲۳].

۳-۴-۲ اشتراک خودرو و حمل‌ونقل درون شهری

در برنامه‌های اشتراک خودرو، اکثر معماری ها از جنس میکروسرویس‌ها[2] توسعه داده شده اند؛ پس ارتباط میان این میکروسرویس ها در عملکرد صحیح برنامه نقش حیاتی‌ای را بازی می‌کند. معماری در این برنامه ها باید از درخورسازی مناسب رابط های میان میکروسرویس های متفاوت اطمینان حاصل کند.معماری باید مطمئن شود میکروسرویس ها درک متقابل صحیحی از معنای پیام های مبادله شده دارند و همچنین از نظر نحوه ی بیان برطبق توافق مشترک از پیش تعیین‌شده ای اقدام به مبادله ی اطلاعات می کنند. همچنین سیستم های اشتراک خودرو نظیر بسیاری دیگر از برنامه های فعال در حوزه‌ی حمل‌ونقل از سرویس‌دهنده های بزرگ GIS اطلاعات نقشه‌های خود را دریافت می‌کنند و اطمینان از وجود قابلیت همکاری با یکی از این GIS ها شاید مرکز عملکرد صحیح این دسته از برنامه ها به حساب آید. شرکت waze با همکاری شرکت ArcGIS که یک شرکت سرویس‌دهنده موقعیت جغرافیایی است به این هدف جامعه عمل می‌پوشاند [۱۳] . از آنجا در یک برنامه‌ی پشنهاد سفر درخواست یک هم سفر زیر سیستم های مختلفی از سیستم را درگیر می‌کند و ان عمل پیچیده به همکاری و فراخوانی به ترتیب خاصی از زیر سیستم هایی نظیر مکان‌یابی و پیشنهاددهنده های متفاوت نیاز دارد، که حتما به تنظیمات دقیق از پیش تعیین‌شده نیاز خواهد داشت.

۳-۴-۳ پارکنیگ های عمومی و حمل‌ونقل درون شهری

در مورد مطالعاتی پارکینگ های عمومی، وجود زیر سیستم های سخت افزاری نظیر سنسور های تشخیص وجود خودرو در پارکینگ ها از اهمیت بالای نیاز قابلیت همکاری در چنین برنامه هایی خبر می‌دهد.در دنیای سخت‌افزار و تکونولوژی های ارتباطی به دلیل رشد سریع دانش، تغییرات به سرعت رخ می‌دهند.برای مثال، اگر برنامه برای تشخیص فضاهای خالی در پارکینگ از سنسور خاصی استفاده می کند، با ورود نوع پیشرفته تر سنسور ها با عملکرد بهینه تر به بازار این دسته از برنامه ها باید از قابلیت همکاری سایر زیرسیستم ها با زیرسیستم مرتبط به سنسورهای جدید اطمینان حاصل کنند. از آنجا که بیشتر این سنسورها از پروتکل‌های معروف برای ارتباط استفاده می‌کنند در نتیجه شرکت‌های زیادی در این رابطه رقابت می‌کنند و هر پارکینگی می‌تواند از سنسورهای شرکت‌های متفاوتی استفاده کند این سنسورها بدلیل استفاده از تکنولوژی آلتراسونیک نسبت ارزان قیمت هستند و در نتیجه می‌تواند در ابعاد بالا از آن‌ها استفاده شود [۳۴] . در برنامه‌ی پارکینگ آنلاین ممکن است هر

[1] Push
[2] Micro-Service

پارکینگ از تجهیزات متفاوتی استفاده کند و در نتیجه سرویس های متفاوتی ارائه شود؛ کاربران نهایی از یک برنامه‌ی یکتا برای جستجوی سرویس استفاده می‌کنند و فلذا این برنامه باید قابلیت جستجوی سرویس های فعال در یک پارکینگ را داشته باشد و با رابط های مناسب با آن ارتباط برقرار کند.

در رابطه با تاکسی‌رانی آنلاین و اشتراک‌گذاری خودرو و پارکینگ آنلاین برای دو قسمت پرداخت و اطلاعات جغرافیایی می‌توان سناریوهای زیر را نوشت:

سناریو: سیستم درخواست تاکسی یا اشتراک‌گذاری خودرو را دریافت کرده و با اطلاعات مکانی فعلی و مقصد ما راننده مورد نظر را معرفی و به سمت ما هدایت می‌کند.

- منبع محرک: سرور
- محرک: درخواست برای تبادل اطلاعات با گوگل مپ
- محصول: سسیتمی که می‌خواهد تبادل اطلاعات کند.
- محیط: زمان اجرا، پیشتر سرویس ها با یکدیگر آشنا شده اند.
- پاسخ: قبول یا رد درخواست و حفظ تاریخچه درخواست توسط جزهای مشارکت‌کننده در این تبادل (اطلاعات جغرافیایی)
- اندازه گیری پاسخ: درصد اطلاعاتی که به درستی پردازش یا رد شده است.

سناریو: کاربر در هنگام پرداخت به درگاه پرداخت هدایت می‌شود و پس از انجام عملیات پرداخت یا تمام شدن مهلت کاربر را به اپ باز می‌گرداند. در این حین سیستم تاکسی‌رانی یا اشتراک خودرو با درگاه پرداخت در حال تبادل اطلاعات این که پرداخت صورت گرفته یا نه هستند.

- منبع محرک: سرور
- محرک: درخواست برای تبادل اطلاعات با درگاه پرداخت
- محصول: سسیتمی که می‌خواهد تبادل اطلاعات کند.
- محیط: زمان اجرا، پیشتر سرویس ها با یکدیگر آشنا شده اند.
- پاسخ: قبول یا رد درخواست و حفظ تاریخچه درخواست توسط جزهای مشارکت‌کننده در این تبادل (گزارش پرداخت)
- اندازه گیری پاسخ: درصد اطلاعاتی که به درستی پردازش یا رد شده است.

فصل چهارم

اصلاح‌پذیری

۴ـ۱ تعریف اصلاح‌پذیری

اصلاح‌پذیری [1] یا همان قابلیت اصلاح یکی از موارد مهمی است که تقریبا در تمامی مدل‌های کیفیت نرم‌افزار از جمله نرم‌افزارهای حوزه حمل و نقل درون شهری مورد سنجش قرار می‌گیرد. در این نیازمندی غیرعملکردی به اهمیت اصلاح‌پذیری نرم‌افزار با هزینه کم پرداخته می‌شود.

برای اصلاح‌پذیری تعاریف زیادی می‌توان ارائه نمود که از جمله‌آن‌ها می‌توان به موارد زیر اشاره کرد [۴۹].

۱. چقدر نرم‌افزار قابل تغییر است؟

۲. هزینه تغییر

۳. میزان سهولت انجام تغییر

مطالعات زیادی نشان داده است که با پرداختن به ویژگی اصلاح‌پذیری می‌توان در حین توسعه به طور چشم‌گیر هزینه یک سیستم نرم‌افزاری را کاهش داد. همچنین به موجب پرداختن به این ویژگی به معمار نرم‌افزار این امکان داده می‌شود که هزینه‌های نگهداری را پیش‌بینی و ریسک و انعطاف پذیری نرم‌افزار را ارزیابی کند.

[1] Modifiability

۲ـ۴ سناریوی عمومی اصلاح‌پذیری

سناریوی عمومی قابلیت اصلاح‌پذیری به صورت زیر است:

- منبع محرک[1] : هر یک از کاربران، توسعه دهندگان، ادمین‌ها؛ برای مثال در حمل و نقل درون شهری یا اشتراک‌گذاری خودرو کاربران نهایی درخواست دهنده سرویس تاکسی یا خودرو، ادمین‌ها، مدیران و توسعه دهندگان تمامی نرم‌افزارهای این حوزه از جمله مثال‌های منبع محرک هستند.
- محرک[2] : اضافه، حذف یا مورد تغییر قرار دادن یک عملکرد، یا تغییر دادن یک ویژگی کیفیت یا ظرفیت یا تکنولوژی: برای مثال کاربران و ادمین‌های نرم‌افزارهای حوزه حمل ونقل درون شهری بسته به دسترسی‌هایی که به نرم‌افزار دارند می‌توانند داده خود یا بقیه را مورد تغییر قرار داده و توسعه‌دهندگان آن‌ها نیز می‌توانند کد را تغییر دهند.
- محصول[3]: کد، داده، مولفه‌ها و ...
- محیط[4]: در زمان کامپایل، زمان اجرا، زمان ساخت، زمان طراحی
- پاسخ[5] : در پاسخ مناسب یک یا چند عملیات رخ می‌دهد. در این عملیات‌ها انجام و تست و استقرار تغییر انجام می‌شود.
- اندازه گیری پاسخ[6]: برای اندازه‌گیری پاسخ از معیارهای هزینه و تعداد و سایز و پیچیدگی موارد مورد بررسی، تلاش و .. استفاده می‌شود.

تاکتیک‌های زیادی را می‌توان برای نرم‌افزارهای حوزه حمل و نقل درون شهری در نظر گرفت که بهبود این نیاز کمک کند که هدف همه آن‌ها کنترل پیچیدگی تغییرات و انجام تغییرات، تست و استقرار تغییرات در زمان و هزینه تعیین شده است.

یکی از این تاکتیک‌ها کم کردن حجم ماژول‌های برنامه است. اگر بتوان یک ماژول که را به ماژول‌های کوچکتر تبدیل کرد، آنگاه در این صورت هزینه تغییرات در آینده نیز به صورت میانگین کاهش پیدا می‌کند. چرا که هرچه حجم ماژول بیشتر باشد هزینه تغییر آن زیادتر است.

---

[1] Source of stimulus
[2] Stimulus
[3] Artifact
[4] Enviroment
[5] Response
[6] Response Measure

تاکتیک دیگر افزایش مقدار چسبندگی [۱] است. به این صورت که اگر دو وظیفه داخل یک ماژول یک هدف را دنبال نمی‌کنند آنگاه باید آن دو در ماژول‌های متفاوتی قرار بگیرند. این کار ممکن است موجب ایجاد ماژول‌های جدید یا جابه‌جایی وظایف به ماژول‌های موجود شود.

همچنین بر اساس اصل اتصال آزادانه [۲] تلاش می‌شود که تا حد ممکن وابستگی‌ها ماژول‌ها کاهش یابد و آن‌ها را محدود کرد [۲۲]. عملیات باز تولید [۳] را نیز بر اساس همین اصل می‌توان زمانی که دو ماژول تحت تاثیر یک تغییر، تاثیر می‌پذیرند انجام داد. چرا که در واقع قسمتی از آن دو کپی یکدیگر هستند. و در آخر باید گفت که هر چه عملیات اتصال را بتوان به تعویق بیندازیم تا انعطاف‌پذیری تغییر هر جز حفظ شود و هزینه تغییر خاص هر کدام کاهش یابد بهتر است.

به منظور بهره بری از این تاکتیک‌ها می‌توان با توجه به هریک از پروژه‌های تاکسی آنلاین، اشتراک‌گذاری خودرو و مدیریت پارکینگ یک فهرست بازبینی تنظیم کرد. این موضوع به بهبود نیاز اصلاح‌پذیری پاسخ می‌دهد.

۴-۳ طراحی فهرست بازبینی برای اصلاح‌پذیری

۴-۳-۱ تخصیص مسئولیت ها

در حوزه نرم‌افزارهای حمل و نقل درون شهری ممکن است که تغییراتی در هر یک بخش‌های فنی، قانونی، اجتماعی، تجاری نرم‌افزار مربوطه رخ دهد. برای مثال نیاز به یه یک عملکرد جدید در بخش فنی تاکسی آنلاین یا تغییر قانون و سیاست‌های این حوزه برای مدیریت پارکینگ‌ها یا بخش اجتماعی و بازخورد جامعه به نحوه اشتراک‌گذاری خودرو، در هر یک ممکن است که باعث شود، بخشی تغییر کند. باید دید که چه تغییراتی احتمال دارد رخ دهد و برای هر تغییر بالقوه مسئولیت‌هایی تعیین شود. همچنین مسئولیت‌هایی که تغییر می‌کنند باید در یک ماژول قرار بگیرند و مسئولیت‌هایی که در زمان‌های مختلف تغییر می کنند در ماژولهای جداگانه قرار بگیرند تا به بهبود این نیاز کمک کنند.

۴-۳-۲ مدل هماهنگی

در این قسمت تعیین شود که کدام ویژگی یا ویژگی کیفیت می تواند در زمان اجرا تغییر کند و این موضوع چگونه بر هماهنگی تأثیر می گذارد. در حوزه نرم‌افزارهای حمل و نقل درون شهری ممکن است که هر یک از دستگاه‌ها، پروتکل‌ها و مسیرهای ارتباطی مورد استفاده برای هماهنگی تغییر پیدا کنند. برای مثال ممکن است پروتکل ارتباطی سنسورهای پارکینگ در زمان اجرا تغییر یابد و یا اطلاعات منتقل شده از کاربران در برنامه‌های

[1] cohesion
[2] loose coupling
[3] refactor

درخواست تاکسی آنلاین یا اشتراک خودرو در زمان اجرا تغییر پیدا کند. به این منظور برای آن دستگاه ها، پروتکل ها و مسیرهای ارتباطی که محتمل است تغییر پیدا کنند اطمینان حاصل شود که تأثیر تغییرات فقط به مجموعه کوچکی از ماژول ها محدود می شود.

۴-۳-۳ مدل داده

در این قسمت تعیین می‌شود که چه تغییراتی (یا دسته بندی تغییرات) در انتزاع داده ها، عملکردهای آنها یا خصوصیات آنها محتمل است که رخ دهد. در حوزه نرم‌افزارهای حمل و نقل درون شهری ممکن است که تغییر یا دسته تغییراتی در انتزاعات داده‌ها شامل ایجاد، شروع اولیه، ماندگاری، دستکاری، ترجمه یا تخریب هر یک رخ بده. به این منظور برای هر تغییر یا دسته تغییر، باید تعیین شود که تغییرات توسط چه منبع محرکی انجام می‌شود. همانگونه که در بخش قبل گفته شد کاربر نهایی، مدیر سیستم یا توسعه دهنده می‌تواند منبع تغییر باشد. در نهایت باید اطمینان حاصل کرد که ویژگی های مورد نیاز کاربر برای انجام تغییر قابل مشاهده است و کاربر دارای مجوزهای صحیح برای اصلاح داده‌ها، عملکردهای آن یا خصوصیات آن است. در نهایت تخصیص اطمینان حاصل شود تعداد تغییرات بالقوه و شدت اصلاحات در انتزاعات به حداقل ممکن برسد.

۴-۳-۴ نگاشت در میان عناصر معماری

در این قسمت تعیین می‌شود که آیا تغییر نحوه نگاشت قابلیت عناصر محاسباتی (به عنوان مثال فرآیندها، رشته‌ها ، پردازنده ها) در زمان اجرا، زمان کامپایل، زمان طراحی یا زمان ساخت مطلوب است؟ در حوزه حمل و نقل درون شهری نیاز است که تغییرات لازم برای افزودن، حذف یا اصلاح یک تابع یا یک ویژگی کیفیت تعیین شود که این این تعیین به موارد زیر ممکن است وابسته باشد:

- اجرای وابستگی‌ها
- اختصاص داده به پایگاه داده
- اختصاص عناصر زمان اجرا به پردازش ها، رشته‌ها یا پردازنده‌ها

همچنین اطمینان حاصل شود که چنین تغییراتی با مکانیزمی که اتصال را به تاخیر می‌اندازد انجام شود.

۴-۳-۵ مدیریت منابع

در این قسمت تعیین می‌شود که چگونه اضافه کردن، حذف یا تغییر یک مسئولیت یا ویژگی کیفیت بر استفاده از منابع تأثیر می گذارد. به عنوان مثال در تاکسی آنلاین بر اساس رشد روزافرون کاربران، ممکن است سیستم نیاز به معرفی منابع جدید و حذف و جایگزینی منابع قبلی داشته باشد. یا ایجاد محدودیت‌ها روی منابع اشتراک‌گذاری

خودرو که باید اطمینان حاصل کرد که برای تامین نیازهای سیستم کافی هستند. یا ایجاد یک سری واسط برای منابع در مدیریت سیستم‌های پارکینگ تا دسترسی تنها از طریق آن‌ها به صورت کپسوله[1] فراهم شود.

۴-۳-۶ زمان اتصال

در تمامی نرم‌افزارهای حوزه حمل و نقل درون شهری نیاز است که برای هر تغییر یا دسته تغییرات:

- آخرین زمانی که باید تغییر ایجاد شود تعیین شود.
- مکانیزم مناسب برای اتصال دیرهنگام انتخاب شود.
- هزینه معرفی مکانیزم و ایجاد تغییرات با استفاده از مکانیزم انتخاب شده تعیین شود.
- همچنین تعداد زیادی انتخاب اتصال معرفی نشود که خود این موضوع مانع تغییر شود، چرا که وابستگی ها در بین گزینه‌ها می تواند پیچیده و ناشناخته باشد.

۴-۳-۷ انتخاب فناوری

نیاز است که فناوری مناسب برای نرم‌افزارهای حوزه حمل و نقل درون شهری به گونه‌ای در نظر گرفته شود که انجام تست و استقرار پس از انجام اصلاحات را راحت‌تر کند و همچنین تعیین شود که تغییر خود فناوری چقدر راحت است؟ فناوری‌ها را باید به گونه‌ای انتخاب شوند که بیشترین پشتیبانی از تغییرات را داشته باشند.

۴-۴ مطالعات موردی

در اینجا مطالعات موردی در ۳ حوزه حمل و نقل درون شهری برای نیاز اصلاح‌پذیری انجام شده است. همانگونه که قبلا بیان شد باید موارد محتملی که ممکن است تغییر پیدا کند شناسایی شوند و مسئولیت‌های مناسب همراه با زمان بندی برای آن‌ها تعیین گردد. در این راه چند تاکتیک مانند کوچک کردن ماژول‌ها یا ترکیب ماژول‌های مرتبط مطرح شد. از نرم‌افزارهای سنجش کد مانند سونارکیوب [2] نیز می‌توان برای کنترل کردن حجم ماژول‌ها و وابستگی آن‌ها نیز استفاده کرد .[۳۱]

۴-۴-۱ تاکسیرانی و حمل‌ونقل درون شهری

اصلاح‌پذیری را می‌توان یکی از زیر مشخصه‌های قابلیت نگهداری دانست. در مستندات پروژه MyWay عنوان شده است که معماری نرم‌افزار باید منعطف باشد، تا بتوان هر نمونه آن را در هر شهر به صورت محلی اجرا کرد و قابلیت اشتراک داده نیز بین آن‌ها فراهم باشد. همچنین در توسعه این نرم‌افزار برای بازتولید کد نسبت

[1] Capsulated
[2] Sonarqube

به مواردی که توسط نرم‌افزارهای سنجش کد گزارش کرده‌اند، مواردی نیز در نظر گرفته شده که این بازتولید کد چنین مواردی در زمان توسعه موجب بهبود اصلاح‌پذیری می‌شود. همچنین در چنین سیستم هایی می‌توان به تنظیمات سطح کاربر اشاره کرد که انجام تغییرات و بازگرداندن اطلاعات به حالت اولیه به خوبی امکان انجام آن باید فراهم شود [۲۳].

۴-۴-۲ اشتراک خودرو و حمل‌ونقل درون شهری

در اپلیکیشن هایی که در اکوسیستم های تازه شکل گرفته فعالیت می کنند، یکی از مهم‌ترین دغدغه ها، عدم فرهنگ‌سازی در رابطه با چگونگی استفاده از اپلیکیشن است.عدم وجود فرهنگ تثبیت شده در جامعه سبب می شود این اپلیکیشن ها نیاز داشته باشند تا بسیار چابک عمل کرده و همراه با تغییرات به سرعت خود را اصلاح کنند [۲]؛ به عنوان مثال وقوع یک مشکل ممکن است قانون گذاران را مجاب کند تا قوانین بر نحوه ی ارائه ی خدمات توسط اپلیکشن های این حوزه اعمال کنند و در چنین شرایطی معماری باید به خوبی از قابلیت اصلاح پذیری پشتیبانی کند.

۴-۴-۳ پارکنیگ های عمومی و حمل‌ونقل درون شهری

در حوزه ی پارکنینگ های عمومی ممکن است با پیشرفت سخت افزار و زیرساخت پارکینگ های شهری نیاز باشد تا رابط های سیستم به سرعت خود را با تغییرات جدید هماهنگ کنند فلذا اصلاح‌نقش پررنگی در چنین سیستم هایی نیز ایفا می‌کند؛ به عنوان یک مثال دقیق تر فرض کنید که پارکینگ ها در یک شهر شروع به نصب دستگاه هایی برای پرداخت هزینه ها از طریق تلفن های ساخت شرکت اپل کرده اند، معماری سیستم با برخورداری از اصلاح پذیری بالا می تواند به سرعت امکان استفاده از این نوع پرداخت را برای کاربران نهایی فراهم سازد. از روی موارد پشتیبانی اپلیکیشنی مثل ParkWhiz می‌توان چنین تغییراتی را در فرمت‌فایل‌ها، پروتکل‌ها و غیره دید [۳۳].

در رابطه با هر ۳ سیستم‌ها می‌توان سناریوهای زیر را نوشت:

سناریو:کاربر باید بتواند اطلاعات نمایه خود را مورد تغییر قرار بدهد و تغییر انجام شود.

- منبع محرک: کاربر نهایی
- محرک: اضافه، حذف و تغییر اطلاعات
- محصول: زمان اجرا
- محیط: داده

- پاسخ: عملیات مربوطه کاربر (اضافه، تغییر یا حذف) انجام شود.
- اندازه گیری پاسخ: میزان تلاش و مقدار زمان برای انجام تغییر

سناریو: توسعه‌دهنده باید بتواند کد را تغییر دهد و هرچه هزینه این کار کمتر باشد بهتر است.

- منبع محرک: توسعه‌دهنده
- محرک: اضافه‌کردن ویژگی به کد
- محصول: کد
- محیط: زمان توسعه و استقرار
- پاسخ: تغییر انجام شود، تست و استقرار نیز انجام شود.
- اندازه گیری پاسخ: میزان هزینه و ریسک انجام تغییر

که از این سناریو به طور خاص برای پارکینگ آنلاین می‌توان این سناریو را در نظر گرفت:

سناریو: پارکینگ های سطح شهر به سیستم پرداخت با apple pay مجهز می‌شوند.

- منبع محرک: گسترش‌دهندگان سیستم
- محرک: افزودن قابلیت پرداخت با apple pay
- محصول: کد
- محیط: زمان ساخت
- پاسخ: افزودن قابلیت پرداخت با apple pay به کد و تست و کارگزاری[1] در سیستم
- اندازه گیری پاسخ: پیچیدگی و مدت زمان مورد نیاز برای افزودن قابلیت پرداخت با apple pay توسط گسترش‌دهندگان سیستم

[1] Deploy

فصل پنجم

کارایی

۵-۱ تعریف کارایی

کارایی[1] به توانایی سیستم در پاسخ گویی به نیازها در زمانی مشخص اشاره دارد. زمانی که رخدادی جدید نظیر خواسته ای از جانب کاربر یا سایر سیستم‌ها پیش می آید، سیستم باید در بازه ی زمانی مورد انتظار آن رخداد را بررسی کند و به نیاز آن پاسخ دهد. در عموم سیستم‌های حوزه حمل و نقل که به صورت برخط و مبتنی بر وب هستند ممکن است کارایی با گزاره هایی نظیر تعداد درخواست بر دقیقه بیان شود.

کارایی در بیشتر اوقات درکنار گسترش پذیری[2] شنیده می شود،گسترش پذیری به افزایش ظرفیت پاسخ دهی با حفظ کارایی اشاره دارد و نباید مفهوم آن با کارایی یکسان در نظر گرفته شود.

۵-۲ سناریو عمومی کارایی

سناریو کارایی با فرارسیدن رویدادها به سیستم آغاز می شود. سیستم باید هم زمان با وجود سایر رویدادها، با تخصیص منابع به درخواست رویدادی که اکنون به سیستم رسیده نیز پاسخ دهد. رویدادها می‌توانند با الگوی مشخص و تبعیت از یک توزیع ریاضی رخ دهند و یا آن که غیر قابل پیشبینی باشند؛ الگوی فرارسیدن رویداد

[1] Performance
[2] Scalability

ها می تواند به صورت دوره ای[۱]، تصادفی[۲] و یا پراکنده[۳] باشد.

در سناریو عمومی کارایی، معیار سنجش پاسخ سیستم به محرک می تواند یکی از موارد زیر باشد:

- تاخیر[۴]: زمان میان فرارسیدن رویداد و پاسخ سیستم را تاخیر سیستم می‌گویند. برای مثال زمانی که یک کاربر باید منتظر پاسخ جستجوی سفرهای مشابه در برنامه ی اشتراک خودرو بماند.
- مهلت های پردازشی[۵]:یک رویداد ممکن زمان مشخصی را برای سیستم به عنوان حداکثر زمانی که باید پاسخ رویداد آماده باشد مشخص کند، تخطی از این زمان سبب کاهش کارایی سیستم خواهد شد.
- توان عملیاتی سیستم[۶]:تعداد پردازه هایی سیستم در واحد زمان می تواند به پایان برساند به عنوان توان عملیاتی شناخته می شود. برای مثال، سرور یک برنامه تاکسی‌برخط توان پاسخ دادن به تعداد مشخصی درخواست سفر از سمت کاربران در لحظه را دارد.

همچنین معیار های دیگری نظیر واریانس تاخیر[۷] و تعداد رویداد های بلوکه شده نیز وجود دارند که کارایی سیستم به وسیله ی آن ها قابل سنجش است.

در ادامه به معرفی بخش های مختلف سناریو عمومی کارایی پرداخته می‌شود.

- منبع محرک[۸] : یک منبع از داخل و یا خارج از سیستم که بتواند رویدادی را ایجاد و به سیستم ارسال کند،می تواند منبع محرک سناریوی کارایی باشد.برای مثال، کاربران نهایی و یا سایر زیرسیستم ها که سیستم های مورد مطالعه با آن‌ها همکاری می کنند.
- محرک[۹] : فرا رسیدن یک رویداد برای مثال درخواست یک سفر به سیستم تاکسی‌رانی برخط.
- محصول[۱۰]: سیستم یا بخشی از آن که مسئول رسیدگی به رویداد است.
- محیط[۱۱]: سیستم می تواند در حالت های مختلف عملیاتی مانند حالت عادی[۱۲] ، اضطراری[۱۳]، اوج بار[۱۴] یا اضافه بار[۱۵] باشد.

---

[1] periodic
[2] Stochastic
[3] Sporadic
[4] Latency
[5] Deadlines in processing
[6] The throughput of the system
[7] Jitter
[8] Source of stimulus
[9] Stimulus
[10] Artifact
[11] Enviroment
[12] Normal
[13] Emergency
[14] Peak Load
[15] OverLoad

- پاسخ[1] : سیستم در پاسخ رسیدن یک رویداد با تخصیص منابع به آن پاسخ می دهد که ممکن است به تغییر محیط نیز منجر شود.
- اندازه گیری پاسخ[2]: معیار های سنجش کارایی سیستم پیش از این بررسی شدند.برای بررسی کارایی سیستم می توان به تاخیر،توان عملیاتی سیستم و مهلت های پردازشی اشاره کرد.

۵ـ۳ تاکتیک ها در کارایی

هدف تاکتیک ها در کارایی تولید پاسخ یک رویداد با توجه به محدودیت های زمانی حاکم بر کارایی است. زمان پاسخ سیستم عموما به زمان پردازش[3] و یا زمان مسدود شدن [4] ختم می شود.

پردازش یک رویداد به منابع مختلفی نیاز دارد و توسط چندین مولفه[5] سیستم، نیاز به رسیدگی دارد. مجموع زمان‌هایی که مولفه‌های مختلف سیستم با به کارگیری منابع به یک رویداد رسیدگی می کنند، زمان پردازشی رویداد به حساب می آید. در کنار زمان پردازش، زمان مسدود شدن نیز نقش مهمی در مجموع زمان تاخیر یک رویداد دارد؛ پردازش یک رویداد ممکن است به دلیل نیاز محاسباتی بخشی از پردازش به پردازشی دیگر و یا به دلیل در دسترس نبودن یک منبع مسدود شود.برای مثال، ممکن است زیرسیستم جستجوی نزدیک ترین راننده‌ها در برنامه‌ی تاکسی‌برخط منتظر دریافت جزئیات نقشه از زیرسیستم نقشه باشد.

تاکتیک هایی که در کارایی وجود دارند به دو دسته ی زیر تقسیم می شوند:

- کنترل تقاضای منابع[6] : این دسته از تاکتیک‌ها در سمت تقاضا فعالیت می کنند و سعی می کنند تا میزان تقاضا برای منابع را کاهش دهند.
- مدیریت منابع[7] : این دسته از تاکتیک‌ها سعی دارند تا کارها را با بازدهی بیشتری انجام دهند تا مصرف منابع از این طریق کاهش یابد.

در ادامه به بررسی بیشتر این تاکتیک ها پرداخته می شود.

۵ـ۳ـ۱ کنترل درخواست منابع

یکی از راه های افزایش کارایی در حوزه نرم‌افزارهای حمل و نقل درون شهری، مدیریت دقیق تقاضای منابع است. این کار می تواند با کاهش تعداد رویدادهای پردازش شده با اعمال نرخ نمونه برداری یا با محدود کردن

[1] Response
[2] Response Measure
[3] Processing time
[4] Blocking time
[5] Component
[6] Control resource demand
[7] Manage resources

سرعت پاسخگویی سیستم به رویدادها انجام شود.

- مدیریت نمونه برداری: با کاهش نرخ نمونه برداری از محیط هر چند مقداری از دقت کاسته خواهد شد اما میزان تقاضا رو می توان به مقدار خوبی کاهش داد. زمانی که جریان سازگار داده[1] اهمیت بیشتری داشته باشد چشم پوشی از مقداری دقت می تواند راه حل خوبی برای کاهش تقاضا به سیستم باشد.برای مثال، در زیرسیستم نقشهي برنامه ی تاکسی‌رانی برخط، ممکن است نرخ به روزرسانی محل کنونی خودرو را کاهش دهیم تا کارایی سرور در زمان هایی که بار زیادی وجود دارد، افزایش یابد.

- محدودیت پاسخ به رویداد ها: زمانی که رویدادهای گسسته با نرخ بالایی به سیستم می‌رسند، باید در صف منتظر شروع پردازش بمانند. به دلیل گسسته بودن این رویداد ها امکان کاهش نرخ نمونه برداری وجود ندارد، در چنین شرایطی سیستم می تواند انتخاب کند که با توجه به شرایطی نظیر اندازه‌ی صف و یا زمانی که تاکنون رویداد در پردازش به خود اختصاص داده است، حداکثر زمانی که یک رویداد می تواند پردازش بگیرد را محمدود کند. با محدود کردن حداکثر زمان پردازشی که یک رویداد می تواند به خود اختصاص دهد، سیستم قادر خواهد بود تا رویداد های قابل پیش‌بینی بیشتری را تضمین کند. در این روش زمانی که تصمیم گرفته می‌شود رویدادی رها[2] شود باید سیاستی برای مدیریت این وضعیت از پیش تعیین شود: آیا رویداد های رها شده ثبت می‌شود و یا به راحتی نادیده گرفته خواد شد؟ آیا سایر سیستم‌ها و کاربران از رها سازی رویداد مطلع خواهند شد؟ در این زمینه می توان به مواقعی اشاره کرد که سیستم زمان زیادی را برای پیدا کردن سفرهای مشابه یک درخواستدر سیستم اشتراک خودرو صرف می کند و در چنین حالتی این تاکتیک توصیه می کند تا سرور این درخواست را رها کند تا با آزاد سازی منابع و پاسخ به درخواست های دیگر کارایی افزایش یابد.

- اولویت‌بندی رویدادها: اگر منابع کافی برای پاسخ دهی به تمامی رویدادها در اختیار سیستم قرار ندارد و یا اگر همه ی رویدادها از اهمیت یکسانی برخوردار نیستند استفاده از یک الگوریتم اولویت بندی می‌تواند تاثیر خوبی در کارایی سیستم داشته باشد.برای مثال، در برنامه‌ی یافتن پارکینگ های درون‌شهری سیستم می تواند درخواست مشتریان را بر اساس زمان پردازشی مورد نیاز اولویت‌دهی کند.

- کاهش سربار: هر چند وجود واسطه‌ها در افزایش اصلاح‌پذیری[3] از اهمیت بالایی برخوردار است اما حذف این واسطه ها می تواند منجر به افزایش کارایی سیستم شود. معماری در این مساله با تقابل دو

[1] Consistent Stream
[2] Drop
[3] Modifiability

نیاز کارایی و اصلاح‌پذیری دست و پنجه نرم می کند و باید بهترین تصمیم با توجه به نیازها و اهداف سیستم اخذ شود. برای مثال معماری می تواند بجای استفاده از چندین سیستم برای پاسخ به درخواست سفر توسط کاربر در برنامه‌ی تاکسی‌برخط از یک زیرسیستم یکپارچه استفاده کند و با حذف واسطه ها کارایی را در مجموع افزایش دهد؛هرچند همان طور که بیان شد انجام چنین کاری سبب کاهش نیاز کیفی اصلاح‌پذیری خواهد شد.

- زمان اجرای محدود: محدودیتی در میزان استفاده از زمان اجرا برای پاسخگویی به یک رویداد تعیین کنید. برای الگوریتم های تکراری و وابسته به داده ها ، محدود کردن تعداد تکرارها روشی برای محدود کردن زمان پردازش است که هر چند می تواند به پاسخی با دقت کمتر ختم شود اما در افزایش کارایی تاثیر مثبتی دارد. برای مثال، در برنامه ی پارکینگ های درون شهری، معماری می‌تواند زمانی که سرور برای پیدا کردن نزدیک‌ترین مسیر تا یک پارکینگ مشخص صرف می کند را با از دست دادن مقداری قابل چشم پوشی دقت، محدود کند.
- افزایش بازدهی منابع: بهبود الگوریتم های مورد استفاده در مناطق بحرانی تاخیر را کاهش می دهد.

۵-۳-۲ مدیریت منابع

هر چند ممکن است کنترل تقاضا برای منابع همواره ممکن نباشد اما مدیریت منابع همیشه در اختیار سیستم قرار دارد. گاهی می توان برای انجام یک پردازش از منابع مختلف استفاده کرد به عنوان مثال می توان در یک سیستم داده های میانی را در حافظه‌ی نهان[1] نگهداری کرد و یا از حافظه های دیگر سیستم به منظور ذخیره سازی آن ها استفاده کنید. در ادامه چندین روش مدیریت منابع سیستم مطرح می شوند:

- افزایش تعداد منابع: افزایش کمیت و کیفیت منابع همواره یکی از راه های کاهش تاخیر رویداد هاست که البته دغدغه های مالی را بر سر راه خود دارد.
- استفاده از هم‌زمانی[2]: اگر درخواست ها قابلیت پردازش موازی را داشته باشند، استفاده از هم‌زمانی و پردازش موازی همراه با الگوریتم‌هایی جهت مدیریت پردازش رویدادها بر نخ های پردازشی متفاوت به جهت حفظ توان عملیاتی و عدالت می تواند کارایی سیستم را به مقدار خوبی افزایش دهد. برای مثال درخواست‌های کاربرهای یک سرویس تاکسیرانی آنلاین تا حد خوبی از یک دیگر مجزا هستند و می‌توان به صورت همزمان انجام شود.

[1] Cache
[2] Concurrency

- حفظ چندین نسخه از محاسبات: وجود چندین سیستم مشابه برای پاسخ به رویداد ها، در زمانی که چندین رویداد هم‌زمان با یکدیگر رخ می دهند می تواند کارایی سیستم را تا حد زیادی افزایش دهد.برای مثال وجود چنین سرور در معماری های کلاینت_سرور[1] مانند آن‌چه که در سرویس تاکسیرانی آنلاین وجود دارد به همراه یک نرم افزار متعادل کننده ی بار[2] می تواند به کارایی سیستم در پاسخ به رویداد ها کمک زیادی کند.

- حفظ چندین نسخه از داده: تکثیر داده ها[3] شامل نگهداری نسخه های جداگانه از داده ها برای کاهش نزاع میان رویداد ها هنگام چندین دسترسی همزمان است. از آنجا که داده های تکثیر شده معمولاً یک کپی از داده های موجود است، حفظ کپی و هماهنگ سازی کپی ها به مسئولیتی تبدیل می شود که سیستم باید آن را بر عهده بگیرد. از این روش می‌توان در بیشتر نرم‌افزارهای این حوزه به خوبی استفاده کرد.

- محدودسازی اندازه ی صف: این روش که اصولا در کنار محدودیت پاسخ به رویدادها استفاده می شود، اقدام به محدود سازی اندازه صف ورودی رویدادها می کند. در این روش باید سیاست مشخصی در برابر رویداد هایی که به دلیل محدودیت اندازه ی صف پذیرش نمی شوند، اتخاذ شود. برای مثال در اپلیکیشن پارکینگ زمانی که پارکینگ‌ها پر است و تعداد درخواست‌ها زیاد است می‌توان از محدود سازی صف استفاده کرد.

- برنامه‌ریزی منابع: در این روش هدف درک ویژگی های استفاده از هر منبع و انتخاب استراتژی برنامه ریزی سازگار با آن است.

۴_۵ طراحی فهرست بازبینی برای کارائی

در ادامه فهرستی برای پشتیبانی از روند طراحی و تجزیه و تحلیل نیاز کارایی ارائه شده است.

۱_۴_۵ تخصیص مسئولیت ها

در زمان تخصیص مسئولیت‌ها باید [4] به موارد زیر توجه شود:

- مسئولیت‌های سیستم در شرایط مختلف نظیر بارسنگین [5] یا نیاز بحرانی به پاسخ یک رویداد تعیین شود.

---
[1] Client-Server
[2] Load Balancer
[3] Data Replication
[4] Allocation of Responsibilities
[5] Heavey Load

برای این مسئولیت ها، الزامات پردازش هر مسئولیت شناسایی شده و تعیین شود که آیا ممکن است موجب گلوگاه[1] های مختلف شود؟

- مسئولیت هایی که از وجود یک رشته کنترلی میان فرآیندها و یا مرزهای پردازشی حاصل می‌شوند، را شناسایی کنید.
- مسئولیت کنترل رشته های پردازشی در هنگام استفاده از روش های چندرشته ای[2].
- اطمینان حاصل شود سیستم با وجود مسئولیت‌هایی که به عهده‌اش گذاشته شده است، می تواند به نیاز های کارایی پاسخ دهد.

۵-۴-۲ مدل هماهنگی

عناصر سیستم را که باید هماهنگ شوند ـ مستقیم یا غیرمستقیم ـ تعیین شود و رویکردهای ارتباطی و هماهنگی انتخاب شود که موارد زیر را انجام می دهند:

- پشتیبانی از هم‌زمانی رویدادها و روش‌های اولویت بندی و استراتژی های برنامه‌ریزی
- اطمینان حاصل شود که می‌توان پاسخ کارایی مورد نیاز را ارائه داد.
- سیستم در صورت لزوم می‌تواند ورودهای دوره ای، تصادفی یا پراکنده را پذیرش کند.
- از خصوصیات مناسب مکانیسم های ارتباطی برخوردار باشند.

۵-۴-۳ مدل داده

مدل داده[3] ای آن بخش از داده ها که قرار است بار زیادی را متحمل شوند و در کارایی سیستم اثر گذار هستند مشخص شوند و موارد زیر در مورد آنها بررسی شود.

- آیا حفظ چندین نسخه از داده‌های کلیدی به نفع عملکرد است یا خیر؟
- آیا تقسیم[4] داده ها به نفع کارایی است؟
- آیا کاهش نیاز های پردازشی برای عملیات هایی که نیاز به تغییر در داده ها دارند امکان پذیر است؟
- آیا امکان افزایش منابع به جهت کاهش گلوگاه ها در هنگام تغییر در داده ها امکان پذیر است؟

---
[1] Bottleneck
[2] Multi-Threading
[3] Data Model
[4] Partitioning

۵-۴-۴ نقشه برداری در میان عناصر معماری

- در مواردی که بارگیری سنگین شبکه رخ می دهد، تعیین اینکه آیا قرارگیری همزمان برخی از اجزا باعث کاهش بارگزاری و بهبود کارایی کلی می شود؟
- اطمینان حاصل شود که مولفه ها با محاسبات سنگین تر به پردازنده هایی با بیشترین ظرفیت پردازش اختصاص داده شده اند.
- اطمینان حاصل شود استفاده از هم‌زمانی ممکن است و آیا تاثیر مثبت قابل توجه ای بر روی کارایی سیستم دارد؟
- تعیین شود که آیا انتخاب رشته های کنترل و مسئولیت های مربوط به آنها گلوگاه هایی را ایجاد می کند یا خیر؟

۵-۴-۵ مدیریت منابع

تعیین شود کدام منابع نقش حیاتی و مهمی در کارایی را ایفا می کنند. اطمینان حاصل شود این منابع در طول اجرا تحت شرایط عادی و بار کاری بالا تحت نظارت و مدیریت صحیح قرار می گیرند.

۵-۴-۶ زمان اتصال

موارد زیر برای مولفه هایی که پس از کامپایل [1] به سیستم اضافه می شوند،تعیین شود:

- زمان لازم برای اضافه شدن مولفه به سیستم
- میزان سربار اضافه ناشی از اضافه شدن دیر هنگام مولفه به سیستم

باید اطمینان حاصل شود که این دو سربار جریمه خیلی زیادی بر روی سیستم نمی گذارند.

۵-۴-۷ انتخاب فناوری

آیا انتخاب فناوری شما امکان تنظیم موارد زیر را دارد:

- سیاست برنامه ریزی
- اولویت ها
- سیاست های کاهش تقاضا

[1] Compile

- تخصیص بخشهایی از فناوری به پردازنده ها
- سایر موارد مربوط به کارایی

۵-۵ مطالعات موردی

در اینجا مطالعات موردی در ۳ حوزه حمل و نقل درون شهری برای نیاز کاریی انجام شده است.

۵-۵-۱ تاکسیرانی و حمل‌ونقل درون شهری

کارایی امروزه یکی از نیاز های مهم در هر نرم‌افزار است؛ برنامه هایی که کارایی لازم در مقایسه با رقبای خود دارا نیستند به راحتی بازار خود را به رقبا واگذار می کنند.کاربر انتظار دارد درخواست های او از سیستم در سریع ترین زمان ممکن پاسخ داده شود. همچنین در سیستم‌هایی مانند تاکسیرانی آنلاین اوبر، تمامی درخواست ها از اهمیت یکسانی دارا نیستند؛ سیستم باید با الویت دهی به درخواست هایی که محدودیت های سخت‌تری دارند کارایی سیستم را افزایش دهد [۴۱] .

در برنامه‌ی تاکسی‌آنلاین می‌توانیم با کاهش نرخ نمونه برداری تاکسی‌ها به درخواست های دیگر زیرسیستم ها منابع بیشتری را بدهیم. به عنوان نمونه، در برنامه ی ،MyWay به صورت میانگین جستجوی سفرهای درون شهری حداکثر باید ظرف ۵ ثانیه انجام شوند و همچنین زمانی که کیفیت اینترنت در وضعیت عادی قرار دارد اکثر ارتباط‌های برنامه با سرور باید ظرف ۲ ثانیه صورت پذیرند [۲۳].

سناریو: کاربر با وارد کردن مقصد خود به دنبال راننده می گردد.

- منبع محرک: کاربر نهایی
- محرک: رویداد جستجوی راننده بر اساس مقصد
- محصول:سیستم
- محیط: محیط اجرای عادی
- پاسخ: راننده به کاربر معرفی می‌شود یا پاسخ داده می‌شود که راننده‌ای به این منظور وجود ندارد.
- اندازه گیری پاسخ: اندازه‌گیری تاخیر تا پاسخ دادن به کاربر (انتظار می‌رود تا ۵ ثانیه پیدا کند)

۵-۵-۲ اشتراک خودرو و حمل‌ونقل درون شهری

در برنامه های اشتراک خودرو یکی از زیرسیستم های اصلی جستجوی افراد با برنامه ی سفر و ویژگی های مشخص مشابه است.الگوریتمی که این زیرسیستم استفاده می کند باید از کارایی بالایی برخوردار باشد و

همچنین باید بتواند هم‌زمان به درخواست هزاران کاربر پاسخ دهد.شرکت هایی که در این زمینه فعالیت می‌کنند باید با استفاده از پردازش های چندرشته ای[1] و تخصیص بهینه‌ي منابع به کارایی بالایی دست یابند. به عنوان مثال در سناریو عنوان شده سیستم پس از ۳۰ ثانیه باید نتایجی که به آن ها دست یافته است را به اطلاع کاربر نهایی برساند و منابع را برای پاسخ به سایر درخواست ها آزاد کند.

سناریو: کاربر با وارد کردن مقصد خود به دنبال همسفر می گردد.

- منبع محرک: کاربر نهایی
- محرک: رویداد جستجوی همسفر (راننده) بر اساس مقصد
- محصول:سیستم
- محیط: محیط اجرای عادی
- پاسخ: لیستی از همسفر ها به کاربر پیشنهاد می‌شود
- اندازه گیری پاسخ: اندازه‌گیری تاخیر تا پاسخ دادن به کاربر (انتظار می‌رود تا ۳۰ ثانیه پیدا کند)

۵-۵-۳ پارکینگ های عمومی و حمل‌ونقل درون شهری

یکی از دغدغه‌های مهم در برنامه‌های این حوزه تغییرات زیاد در وضعیت فضا های خالی در پارکینگ هاست؛ برنامه هایی که چند ده هزار یا بیشتر پارکینگ را پوشش می دهند ممکن است در هر لحظه نیاز به هزاران نوشتن و خواندن از پایگاه‌داده داشته باشند.هر تغییر وضعیت در پاکینگ ها باید به سرعت و بدون تداخل در قالب تراکنش های اتمیک[2] در پایگاه داده ثبت شوند. در چنین سیستم‌هایی خوب است که سیستم وضعیت پارکینگ هایی که بیشتر مورد استفاده قرار می‌گیرند را در حافظه هایی با سرعت بیشتر نسبت به سایر پارکینگ ها نگهداری کند.

سناریو: کاربر نهایی در یک شهر به جستجوی پارکینگ ها با ظرفیت خالی می گردد.

- منبع محرک: کاربر نهایی
- محرک: رویداد یافتن پارکینگ ها با ظرفیت خالی در سطح شهر
- محصول:سیستم
- محیط: محیط اجرای عادی

[1] Multi-Thread
[2] Atomic

- پاسخ: وضعیت پارکینگ ها با ظرفیت خالی توسط سیستم پردازش شده و به کاربر پاسخ داده می‌شود.
- اندازه گیری پاسخ: اندازه‌گیری تاخیر تا پاسخ دادن به کاربر (انتظار می‌رود تا ۵ ثانیه پیدا کند)

فصل ششم

امنیت

۶ـ۱ تعریف امنیت

امنیت [1] یکی از نیازهای غیر عملکردی سیستم است که بوسیله آن می‌توان توانایی سیستم برای محافظت داده و اطلاعات از اشخاصی که مجوز دسترسی ندارند را نشان داد. در صورت وجود امنیت در سیستم‌های حوزه حمل‌ونقل درون شهری ممکن است که یک دسترسی بدون مجوز بخواهد که به داده کاربران دسترسی پیدا کند و آن را تغییر دهد یا باعث شود که سرویس برای بقیه کاربرهای دارای مجوز در دسترس نباشد.

امنیت ۳ مشخصه اصلی دارد که در ذیل خلاصه شده است [۳۵]:

۱. محرمانگی: در این معیار داده و سرویس‌ها از دسترس دسترسی‌های غیر مجوز دار حفظ می‌شود: برای مثال در سرویس تاکسی آنلاین داده‌های کاربران تنها در اختیار افراد معتبر خواهد بود و هرکسی به آن دسترسی نخواهد داشت.

۲. اصالت: در این معیار اصالت داده‌ها تضمین می‌شود و دست‌خوش تغییرات نمی‌شود: برای مثال در سیستم پارکینگ اگر فضای خالی وجود دارد و نرم‌افزار خالی بودن را نشان می‌دهد این داده اصالت دارد و تنها در صورتی که آن فضا پر شود داده نیز نشان دهنده پر بودن آن است و توسط افراد غیر مجاز دست‌خوش

[1] Security

تغییرات نباید بشود.

۳. در دسترس‌پذیری: در این معیار سیستم برای افراد دارای مجوز در دسترس خواهد بود و برای مثال استفاده از حمله منع سرویس نباید باعث شود که به در دسترس‌پذیری سرویس لطمه وارد شود: برای مثال ممکن است که یک حمله کننده تعداد زیادی درخواست به سرویس اشتراک خودرو دهد و سرویس آن بدلیل بار زیاد از دسترس کاربران دیگر خارج شود.

همچنین مشخصه‌های دیگری را نیز می‌توان برای امنیت لحاظ کرد که زیر مجموعه معیارهای قبلی هستند. در ذیل به ۳ مورد از آن‌ها اشاره شده است.

۱. احراز هویت: هویت افرادی که با سیستم درگیر هستند را تایید می‌کند: برای مثال در سیستم تاکسی آنلاین با استفاده از نام کاربری و رمز عبور و شماره همراه به تایید کاربر پرداخته می‌شود.

۲. عدم پذیرش مسئولیت: تضمین می‌کند که فرستنده در آینده فرستادن پیام خود را انکار نمی‌کند: برای مثال در سامانه اشتراک خودرو بین افرادی که یک خودرو را به اشتراک می‌گذارند عدم پذیرش مسئولیت پیام‌های رد و بدل شده بوجود نیاید که این موضوع ممکن است در آینده باعث غش در توافق بین آن‌ها بینجامد.

۳. احراز اصالت: به کاربر مجوزهای لازم برای انجام وظیفه مربوطه را اعطا می‌کند: برای مثال در سامانه پارکینگ افراد احراز هویت شده مجوز لازم برای رزرو جای پارک را داشته باشند.

۲-۶ سناریوی عمومی امنیت

سناریوی عمومی امنیت به صورت زیر است:

- منبع محرک[1] : انسان یا سیستمی که تشخیص داده شده است حال چه به صورت صحیح چه ناصحیح. برای مثال یک حمله‌کننده انسان خارج از شرکت سرویس تاکسی آنلاین
- محرک[2] : یک درخواست بدون مجوز برای نمایش داده، تغییر، حذف یا دسترسی به سرویس ها: یک درخواست بدون مجوز توسط یک فرد تایید نشده برای حذف رانندگان مجاز از سامانه اشتراک خودرو
- محصول[3]: سرویس‌های سیستم، داده‌های درون سیستم، مولفه‌ها و منابع سیستم: برای مثال تمام منابع موجود در سیستم‌های نرم‌افزاری حوزه حمل و نقل درون شهری

[1] Source of stimulus
[2] Stimulus
[3] Artifact

- محیط[1]: سیستم که می‌تواند به صورت برون خط یا برخط باشد، به شبکه متصل باشد یا نباشد، پشت دیواره آتش باشد یا نباشد: برای مثال سیستم‌های برخط حوزه حمل و نقل درون شهری که پشت دیواره آتش هستند

- پاسخ[2] : بسته به نوع محرک و محصول پاسخ‌هایی بدین شکل تولید خواهد شد: داده و سرویس از دسترسی بدون مجوز محافظت شود، تغییر نیابد. منابع و سرویس‌ها در دسترس باشد و ... .همچنین تمامی فعالیت‌ها ضبط شود تا بعدا بتوان در صورت بروز خرابی علت را پیگیری کرد.

- اندازه گیری پاسخ[3]: می‌توان معیارهای متفاوتی در نظر گرفت: چه مقدار طول کشیده است تا سیستم بعد از یک حمله موفق دوباره بازیابی شود، چه مقدار از مولفه‌های سیستم یا داده مورد خرابی قرار گرفته است یا چقدر طول کشیده است تا متوجه حمله شود.

برای دست یابی به هر نیازمندی مجموعه ای از تکنیک ها را به کار می بندیم که در مورد امنیت می‌توان به ۴ دسته شناخت، مقاومت، عکس‌العمل و بازیابی اشاره کرد.

برای شناخت حمله می‌توان از مقایسه کردن ترافیک شبکه یا سرویس یا مقایسه پترن‌های سیستم و پیدا کردن رفتارهای غیر معمول بهره برد. همچنین از روی چنین پترن‌هایی می‌توان به حمله منع سرویس نیز پی برد یا از روی دیرکرد پیام به آن مشکوک شد. برای چک کردن اصالت داده‌ها نیز می‌توان از روش‌های چک کردن اصالت داده مانند کدهای خطا[4] و توابع هش[5]استفاده کرد.

به منظور مقاومت می‌توان منابع ورودی به سیستم را شناسایی کرد و مطمئن شد که آن منبع دقیقا همان کس یا چیزی است که باید باشد. به منابع مطمئن مجوزهای لازم برای تغییر و دسترسی به داده اعطا شود و تا حد ممکن دسترسی به منابع محدود شود. همچنین منابع را از نظر فیزیکی تا جای ممکن ایزوله باشد تا اگر یکی مورد حمله قرار گرفت دیگری را تحت تاثیر قرار ندهد. همچنین می‌تواند از مندهای رمزنگاری بر روی داده و ارتباط استفاده کرد و کاربر را مجبور کرد که تنظیمات خود را از حالت معمول خارج کرده و آن را تغییر دهد.

عکس العمل مناسب در برابر حمله شامل اقدامات باطل کردن دسترسی حمله به منابع حتی برای کاربرهای مجاز است تا زمانی که حمله رفع شود. اگر تلاش‌ها مکرری برای دسترسی موفقیت آمیز نبود، دسترسی محدود و قفل شود. به اپراتورها و افراد مناسب در زمان اتفاق افتادن امری مشکوک یا تشخیص حمله اطلاع داده شود.

به منظور بازیابی مناسب از تاکتیک‌های مربوط به دسترس‌پذیری استاده شود تا منابع بازیابی شوند و از تمامی

---

[1] Enviroment
[2] Response
[3] Response Measure
[4] Checksum
[5] Hash

اتفاقات رکورد تهیه شود تا بعدا بتوان حمله کننده را تشخیص داد و آن را ردیابی کرد.

به منظور بهره بری از این تاکتیک‌ها می‌توان با توجه به هریک از پروژه‌های تاکسی آنلاین، اشتراک‌گذاری خودرو و مدیریت پارکینگ یک فهرست بازبینی تنظیم کرد. این موضوع به بهبود نیاز امنیت پاسخ می‌دهد.

۶ـ۳ طراحی فهرست بازبینی برای امنیت

در ادامه فهرستی برای پشتیبانی از روند طراحی و تجزیه و تحلیل برای معیار امنیت ارائه شده است.

۶ـ۳ـ۱ تخصیص مسئولیت ها

در این گام نیاز است که تعیین شود چه مسئولیت‌هایی از سیستم نیاز است که امن شود. برای این مسئولیت‌ها نیاز است که مطمئن شد مسئولیت‌های اضافه تخصیص یافته است:

- منبع محرک شناسایی شود.
- منبع محرک احراز اصالت شود.
- منبع محرک احراز هویت شود.
- مجوزهای لازم برای دسترسی به داده و سرویس‌ها اعطا شود.
- داده‌ها رمزگذاری شود.
- اگر دردسترس‌پذیری کاهش یافت تشخیص داده شود و اقدام مناسب مانند محدود کردن دسترسی و مطلع کردن اپراتور صورت گیرد.
- بعد از حمله بازیابی صورت گیرد.
- کدهای خطا و مقادیر هش تایید شود.

۶ـ۳ـ۲ مدل هماهنگی

در این گام نیاز است که مکانیزم‌های لازم برای ارتباط با بخش‌های دیگر تعیین شود. همچنین باید مطمئن شد که این این مکانیزم‌ها از قابلیت احراز هویت و اصالت بهره می‌برند و داده به صورت رمزشده میان بخش‌های مختلف انتقال پیدا می‌کند. در نهایت باید مطمئن شد که مکانیزم‌هایی برای مانیتور کردن و تشخیص تقاضا منابع زیاد وجود دارد و در صورت بروز تقاضا آن‌ها را محدود کرد.

۶-۳-۳ مدل داده

در این بخش حساسیت فیلدهای مختلف داده تعیین شود و موارد زیر اجرا گردد:

- باید مطمئن شد که داده‌های با سطح حساسیت متفاوت به صورت جدا ذخیره شده است.
- باید مطمئن شد که برای دسترسی به داده‌های با سطح حساسیت متفاوت، دسترسی‌هایی با سطح متفاوت نیاز است.
- باید مطمئن شد که درخواست دسترسی به داده‌های حساس ضبط می‌شود و از داده‌های ضبط آن نیز به طور مناسب محافظت می‌شود.
- باید مطمئن شد که داده‌ها به شکل مناسب محافظت شده است و کلیدهای آن نیز جدا از داده‌ها ذخیره و محافظت می‌شوند.
- باید مطمئن شد که اگر داده ها به طور نامناسب تغییر پیدا کرد قابل بازیابی باشد.

۶-۳-۴ نگاشت در میان عناصر معماری

در این قسمت تعیین می‌شود که چگونه نگاشت جایگزین اجزا معماری ممکن است که نحوه خواندن و نوشتن و تغییر دادگان و تنظیم دسترسی‌ها و کاهش دردسترس پذیری را تحت تاثیر قرار دهد. همچنین تعیین می‌شود که چگونه نقشه‌برداری ضبط داده‌های دسترسی به دادگان و سرویس‌ها را تحت تاثیر قرار می‌دهد یا تقاضاهای زیاد سیستم را تشخیص می‌دهد. برای هر یک از این نقشه برداری‌ها باید مطمئن شد که مسئولیت‌های عنوان شده در بخش تخصیص مسئولیت وجود دارد.

۶-۳-۵ مدیریت منابع

در این بخش نیاز است که منابعی از سیستم که نیاز است شناسایی، مانیتور، احراز اصالت و دسترسی خاص به منابع داشته باشند تعیین شود. همچنین نیاز است که منابعی که نیاز است تا احراز هویت، دادن مجوز، اطلاع افراد مناسب، رکورد دسترسی به داده‌ها، رمزگذاری داده‌ها و تشخیص تقاضای زیاد و محدود کردن دسترسی داشته باشند تعیین شود. برای این منابع نیاز است که تعیین شود که افراد خاص سازمان چقدر به منابع مخصوصا منابع حساس دسترسی داشته باشند. چگونه منابع مانیتور شوند و مطمئن شد که بخش‌های لازم عملیات‌های امنیتی مورد نیاز را انجام می‌دهند. همچنین باید مطمئن شد که از منابع مشترک برای انتقال داده‌های حساس از یک کاربری با حق دسترسی به یک کاربری بدون حق دسترسی استفاده نمی‌شود.

۶-۳-۶ زمان اتصال

مواردی را در نظر بگیرید که یک مولفه دیررس قابل اعتماد نباشد. به این منظور باید از مکانیزم‌های مناسب برای مدیریت و اعتبارسنجی آن استفاده کرد. دسترسی آن مولفه می‌تواند به سرویس‌ها و داده‌ها مسدود شود و دسترسی‌های ضبط شود. همچنین دادگان نیز رمزگذاری و کلید آن‌ها در جایی که اجزا دیررس دسترسی ندارند ذخیره شود. تمام این موارد انجام شود تا مولفه ارزیابی شود و صلاحیت آن اثبات شود.

۶-۳-۷ انتخاب فناوری

از فناوری‌هایی که می‌توان بوسیله آن از دادها محافظت نمود و حق دسترسی‌ها را به خوبی مدیریت کرد و قابلیت احراز هویت کاربران را فراهم نمود باید استفاده کرد. همچنین باید دقت داشت که تکنولوژی مورد انتخاب تاکتیک‌های مورد نیاز برای بحث امنیت را پشتیبانی کند.

۶-۴ مطالعات موردی

در اینجا مطالعات موردی در ۳ حوزه حمل و نقل درون شهری برای نیاز امنیت انجام شده است. همانگونه که قبلا بیان شد باید بسته یه قسمت‌های حساس پروژه از رمزگذاری داده‌ها و ارتباط، شناسایی، احراز هویت و اصالت، کاهش دسترسی‌پذیری، بازیابی، ذخیره مناسب کلیدها و کدهای هش و خطا استفاده کرد تا امنیت مورد نیاز تامین شود.

در مستندات پروژه MyWay اعلام شده است که اطلاعات کاربر ناشناس می‌ماند، پروفایل‌های کاربر در سرور ذخیره می‌شود و حریم شخصی و مدیریت مناسب انجام می‌شود. ذخیره پروفایل‌های کاربر با قوانین محلی سازگار است، ارتباطات برای داده‌های حساس به صورت رمز شده است. قابلیت ردیابی برای عملیات‌های حساس انجام شده توسط کاربران وجود دارد، به خصوص برای ادمین‌ها. همچنین این اپلیکیشن صحت داده و مجوزهای لازم برای دسترسی به داده را تضمین می‌کند. ادمین‌ها دسترسی لازم برای مشاهده و تغییر داده‌های درونی را نیز دارند. در آخر باید گفت که کاربران در این اپلکیشن احراز هویت می‌شود و سطح دسترسی‌ها و مجوزهای مختلفی میان نقش‌های اپلیکیشن وجود دارد. کاربر احراز هویت شده می‌تواند به نقش‌های مختلفی نگاشت شود [۲۳]. همچنین با بررسی‌ای که از برنامه Uber ، ParkWhiz و Waze-Carpool به عمل آمد مشخص شد که این اپلیکیشن‌ها از قابلیت‌های احراز هویت با استفاده از نام کاربری و رمز عبور، نگه داشتن نشست کاربر و دنبال کردن فعالیت‌های کاربر، ضبط رکوردهای درخواست کاربر به سرور، استفاده از متدهای رمزنگاری مناسب، نگه‌داشتن تاریخچه مناسب از دادگان، نسبت دادن برخی توابع خاص تنها به بعضی از ماژول‌ها، محدود کردن ارتباط تنها در قسمت‌هایی از برنامه، چک کردن صحت دادگان برای متغیرهای حیاتی،

دنبال کردن خطاها، استفاده از ارتباط امن ssl برخوردار هستند.

در این رابطه تمام موارد مورد مطالعه می‌توان سناریوهای زیر را نوشت:

سناریو: اگر یک اتکر بخواهد به جای یک کاربر دیگر اقدام به درخواست سروی کند باید از این درخواست بدون مجوز جلوگیری شود.

- منبع محرک: اتکر
- محرک: تلاش بدون مجوز برای درخواست سرویس (به جای یک کاربری دیگر)
- محصول: سرویس
- محیط: سیستم آنلاین
- پاسخ: جلوگیری از درخواست بدون مجوز وضبط تغییرات و مطلع کردن کاربر از این که آیا این تغییر با درخواست شما صورت گرفته است یا خیر
- اندازه گیری پاسخ: مقدار زمانی که طول کشیده شده این حمله شناسایی شود و چه مقدار آسیب وارد شده است.

سناریو: ممکن است اتکر سعی در انجام حمله منع سرویس کند تا از دردسترس‌پذیری سرویس تاکسی رانی آنلاین بکاهد.

سناریو: هکر سعی می‌کند با ارسال تعداد زیادی درخواست (حمله‌ی منع سرویس) سرویس ها را از دسترس خارج کند.

- منبع محرک: اتکر
- محرک: حمله منع سرویس به جهت کاهش دسترس‌پذیری
- محصول: سرویس‌های سیستم
- محیط: سیستم آنلاین
- پاسخ: داده و سرویس‌ها برای کاربران مجاز در دسترس قرار بگیرد و به کمک دیواره آتش اتکر شناسایی و دسترسی آن محدود شود.
- اندازه گیری پاسخ: مقدار زمانی که سیستم نسبت به حمله آسیب‌پذیر بوده است (چقدر طول کشیده تا به حالت نرمال برگردد)

فصل هفتم

قابلیت آزمون

۷ـ۱ تعریف قابلیت آزمون

طبق تخمین ها بین ۱۵ تا ۲۵ درصد هزینه تولید یک سیستم به آزمون سیستم اختصاص دارد و بعضی مدیران هزینه آن را تا ۵۰ درصد برآورد می‌کنند [۱۱]. قابلیت آزمون[1] به سهولت ساخت نرم افزار برای نشان دادن عیب های خود از طریق آزمون اشاره دارد. به طور خاص ، قابلیت آزمون احتمال آن است که اگر برنامه دارای خطایی باشد، در آزمون خطا خود را نشان دهد. به صورت شهودی، اگر یک سیستم به راحتی خطا های خود را آشکار سازد، دارای قابلیت آزمون است

معیارهای پاسخ دهی برای قابلیت آزمون به میزان موثر بودن آزمون ها در کشف عیب ها و مدت زمان انجام آزمایش ها تا رسیدن به حد مطلوبی از پوشش[2] مربوط می شوند.

۷ـ۲ سناریوی عمومی قابلیت آزمون

سناریوی عمومی قابلیت استفاده به صورت زیر است:

[1] Testability
[2] Coverage

- منبع محرک[1] : هر کسی که برای سیستم آزمون طراحی می کند می تواند جز محرک های سناریو عمومی باشد. آزمون‌های طراحی شده می توانند به صورت دستی یا اتوماتیک اجرا شوند.
- محرک[2] : اجرای مجموعه ی خاصی از آزمون‌ها که می تواند به یکی از دلایل اضافه شدن مولفه ای جدید یا تحویل به مشریان صورت پذیرفته باشد.
- محصول[3]: سیستم یا بخشی از آن که مورد آزمون قرار می گیرد.
- محیط[4]:: زمان طراحی، زمان توسعه، زمان کامپایل، زمان ادغام[5]، زمان استقرار[6] ، زمان اجرا
- پاسخ[7] : پاسخ سیستم می تواند زیر مجموعه ای از موارد زیر باشد.
  - اجرای مجموعه ی آزمون ها و دریافت پاسخ آن ها
  - جمع آوری فعالیت هایی که به خطا در آزمون منجر شده اند
  - کنترل و نظارت بر وضعیت سیستم
- اندازه گیری پاسخ[8]: یک یا چند مورد از این موارد را می‌توان به عنوان اندازه گیری استفاده نمود.
  - تلاش برای یافتن یک عیب یا کلاسی از عیب‌ها
  - تلاش برای دستیابی به درصد معینی از پوشش فضا[9]
  - احتمال بروز خطا در آزمون بعدی
  - زمان انجام آزمون ها
  - طول طولانی ترین زنجیره وابستگی در آزمون
  - مدت زمان آماده سازی محیط آزمون

۳ـ۷ تاکتیک ها در قابلیت آزمون

هدف تاکتیک های قابلیت آزمون آسان‌تر ساختن انجام آزمون ها پس از پایان فرآیند توسعه اولیه سیستم است.روش هایی که برای افزایش قابلیت آزمون وجود دارند در دو دسته از تاکتیک های قابلیت آزمون قرار می

[1] Source of stimulus
[2] Stimulus
[3] Artifact
[4] Enviroment
[5] Integration Time
[6] Deployment Time
[7] Response
[8] Response Measure
[9] Space Coverage

گیرند؛ دسته ی اول به قابلیت کنترل و نظارت بر سیستم می افزایند و از این طریق انجام آزمون ها را آسان تر می سازند و دسته ی دیگر بر محدود کردن پیچیدگی های سیستم تمرکز می کنند.

۷ـ۳ـ۱ کنترل و نظارت بر وضعیت سیستم

ساده ترین شکل کنترل و نظارت بر یک سیستم نرم افزاری با مجموعه ای از ورودی ها است.اجازه می دهید سیستم کار خود با ورودی های داده شده را انجام دهد و سپس خروجی آن را مشاهده می کنید.به صورت خاص روش هایی که در این دسته بندی وجود دارند شامل موارد زیر هستند:

- رابط های تخصصی[1] : برخورداری از یک رابط این قابلیت را می دهد که در زمان آزمون نرم افزار یا اجرا عادی به مقادیر متغیر ها در برنامه دسترسی داشته باشیم.
- ضبط/پخش [2] : بازایجاد حالتی که خطا در آن رخ می دهد در بسیاری از مواقع دشوار است.ضبط حالت هنگام فراخوانی یک رابط باعث می شود تا از این حالت برای ایجاد مجدد خطا استفاده شود.
- محلی سازی ذخیره سازی [3] : برای راه اندازی یک سیستم یا زیرسیستم از یک حالت آغازین تصادفی به منظور آزمون،بهترین روش این است که آن حالت در یک مکان واحد ذخیره شود.
- منابع داده انتزاعی [4] : استفاده از منابع داده ای انتزاعی این امکان را به شما می دهد تا داده های آزمون را راحت تر جایگزین کنید.به عنوان مثال ، اگر پایگاه داده ای از داده های قابل آزمون را در اختیار داشته باشید، می توانید معماری خود را طوری طراحی کنید که به راحتی در مواقع آزمون سیستم از پایگاه داده ی مخصوص آزمون استفاده کند.
- جعبه شنی [5] : منظور از جعبه شنی این است که نمونه ای از سیستم را از دنیای واقعی جدا کنید تا بتوانید آزمونی را انجام دهید بدون آن که نگرانی ای در مورد عواقب آزمون شما را نگران کند.با استفاده از جعبه شنی می توانید نسخه ای از منابعی را که رفتار آن تحت کنترل شماست،بسازید.
- ادعاهای قابل اجرا [6] : با استفاده از این تاکتیک ،ادعاها معمولا به صورت دستی کد زده می شوند و در مکان های دلخواه قرار می گیرند تا نشان دهند چه زمانی و در کجا برنامه در وضعیت معیوبی قرار دارد.

---

[1] Specialized interfaces
[2] Record/Playback
[3] Localize state storage
[4] Abstract data sources
[5] Sandbox
[6] Executable assertions

۷ـ۳ـ۲ محدودسازی پیچیدگی های سیستم

هر چقدر یک نرم افزار پیچیده‌تر باشد، آزمون آن سخت تر خواهد بود زیرا بنا بر تعریف پیچیدگی ، فضای حالت عملیاتی یک برنامه ی پیچیده بسیار بزرگ است و ایجاد مجدد یک حالت خاص در یک فضای حالت بزرگ دشوارتر از انجام این کار در فضای حالت کوچک است.به طور خاص دو روش زیر به محدودسازی پیچیدگی نرم افزار کمک می کنند.

- محدودسازی پیچیدگی ساختاری : این تاکتیک شامل اجتناب یا حل وابستگی های حلقوی میان مولفه های سیستم ، جداسازی و کپسول سازی[1] وابستگی ها به محیط خارجی و کاهش وابستگی بین اجزا به طور کلی است.در این روش می توانید عمق درخت ارث بری [2] و تعداد فرزندان یک کلاس را محدود کنید.
- دوری از عدم قطعیت [3] : این تاکتیک شامل یافتن تمام منابع عدم قطعیت، مانند موازی کاری غیرقانونی و حذف هرچه بیشتر آن‌ها است.

۷ـ۴ طراحی فهرست بازبینی برای قابلیت آزمون

۷ـ۴ـ۱ تخصیص مسئولیت ها

در مورد تخصیص مسئولیت ها موارد زیر باید بررسی شوند:

- تعیین شود مهم ترین مسئولیت های سیستم چه مسئولیت هایی هستند زیرا این مسئولیت ها باید به دقت مورد آزمون قرار گیرند.
- اطمینان حاصل شود مولفه هایی در سیستم وظایف زیر را بر عهده دارند.
  - اجرای تست‌ها و دریافت نتایج حاصل از آن‌ها
  - ذخیره سازی لاگ[4] ها رفتار های غیر قابل پیش‌بینی سیستم و لاگ خطا های رخ داده در سیستم
  - کنترل و نظارت بر حالت های مرتبط با سیستم به جهت آزمون سیستم
- اطمینان حاصل شود که تخصیص عملکرد ها انسجام بالا ، اتصال[5] کم ، جدایی شدید نگرانی ها[6] و پیچیدگی ساختاری کم را فراهم می کند.

---
[1] Encapsulation
[2] Inheritence
[3] Nondeterminism
[4] Logs
[5] coupling
[6] separation of concerns

۷-۴-۲ مدل هماهنگی

از مکانیسم های هماهنگی و ارتباط سیستم اطمینان حاصل کنید:

- پشتیبانی از اجرای تست ها و دریافت پاسخ آن ها در یک یا میان چندین سیستم
- پشتیبانی از دریافت تاریخچه خطا های رخ داد در یک سیستم و یا میان چندین سیستم
- پشتیبانی از تزریق[1] و نظارت[2] بر وضعیت در کانال های ارتباطی برای استفاده در آزمون ها،درون یک سیستم یا بین سیستم ها

۷-۴-۳ مدل داده

در بخش باید انتزاعات اصلی داده را که باید برای اطمینان از عملکرد صحیح سیستم مورد آزمون قرار گیرند را تعیین کنید.

- اطمینان حاصل شود که امکان گرفتن مقادیر نمونه های این انتزاعات داده وجود دارد.
- اطمینان حاصل شود که می توان مقادیر نمونه های این انتزاعات داده را هنگام تزریق حالت به سیستم تنظیم کرد؛بنابرین می توانیم حالت سیستم که منجر به خطا شده است،دوباره قابل ایجاد است.
- اطمینان حاصل شود که هر گونه تغییر در این داده های انتزاعی ممکن و قابل آزمون است.

۷-۴-۴ نقشه برداری در میان عناصر معماری

نحوه آزمون نگاشت های احتمالی عناصر معماری (به ویژه نگاشت فرآیندها به پردازنده ها ،رشته ها برای پردازش ها و ماژول ها به اجزا) را تعیین کنید تا پاسخ آزمون مورد نظر حاصل شود و وجود وضعیت رقابتی[3] مشخص شود.

۷-۴-۵ مدیریت منابع

در این بخش باید از وجود منابع کافی جهت اجرای آزمون های اطمینان حاصل کنید و همچنین مطمئن شوید محیطی که آزمون ها در آن به اجر در می آیند نمود محیط واقعی که سیستم در آن فعالیت می کند است.

در بحث مدیریت منابع باید به موارد زیر توجه شود:

- محدودیت منابع آزمون

---

[1] Injection
[2] Monitoring
[3] Race Condition

- ثبت دقیق مصرف منابع در هنگام رویداد ها علی الخصوص خطا ها
- وضع محدودیت های جدید بر روی منابع برای آزمون شرایط خاص
- ایجاد منابع مجازی با هدف آزمون

۷ـ۴ـ۶ زمان اتصال

در این بخش باید اطمینان حاصل شود مولفه هایی که پس از زمان کامپایل دیرهنگام به سیستم متصل می‌شوند از قابلیت آزمون برخوردار هستند.

۷ـ۴ـ۷ انتخاب فناوری

انتخاب فناوری باید به گونه‌ای باشد که سناریوهای قابلیت آزمون را بر روی سیستم بتوان اجرا کرد.

۷ـ۵ مطالعات موردی

هر ۳ حوزه مطرح شده دارای نرم‌افزارهای آنلاین به صورت حداقل موبایل و وب هستند. در هنگام توسعه، توسعه‌دهندگان این برنامه‌ها از یونیت‌تست[1] برای پوشش آزمون‌های هر مولفه نرم‌افزار استفاده می‌کنند که این پروسه تولید نرم‌افزار بر اساس معیارهای TDD [2] طراحی و انجام می‌شود.

همچنین برای پوشش کیفیت تست‌ها از نرم‌افزارهای سنجش کد مانند سونارکیوب [3] همراه با جنکینز [4] و خط لوله‌های [5] تولید نرم‌افزار استفاده می‌شود. همچنین برای آزمون عملکرد برنامه‌های تحت وب نیز از سلنیوم [6] استفاده می‌شود.

همچنین موارد دیگری از تست نیز وجود دارد که به نام تست A/B معروف هستند و فیدبک کاربر را نسبت به تغییرات جدید می‌سنجند معمولا هر آپدیت شرکت‌هایی مثل اوبر قبل از عرضه نهایی به همه کاربران از این نوع آزمون نیز عبور می‌کند [۴۲].

سناریوهای زیر را می‌توان در نظر گرفت:

سناریو: کاربران نهایی که در تست A/B اخرین تغییر اپلکیشن تاکسی‌رانی آنلاین شرکت کردند و شکست آپدیت جدید را گزارش می‌کنند.

- منبع محرک: کاربران نهایی که به منظور تست انتخاب شدند

[1] Unit Test
[2] Test-driven development
[3] Sonarqube
[4] Jenkins
[5] Pipeline
[6] Selenium

- محرک: فیدبک شکست
- محصول: زمان اجرا
- محیط: اپ تغییر یافته
- پاسخ: ثبت خطاها و فیدبک‌ها
- اندازه گیری پاسخ: احتمال اینکه با توجه به تعداد کاربرانی که در این آزمون شرکت کردند خطاها و دلایل شکست کشف شود.

سناریو: تست توسط ابزارهای اتومات تست کل سیستم اشترک‌گذاری خودرو در هنگام استقرار

- منبع محرک: ابزار اتومات تست
- محرک: پیاده‌سازی کامل سیستم
- محصول: زمان استقرار
- محیط: سیستم
- پاسخ: انجام تست‌ها و خروجی نهایی و کنترل حالت‌های سیستم
- اندازه گیری پاسخ: طول بلندترین زنجیره تست، زمان انجام تست‌ها

سناریو: تست مجموعه‌ای از موارد آزمون توسط tester unit برای بخش apple pay پارکینگ آنلاین که توسعه آن به تازگی به پایان رسیده است.

- منبع محرک: tester unit
- محرک: مجموعه‌ای از تست‌ها
- محصول: قسمتی از سیستم (apple pay)که در حال تست است.
- محیط: زمان توسعه
- پاسخ: تست‌ها انجام می‌شود و تعداد تست‌های موفقیت آمیز و شکست خورده و زمانی که هر تست طول کشیده گزارش می‌شود.
- اندازه گیری پاسخ: میزان تلاش برای پیدا کردن خطا

فصل هشتم

قابلیت استفاده

۸ـ۱ تعریف قابلیت استفاده

قابلیت استفاده[1] به این تعریف می‌پردازد که چقدر برای کار انجام دادن یک وظیفه دلخواه راحت است. این ویژگی یکی از ارزان و راحت‌ترین ویژگی‌هایی است که می‌توان به وسیله آن کیفیت سیستم را افزایش داد. قابلیت استفاده شامل موارد زیر است:

- ویژگی‌ها سیستم یادگیری
- استفاده موثر از یک سیستم
- به حداقل رساندن تاثیر خطاها
- تطبیق سیستم با نیازهای کاربر
- افزایش اعتماد به نفس و رضایت

[1] Usability

۸ـ۲ سناریوی عمومی قابلیت استفاده

سناریوی عمومی قابلیت استفاده به صورت زیر است:

- منبع محرک[1] : کاربر نهایی
- محرک[2] : کاربر نهایی که سعی می‌کند از سیستم به طور موثر استفاده کند و آن را یاد بگیرد و تاثیر خطاها را کمینه کند، با سیستم تطبیق پذیرد و آن را تنظیم کند.
- محصول[3]: سیستم یا بخشی از سیستم که کاربر در حال تعامل با آن است.
- محیط[4]:درمرحله اجرا یا در مرحله کانفیگ
- پاسخ[5] : سیستم باید ویژگی‌های مورد نیاز کاربر را فراهم کند یا با توجه به نیازهای کاربر آن را پیش‌بینی کند.
- اندازه گیری پاسخ[6]: یک یا چند مورد از این موارد را می‌توان به عنوان اندازه گیری استفاده نمود. زمان تسک، تعداد خطاها، تعداد تسک‌های تکمیل شده، میزان رضایت کاربر، درصد موفقیت‌آمیز عملیات‌ها نسبت به کل عملیات‌ها یا میان زمان یا داده از دست رفته زمانی که خطا رخ داده است.

محققان حوزه ارتباط انسان و کامپیوتر از اصطلاحات ابتکار کاربر [7] ، ابتکار سیستم و یا ابتکار درهم استفاده می‌کنند تا توصیف کنند که کدام جفت انسان_کامپیوتر ابتکار عمل خاصی را انجام می‌دهند و تعامل گونه پیش می‌رود.

از تمایز بین ابتکار کاربر و سیستم برای بحث در مورد تاکتیک‌های قابلیت استفاده می‌توان بهره برد و سناریوهای مختلف را بررسی کرد. هدف نهایی این تاکتیک‌ها ان است که به کاربر فیدبک و یاری مناسب داده شود.

دسته اول تاکتیک‌ها پشتیبانی ابتکار کاربر است. که می‌توان آن‌ها را به صورت زیر طبقه بندی کرد.

- کنسل: سیستم باید به درخواست کنسل کردن گوش دهد و دستوری که کنسل شده متوقف شود و منابع آن ازاد گردد و مولفه‌های وابسته نیز اطلاع داده شوند.

---

[1] Source of stimulus
[2] Stimulus
[3] Artifact
[4] Enviroment
[5] Response
[6] Response Measure
[7] User Initiative

- توقف/ادامه: سیستم در صورت توقف باید به صورت موقت منابع را آزاد کند که ممکن است بوسیله تسک‌های دیگر دوباره تخصیص داده شوند.
- بازگشت: باید این امکان وجود داشته باشد که یک یک حالت قبلی به درخواست کاربر بازگردانی شود.
- ترکیب: امکان ترکیب اشیا با سطح پایین به گروه. در این صورت کاربر می‌تواند یک عملیات را بر روی یک گروه اعمال کند و نه بر روی تک تک اشیا سطح پایین.

دسته دوم تاکتیک‌ها پشتیبانی ابتکار سیستم است. که می‌توان آن‌ها را به صورت زیر طبقه بندی کرد.

- حفظ مدل تسک: زمینه را تعیین می‌کند تا سیستم بتواند از آنچه کاربر در تلاش است ایده ای داشته باشد و به او کمک کند.
- حفظ مدل کاربر: صریحا دانش کاربر از سیستم، رفتار کاربر از نظر زمان پاسخ مورد نظر و غیره را نشان می‌ُهد.
- حفظ مدل سیستم: سیستم یک مدل صریح از خود را حفظ می‌کند. که از آن به منظور تعیین رفتار سیستم استفاده می‌شود تا برخورد مناسب به کاربر ارائه شود.

۸ـ۳ طراحی فهرست بازبینی برای قابلیت استفاده

۸ـ۳ـ۱ تخصیص مسئولیت ها

اطمینان حاصل شود که مسئولیت‌های اضافی در سیستم در صورت لزوم برای کمک به کاربر در موارد زیر در نظر گرفته شده است:

- یادگیری نحوه استفاده از سیستم
- دستیابی موثر به وظیفه موجود
- تطبیق‌پذیری و پیکربندی سیستم
- بازیابی از خطاهای کاربر و سیستم

۸ـ۳ـ۲ مدل هماهنگی

در این قسمت باید تعیین شود که عناصر سیستم بر چگونگی یادگیری کاربر برای استفاده از سیستم، کامل کردن تسک‌ها، تنظیم سیستم و بازیابی خطاها و سیستم، رضایت و اعتماد به نفس کاربر تاثیر می‌گذارد. برای مثال آیا

می‌توان سیستم به رویدادهای حرکت کاربر بر روی منوها به خوبی پاسخ دهد و در زمان واقعی بازخورد مناسب بدهد؟

۸-۳-۳ مدل داده

در این قسمت انتزاعات عمده داده را که با رفتار قابل درک کاربر درگیر هستند تعیین تعیین می‌شود. باید اطمینان حاصل کرد که این انتزاعات اصلی داده ها ، عملکردهای آنها و خصوصیات آنها برای کمک به کاربر در دستیابی به وظیفه پیش رو، سازگاری و پیکربندی سیستم، بازیابی از خطاهای کاربر و سیستم، یادگیری نحوه استفاده از سیستم و افزایش رضایت کاربر طراحی شده است. به عنوان مثال، انتزاعات داده باید به گونه ای طراحی شوند که از عملیات بازگشت و لغو پشتیبانی را پشتیبانی کند: و ریزدانگی این عملیات‌ها به گونه‌ای باشد که بیش از حد طولانی نشود.

۸-۳-۴ نقشه برداری در میان عناصر معماری

در این مرحله باید تعیین شود که چه نقشه برداری از بین عناصر معماری برای کاربر نهایی قابل مشاهده است (به عنوان مثال میزان آگاهی کاربر نهایی از اینکه کدام سرویس ها محلی و کدام یک از راه دور هستند). برای آن‌هایی که قابل مشاهده هستند، تعیین شود که این قابل مشاهده بودن چگونه بر روی سهولت یادگیری کاربر برای استفاده از سیستم، دستیابی به وظیفه پیش رو، سازگاری و پیکربندی سیستم، بازیابی از خطاهای کاربر و سیستم و اعتماد به نفس و رضایت کاربر تاثیر می‌گذارد.

۸-۳-۵ مدیریت منابع

در این قسمت تعیین می‌شود که کاربر چگونه می تواند از منابع سیستم استفاده کند و پیکربندی را انجام دهد. باید اطمینان حاصل شود که محدودیت های منابع تحت تنظیمات تمام کنترل شده توسط کاربر، احتمال رسیدن کاربران به وظایفشان را کمتر نخواهد کرد. به عنوان مثال، از تنظیماتی که منجر به زمان پاسخ بیش از حد طولانی می شود، جلوگیری باید کرد. همچنین اطمینان حاصل شود که سطح منابع بر توانایی کاربران در یادگیری نحوه استفاده از سیستم تأثیر نگذارد، یا میزان اطمینان و رضایت آنها از سیستم را کاهش ندهد.

۸-۳-۶ زمان اتصال

در این بخش تصمیمات مربوط به زمان اتصال باید تحت کنترل کاربر باشد و اطمینان حاصل شود که کاربران می توانند تصمیماتی بگیرند که به قابلیت استفاده کمک می کند. به عنوان مثال، اگر کاربر می تواند در زمان اجرا، پیکربندی سیستم یا پروتکل های ارتباطی آن یا عملکرد آن را از طریق افزونه ها انتخاب کند، باید اطمینان

حاصل کرد که چنین گزینه هایی بر توانایی کاربر در یادگیری ویژگی های سیستم، استفاده موثر از سیستم، به حداقل رساندن تأثیر خطاها، سازگاری و پیکربندی بیشتر سیستم یا افزایش اطمینان و رضایت تأثیر منفی نمی گذارد.

۸ـ۳ـ۷ انتخاب فناوری

انتخاب فناوری باید به گونه‌ای باشد که سناریوهای قابلیت استفاده را بر روی سیستم بتوان اجرا کرد. باید اطمینان پیدا کرد که تکنولوژی انتخاب شده تاثیر برعکس ندارد و به ویژگی‌های یادگیری سیستم خللی وارد نمی‌کند.

۸ـ۴ مطالعات موردی

در اینجا مطالعات موردی در ۳ حوزه حمل و نقل درون شهری برای نیاز قابلیت استفاده انجام شده است.

۸ـ۴ـ۱ تاکسیرانی و حمل‌ونقل درون شهری

واسط منطقی باید به راحتی قابل فهم و استفاده از آن راحت باشد. برای جلوگیری از بروز دادن بسیاری از خطاها راهنمایی به کاربر و محدود کردن فیلدها به موارد مجاز انجام شود. کلیدهای بازگشت و لغو کردن وجود داشته باشد از ریزدانگی مناسب برخوردار باشد. امکان گروه کردن و انتخاب موارد در برنامه وجود داشته باشد، برای مثال حذف لیست سفرها

در مقاله [۴۰] نسبت به بررسی قابلیت استفاده اپلیکیشن موبایل uber پرداخته شده است، به طور خلاصه در این مقاله بیان شده است که چون اکثر کاربران از تنها انگشت شست خود برای کاربر کردن با برنامه‌های موبایل استفاده می‌کنند محل ثبت نام و ورود برنامه در مکانی که برای این انگشت راحت‌تر است گذاشته شده همچنین اگر برنامه به لوکیشن شما دسترسی داشته باشد در هنگام استفاده از نقشه به منظور تعیین لوکیشن ابتدا به محل لوکیشن شما می‌رود تا مبدا را راحت‌تر پیدا کنید. همچنین برنامه مطمئن می‌شود که در هیچ صفحه‌ای مقدار زیادی اطلاعات به شما ندهد که شما با حجم عظیمی از اطلاعات گیج شوید. همچنین اسلایدبار در کنار صفحه وجود دارد که به راحتی هم گزینه کمک را در آنجا می‌توانید پیدا کنید و هم زمانی که در صفحات خود برنامه هستید. فیدبک مناسب از زمان سفر شما را بهتون خواهد داد و از شما در مورد سفرتان نظرخواهی خواهد کرد. همچنین وقتی تاکسی شما در راه رسیدن به شماست اطلاعات کاملی از راننده و ماشین آن ارائه می‌دهد تا شما احساس بهتری داشته باشید که تاکسی شما در راه رسیدن به شماست.

سناریو: خطاهای کاربر در هنگام ورود به اپلکیشن تاکسی انلاین و کار با منوها کاهش یابد.

- منبع محرک: کاربر نهایی
- محرک: تاثیر خطاهای کاربر کمینه شود.
- محصول: سیستم، در کلاینتی که کاربر در حال حاضر با آن کار می‌کند.
- محیط: زمان اجرا
- پاسخ: گزینه فراموشی و بازگرداندن رمز عبور، استفاده از اسلایدبار، عملیات لغو، بازگشت
- اندازه گیری پاسخ: تعداد خطاها، تعداد مشکلات حل شده

۸ـ۴ـ۲ اشتراک خودرو و حمل‌ونقل درون شهری

استفاده راحت و قابل فهم، در هنگام پیدا کردن بهترین مسیر با استفاده از یک نوار پیشرفت کاربر باید از میزان پیشرفت مطلع شود. از وضعیت کاربر مطلع بوده و پیشنهادات مناسب به کاربر داده شود تا از پیچیدگی انجام تسک‌ها توسط کاربر بکاهد. کلیدهای لغو درخواست و معوق کردن آن‌ها وجود داشته باشد. همانند uber ، waze-carpool نیز از قابلیت استفاده بالایی برخوردار است، ثبت نام و ورود آن در نقاط مناسبی جایگذاری شده، دکمه‌های راهبر نیز به خوبی نمایان هستند و با علامت‌های مناسب نظیر چرخنده برای تنظیمات به کاربر مفهوم را منتقل می‌کنند. در ۳ گام پشت سر هم می‌توان نسبت به برنامه ریزی سفر، پیدا کردن رانندگان و مطلع شدن از برنامه سفر با آن کار کرد. برای اطلاع دادن به کاربر از پوش‌نوتیفیکیشن استفاده می‌کند که کار را خیلی راحت‌تر می‌کند.

سناریو: کاربر نهایی طی چند مرحله برنامه‌ی سفر را در برنامه اشتراک خودرو وارد می کند که ناگهان تصمیم به تغییر یکی از قسمت های برنامه‌ی سفر می‌گیرد.

- منبع محرک: کاربر نهایی
- محرک: درخواست بازگشت به مراحل قبل وارد کردن اطلاعات برنامه سفر
- محصول: زیر سیستم مربوط به برنامه ریزی سفر
- محیط: زمان اجرا
- پاسخ: سیستم از حالت چند گامه برخوردار بوده و امکان بازگشت به عقب و ویرایش برنامه مسافرت را به او می‌دهد، همچنین از گزینه‌های با شکل مفهومی استفاده شده که بیشترین فیدبک به کاربر از گزینه‌ها داده شود.

- اندازه گیری پاسخ: میزان هدر رفت زمان و داده ها در هنگام رخداد خطا چقدر است؟ میزان رضایت کاربر از عملکرد چقدر است؟

۸ـ۴ـ۳ پارکنیگ های عمومی و حمل‌ونقل درون شهری

نرم‌افزار به راحتی قابل استفاده باشد و با استفاده از علامت‌ها و نشانه‌ها و نمایش‌گذاری به فهم بهتر کاربر کمک کند. امکان لغو درخواست و بازگشت به عقب و فیدبک دادن به کاربر وجود داشته باشد.

در مقاله [۲۵] نسبت به بررسی طراحی اپلیکیشن Parkwhiz پرداخته شده است. علاوه بر رعایت نکات پایه علامت‌ها و نشانه‌ها وجود دکمه‌های راهنما و حرکت موارد خاصی در این برنامه نیز وجود دارد. در این برنامه کاربران می‌توانند آدرس مقصد و میزان ساعتی که پارکینگ را دردسترس می‌خواهند به عنوان گزینه وارد کنند. نه تنها این اپلیکیشن گزینه‌های پارک برای کاربر را بر اساس قیمت مقدار ساعت وارد شده نشان می‌دهد بلکه زمانی که نیاز است تا از ان پارکینگ به آن مقصد کاربر برسد را نیز نشان می‌دهد. این ویژگی منحصر به فرد باعث می‌شود که دیگر نیازی به مقایسه زمان‌بر و طاقت فرا بین گزینه‌ها کاربر نبپردازد و سریع گزینه مطلوب را انتخاب کند. این اپ با google-map و waze یکپارچه ست و برای رسیدن به پارکینگ نزدیک مقصد نیازی نیست که کاربر به صورت دستی از آن‌ها استفاده کند و مستقیم به آن‌ها هدایت می‌شود. همچنین اطلاعاتی از پارکینگ و این که چه زمانی حدودا به پارکینگ می‌رسید نیز خیلی برای کاربرانی که اولین بار است از این اپ استفاده می‌کنند مفید است. همچنین پرداخت را با ساخت یک بارکد راحت می‌کند و مکان‌هایی که تابحال در آن ها پارک کرده اید را راحت می‌توانید از صفحه اصلی دوباره‌ـ رزرو کنید.

سناریو: کاربر جدید باید بتواند در فرصت کوتاهی بتواند ویژگی‌های سیستم را یاد بگیرد.

- منبع محرک: کاربرنهایی
- محرک: کاربر می‌خواهد رزرو پارکینگ به کمک این سیستم را یاد بگیرد و به طور موثر از آن‌ها استفاده کند.
- محصول: سیستم، در کلاینتی که کاربر در حال حاضر با آن کار می‌کند.
- محیط: زمان اجرا
- پاسخ: اینترفیس آشنا و شبیه بقیه سرویس‌هایی که با نقشه کار می‌کنند باشد و اطلاعاتی از پارکینگ و این که چه زمانی حدودا به پارکینگ می‌رسد به او بازگردانده شود.
- اندازه گیری پاسخ: میزان رضایت کاربر

فصل نهم

معماری نمونه مطالعاتی تاکسی‌آنلاین Uber

۹ـ۱ معماری میکروسرویس دامنه‌گرا

اوبر [1] دارای بیش از ۲۲۰۰ میکروسرویس است[۱۵] و وجود تعداد زیادی میکروسرویس باعث پیدایش پیچیدگی زیادی در نگهداری این سرویس ها و توسعه سرویس‌های جدید می‌شود تا جایی که Uber سعی کرد با تغییراتی در معماری میکروسرویس، معماری میکروسرویس دامنه‌گرا[2] را ارائه دهد؛ در معماری میکروسرویس دامنه‌گرا سعی شده است تا با حفظ مزیت‌های معماری میکروسرویس، از پیچیدگی کل سیستم کاسته شود.

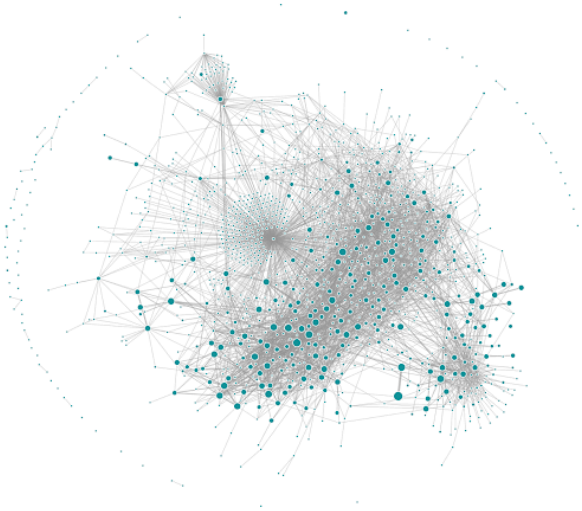
در معماری میکروسرویس، سرویس‌ها با عملکرد محدود به یک حوزه بر روی یک شبکه مستقر می‌شوند و از طریق رابط های تعریف شده به درخواست‌های که به صورت Remote Procedure Call ارسال می‌شوند پاسخ می‌دهند؛ اوبر به دلایل زیر در سال ۲۰۱۲ معماری خود را از monolithic به micro-service تغییر داد:

- ریسک‌های دردسترس‌پذیری[3] : یک خطا در سیستم monolithic می‌توانست منجر به از دست خارج‌شدن کل سیستم Uber شود.
- استقرار[4] های پرخطر و زمان‌بر

[1] Uber
[2] Domain-Oriented Microservice Architecture
[3] Availability
[4] Deployment

- تفکیک ضعیف نگرانی‌ها[1] : باوجود یک پایگاه کد بزرگ تفکیک مرز میان منطق تجاری و مولفه‌ها در یک سیستم با رشد بسیار سریع به خوبی صورت نمی‌پذیرد.
- اجرای ناکارآمد : وجود مشکلاتی که به چندین تیم وابستگی دارد، سبب می‌شود تا کارایی در اجرا بسیار پایین باشد.

هر چند مهاجرت از سیستم با معماری monolithic به microservice در زمانی که اندازه شرکت اوبر به صدها مهندس رسیده بود بسیاری از مشکلات را حل کرد، اما زمانی که تعداد میکروسرویس‌ها افزایش یافت شرکت متوجه پیچیدگی روزافزون معماری میکروسرویس شد؛ به عنوان مثال گاهی نیاز است برای یافتن ریشه‌ی یک خطا چندین میکروسرویس از تیم‌های مختلف مورد بررسی قرار گیرند. همچنین وابستگی میان سرویس‌ها گاه سبب می‌شود تا تاخیر پاسخ یک سرویس دیگر قابل قبول نباشد.همان طور که از پیچیدگی شبکه Uber در سال ۲۰۱۸ ۹ـ۱ مشخص است میکروسرویس ها شدیدا به یکدیگر وابسته هستند.



به جهت پیاده‌سازی یک امکان جدید در Uber یک تیم باید با سرویس‌های متفاوت متعلق به تیم‌های متفاوت کار کند و این عمل به جلسات زیادی بر روی طراحی و بازبینی کد نیاز دارد. همچنین مزیت معماری میکروسرویس در مورد داشتن خطوط مشخص مالکیت سرویس، هنگامی که تیم ها در سرویس های یکدیگر کد ایجاد می‌کنند، مدل‌های داده یکدیگر را اصلاح می کنند و حتی از طرف دارندگان سرویس نسخه‌های جدید را استقرار[2] می‌دهند، به خطر می افتد.در نتیجه با افزایش تعداد سرویس‌ها گویا با یک سیستم Network Monolithic سر کار داریم که یکی از نتایج آن این است که چندین میکروسرویس به ظاهر مستقل نیاز دارند تا همزمان در سامانه مستقر شوند تا سامانه بدون نقص به عملکرد خود ادامه دهد.

[1] Concerns
[2] deployment

در "معماری سرویس دامنه‌گرا" به طور عمده از روشهای تثبیت‌شده ساختاردهی کد مانند طراحی مبتنی بر دامنه [۱] [۵۰]، معماری تمیز[۲] [۵۱]، معماری سرویس‌گرا[۳] [۵۲] و الگوهای طراحی شی‌گرا [۴] و رابطه گرا[۵] استفاده می شود.

اصول اصلی و اصطلاحات مرتبط با معماری سرویس دامنه‌گرا به شرح زیر است:

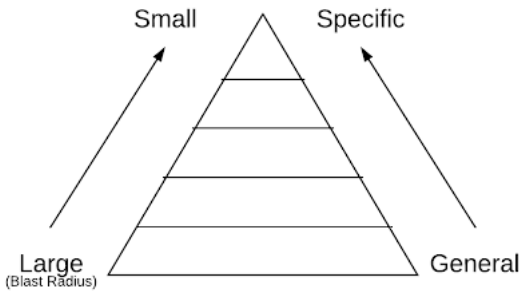
- با تعریف Domain به جای شکل گیری پیرامون یک به یک میکروسرویس‌ها، پیرامون مجموعه‌ای از میکروسرویس‌های مرتبط شکل گرفته است.
- معماری Uber همچنین مجموعه دامنه هایی را ایجاد می‌کند که لایه نامیده می‌شوند. لایه‌ای که دامنه به آن تعلق دارد مشخص می کند که میکروسرویس‌های موجود در آن دامنه چه وابستگی هایی را دارند.
- معماری برای دامنه‌ها رابط هایی مشخص ارائه می‌دهد که به عنوان یک نقطه واحد ورود به مجموعه رفتار می‌کنند و به آن‌ها دروازه[۶] گفته می‌شود.
- هر دامنه باید برای دامنه های دیگر ناشناخته باشد ، به عبارت دیگر ، یک دامنه نباید منطق مربوط به دامنه دیگری را داشته باشد که در داخل کد یا مدل داده‌ای دامنه دیگر وجود دارد. از آنجا که گاهی تیم ها نیاز دارند تا منطق را در دامنه تیم دیگری قرار دهند، ما یک معماری تکمیلی برای پشتیبانی از نقاط توسعه یافته تعریف شده در دامنه ارائه می دهیم.

دامنه‌ها به واسطه گردهمایی سرویس‌هایی که منطق تجاری مشابهی دارند، ایجاد می‌شوند و تعداد سرویس‌های درون یک دامنه متغیری است که دامنه‌ی گسترده‌ای دارد و دامنه ها می توانند شامل یک تا ده ها سرویس باشند؛ به عنوان مثال Uber Maps به سه دامنه تقسیم می‌شود که این ۳ دامنه در مجموع ۸۰ میکروسرویس را در بر دارند و ۳ gateway بر سر راه آن‌ها تعبیه شده است.

معماری Uber برای پاسخ به این سوال که "چه سرویس‌هایی می‌توانند چه سرویس‌های دیگری را فراخوانی کنند" دست به ایجاد لایه‌ها در سطح معماری زده‌است.با وجود لایه‌های مختلف مدیریت وابستگی در مقیاس صورت می‌پذیرد و نگرانی‌های مختلف معماری در لایه‌های مختلف از یکدیگر تفکیک می‌شوند.به کمک لایه‌ها با در نظر گرفتن شعاع انفجار شکست[۷] دامنه‌ها، دامنه‌هایی که در پایین هرم قرار می‌گیرند وابستگی‌های بیشتری

[1] Domain-driven Design
[2] Clean Architecture
[3] Service-Oriented Architecture
[4] object-oriented design
[5] interface-oriented design
[6] gateway
[7] failure blast radius

داشته و مجموعه‌ی بزرگتری از عملکردهای تجاری را ارائه می‌دهند.شکل ۹ـ۱ به خوبی ایده‌ی لایه‌بندی را نشان می‌دهد:



شکل ۹ـ۱: لایه ها

لایه‌های پایینی عملکردهایی نظیر مدیریت حساب کاربران را پشتیبانی می‌کنند در حالی که در قله هرم به عملکردهای جزئی‌تر نظیر تجربه کاربری یک امکان در موبایل پرداخته می‌شود.امکانات ممکن است با پخته‌تر شد تعمیم داده شوند و به لایه‌های پایین تر هرم مهاجرت کنند.

شرکت Uber از ۵ لایه‌ی زیر در معماری لایه‌های خود استفاده می‌کند:

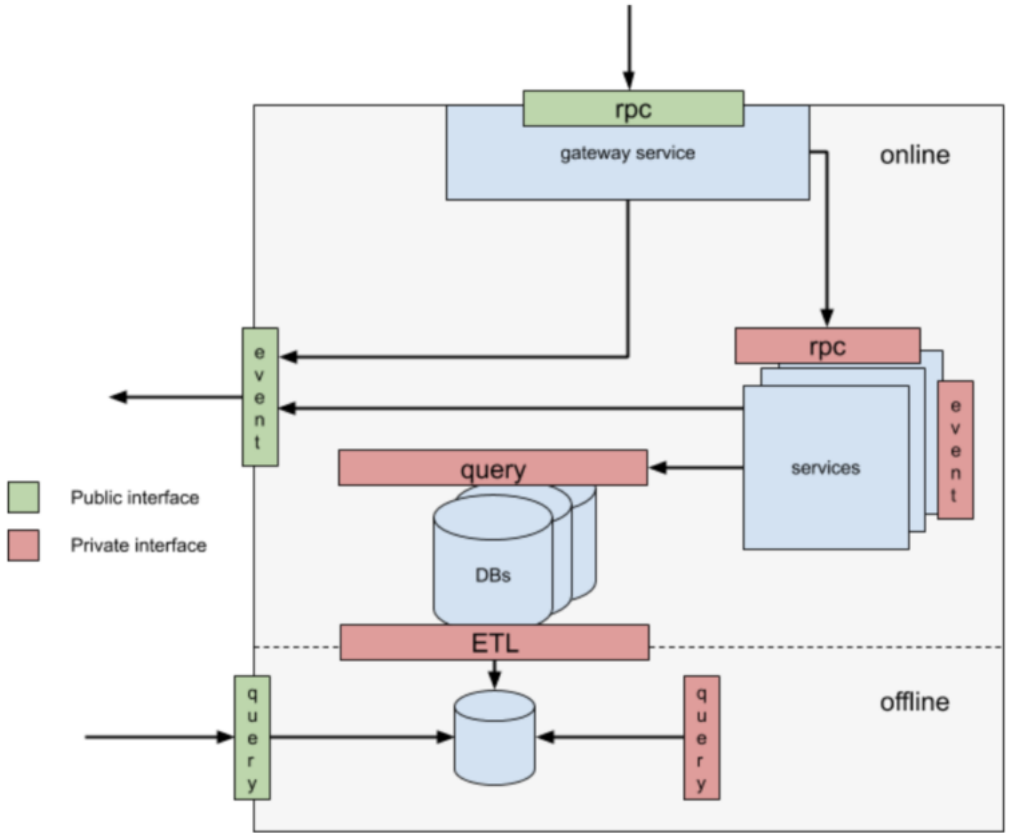
- لایه زیرساخت[1] : پاسخ سوالاتی که هر واحد مهندسی می‌تواند از آن استفاده کند نظیر فضای ذخیره‌سازی و شبکه در این لایه داده شده است.
- لایه تجاری[2] : عملکردهای تجاری که سازمان عبور می‌تواند استفاده کند اما منحصرا مربوط به یک محصول خاص نظیر Uber Ride یا Uber Eat نیستند در این لایه قرار می‌گیرد.
- لایه محصول [3] : عملکردهای تجاری که مربوط به یک محصول و خط کسب‌وکار است توسط این لایه تامین می‌شود اما این لایه نیاز های مختص به یک پلتفرم خاص را پیاده‌سازی نمی‌کند؛ به عنوان به مثال به نحوه پیاده سازی "درخواست سفر" در اپلیکیشن موبایل کاری ندارد.
- لایه نمایش [4] : نیازهای عملکردی که کاربران در زمان استفاده از یک برنامه و پلتفرم خاص نیاز دارند را برطرف می‌کند.
- لایه لبه [5] : لایه لبه امکانات Uber را به دنیای بیرون عرضه می‌کنند و اپلیکیشن‌های موبایل نیز از ویژگی‌های

[1] Infrastructure Layer
[2] Business Layer
[3] Product Layer
[4] Presentaion layer
[5] Edge Layer

این لایه استفاده می‌کنند.

همان‌طور که مشاهده می‌شود هر چه به لایه‌های بالاتر می‌رویم میزان شعاع انفجار شکست کاهش می‌یابد و عملکرد دامنه‌ها خاص‌تر می‌شود.

دروازه‌های تعریف شده در معماری میکروسرویس دامنه‌گرا بجای اینکه به یک میکروسرویس مربوط باشد به مجموعه‌ای از میکروسرویس‌ها که ما از آن‌ها با نام دامنه یاد می‌کنیم مربوط هستند.شکل ۹ـ۲ به خوبی ارتباط را نمایش می‌دهد.



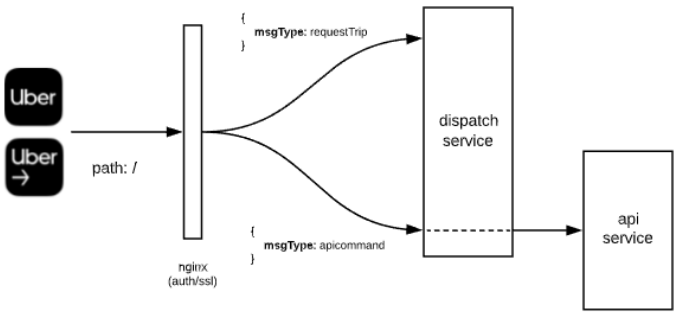
شکل ۹ـ۲: دروازه ها

اگر به معنای طراحی شی گرا به دروازه ها نگاه شود، دروازه‌ها در واقع تعاریف واسط هستند، که به ما امکان می دهد هر کاری را که می خواهیم از نظر "پیاده سازی" در پایگاه کد انجام دهیم.

۹ـ۲ طراحی دروازه رابط برنامه‌نویسی کاربردی در Uber

طراحی مناسب دروازه‌های برنامه‌نویسی کاربردی[1] نقش پررنگی در معماری میکروسرویس کاملا دردسترس، قابل توسعه و پیکربندی هم‌زمان با مشاهده‌پذیری بالا دارد. از سال ۲۰۱۴ که Uber با رشد چشم‌گیری رو به رو شد و همچنین چندین محصول دیگر نظیر Uber eat را نیز بعدا ها ارائه کرد دروازه برنامه‌نویسی کاربردی معماری این شرکت تا کنون چندین نسل تغییرات را تجربه کرده است. [۸]

در ابتدای سال ۲۰۱۴ Uber نیز همانند بسیاری از شرکت‌ها با اندازه کوچک و متوسط از یک سرویس اعزام[2] ساده استفاده می‌کرد و معماری اوبر متشکل از دو سرویس اساسی اعزام و رابطکاربردی برنامه‌نویسی [3] بود.سرویس اعزام مسئولیت ارتباط مسافران و رانندگان را بر عهده داشت و سرویس رابطکاربردی برنامه‌نویسی مجموعه ای از درخواست‌های خاص را پاسخ می‌داد.



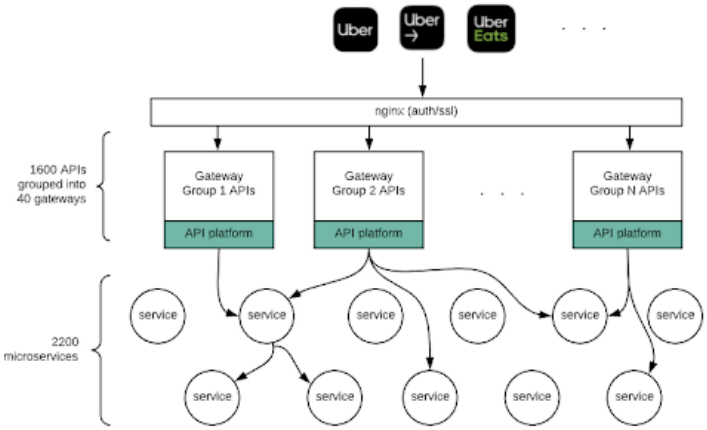
هم برنامه راننده و هم برنامه مسافر با استفاده از یک نقطه انتهایی واحد میزبانی شده در ‘/’ به سرویس اعزام متصل می شوند. بدنه نقطه پایانی دارای یک قسمت خاص به نام MessageType بود که دستور RPC را برای فراخوانی یک کنترل کننده خاص تعیین می کرد.پیام ها با ””MessageType برابر با ApiCommand به سرویس رابطکاربردی برنامه‌نویسی هدایت می‌شوند.

به فاصله کمتر از یک سال Uber با رشد زیاد خود اولین دروازه رابط کاربردی برنامه‌نویسی به معنی واقعی کلمه با قابلیت پاسخ‌گویی به هزاران درخواست در لحظه را هم‌زمان با فراهم‌آوری امکان جستجو مقصد برای کاربران را رونمایی کرد.

دروازه جدید RTAPI نامیده شد که اختصار Real Time-API است.در ابتدای سال ۲۰۱۵ دروازه تنها به یک Restfull API خدمت می‌داد اما به تدریج تبدیل به یک Public API بزرگ شد که به بیش از ۲۰ بستر[4] متفاوت

[1] API Gateway
[2] Dispatch service
[3] API service
[4] Platform

خدمت می‌کرد.این سرویس یک دروازه واحد بود که با ادامه رشد نمایی به چندین گروه استقرار تخصصی تقسیم شد.



این دروازه با برخی از آمار چشمگیر یکی از بزرگترین برنامه های NodeJS در Uber بود:

- بسیاری از نقاط انتهایی[1] در ۱۱۰ گروه‌بندی منطقی نقطه پایانی
- پاسخ‌گویی به بیش از ۸۰۰ هزار درخواست در ثانیه
- ۲.۱ میلیون ترجمه برای بومی سازی داده ها برای مشتریان انجام شده است
- ۵۰،۰۰۰ تست ادغام در هر تفاوت زیر ۵ دقیقه اجرا می شود
- برای طولانی ترین زمان تقریباً هر روز استقرار وجود داشت
- نزدیک به یک میلیون خط کد جریان‌های کاربران را کنترل می‌کردند.

۱۰۰ تیم به طور موازی امکانات جدید را ایجاد می کردند.روزانه تعداد زیادی خدمات جدید از سمت Backend ارائه می‌شد و تیم‌های موبایل با همان سرعت در حال ساخت تجربه‌های جدید در محصول بودند. دروازه وابستگی حداقلی میان مولفه‌ها را فراهم کرده و به برنامه های در Uber اجازه می‌داد تا به دروازه API پایدار و قراردادهایی که ارائه داده است، اعتماد کنند.

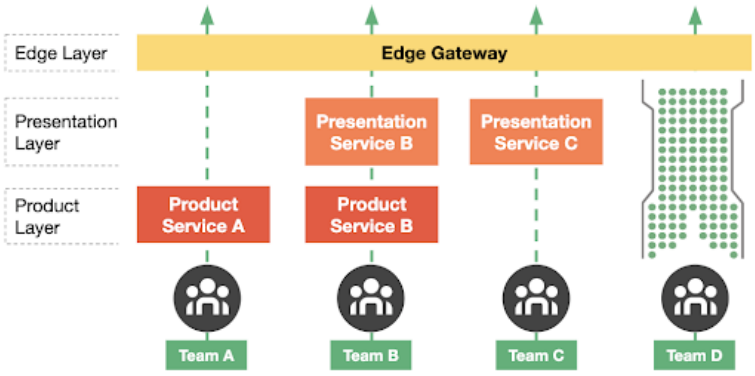
اما دروازه به تدریج بزرگ شد و شامل بیش از ۴۰ میکروسرویس مستقل شد که بیش از ۲۵۰۰ بسته NodeJs را نیاز داشت و به همین دلیل به روز نگه داشتن بسته‌های مورد استفاده در دروازه با دشواری های بسیاری همراه شد.این بدان معنی است که Uber نمی‌توانست آخرین نسخه از کتابخانه های متعدد را استفاده کند. در آن زمان

[1] End Point

Uber شروع به پذیرفتن gRPC به عنوان پروتکل جدید خود کرد اما نسخه Node.js در در حال استفاده این امکان را فراهم نمی‌ساخت.

تکرار زیاد موارد استثناهای اشاره‌گر پوچ[1] که ممکن نبود از آنها در هنگام بازبینی کد و ترافیک سایه[2] جلوگیری کرد منجر به متوقف شدن استقرار دروازه رابط کاربردی برنامه‌نویسی برای چند روز شد تا استثنا ها اصلاح شوند و این امر سرعت مهندسی تیم Uber را بیشتر کاهش می داد.

در نهایت وجود چنین مشکلاتی سبب شد تا Uber به دروازه رابط کاربردی کنونی حرکت کند. در نهایت



مشکلاتی که در نسل دوم دروازه رابط کاربردی برنامه‌نویسی وجود داشت معماران را به ساخت نسل جدید معماری لایه‌ای دروازه تشویق کرد که به صورت مختصر به توضیح آن پرداخت می‌شود.

معماری دروازه معرفی شده شامل بخش های زیر است:

- لایه‌ی لبه[3]: لایه‌ی لبه امکانات زیر را به صورت کامل در اختیار قرار می‌دهد.

– Decoupling : تیم‌های مستقل می‌توانند با حداقل وابستگی به یک دیگر به تولید امکانات بپردازند.

– transformations Protocol : همه ارتباطات بین تلفن همراه به سرور در درجه اول در HTTP / JSON است اما از نظر داخلی، Uber پروتکل داخلی جدیدی را نیز ارائه کرده است که برای تهیه پروتکل حمل و نقل دو طرفه چند منظوره ساخته شده است و همین موضوع تبدیل این پروتکل‌ها به HTTP/JSON را ضروری می‌سازد.

– concerns Crosscutting : تمام رابط های برنامه کاربردی مورد استفاده شرکت مورد نیاز یک سری عملکرد خاص است که باید معمول و قوی باقی بماند.

– payloads: Streaming ارسال Push notification به موبایل‌ها باید توسط سیستم واحد مدیریت

[1] null pointer exceptions
[2] Shadow traffic
[3] Edge Layer

جریان داده صورت پذیرد.

- لایه نمایش [1]: میکروسرویس‌ها که به طور خاص برای دست‌یابی به یک هدف پیاده‌سازی شده اند تا ویژگی ها و محصولات خود را برای قسمت FrontEnd ارائه دهند. این روش منجر به این می شود که تیم های تولیدی خدمات ارائه و ارکستراسیون خود را که API مورد نیاز برنامه های مصرف کننده را برآورده می کنند، مدیریت کنند.
- لایه محصولProduct Layer: لایه محصول به پیاده‌سازی امکانات برای محصولات خاص اختصاص داده شده است.
- لایه دامنهDomain Layer: شامل میکروسرویسها که گره برگ که قابلیت خاص منظوره شده‌ای را تنها برای یک تیم محصول فراهم می کند.

[1] Presentation Layer

۹ـ۳ نیازمندی‌های پوشش‌داده‌شده توسط معماری

در شرکت Uber تیم‌های زیادی در محصولات مختلف در سایه معماری تدوین‌شده برای سیستم فعالیت می‌کنند که در ادامه به صورت مفصل به بررسی و شرح و توضیح بیشتر نحوه پاسخ‌گویی به این نیازها پرداخته شده است.

۹ـ۳ـ۱ قابل مشاهده بودن در مقیاس بزرگ

در هنگام تصور سازمانی به بزرگی Uber یکی از نیازهای مهم قابلیت مشاهده و مانیتورینگ بسیاری از نیازهای کیفی همراه با رشد و گسترش سازمان است. برای حفظ ثبات در معماری میکروسرویس دامنه‌گرا که در بخش ۹ـ۱ به آن اشاره شد تیم مشاهده[1] شرکت Uber اقدام به ساخت خط لوله‌هایی به منظور تشخیص، هشدار و کاهش عواقب مشکلات در میکروسرویس‌های ساخته‌شده در تیم‌های مهندسی مختلف کرده‌است؛ به عنوان مثال دو نمونه از این خط لوله‌ها به جهت مراقبت از دیتاسنترها با نام‌های uMonitor و Neris ساخته شده اند.uMonitor سیستم هشدار مبتنی بر معیارها در Uber است که بر اساس پلتفرم M2 [۱۸] مراکز داده را بررسی می‌کند، در حالی که Neris در درجه اول به دنبال هشدارها در زیرساخت های سطح میزبان است.

۹ـ۳ـ۲ مدیریت منابع در Uber

یکی زیرساخت‌های نرم افزاری رایج در بین شرکتهای فناوری کلاستر[2] ها هستند.کلاستر مجموعه‌ای از منابع با میزبان‌های فیزیکی متفاوت است که با تبدیل به یک منبع مشترک منطقی در جهت تقویت توان محاسباتی و استفاده انعطاف پذیر از سخت افزار مراکز داده فعالیت می‌کند. در Uber، مدیریت کلاسترها یک لایه انتزاعی برای کارهای مختلف فراهم می کند[۲۶].

با افزایش گستره تجارت Uber استفاده کارآمد از منابع کلاسترها بسیار مهم می شود.پشته محاسباتی Uber به دلیل وجود کلاستر های اختصاصی برای موارد استفاده دسته‌ای[3] ، باوضعیت[4] و بی‌وضعیت[5] به مدیریت ویژه‌ای نیازمند است و همچنین ماهیت دینامیک کسب‌وکار Uber این مساله را سخت‌تر نیز می‌سازد؛ به عنوان مثال دامنه تقاضا برای سفرها با توجه به تعطیلات و ایام هفته بسیار متغیر است.هر چند Uber سعی کرده است تا با فراهم آوردن سخت‌افزار بیش از حد نیاز تا جای ممکن در پاسخ به نیاز مشتریان کاستی نداشته باشد اما گاهی در هنگام اوج بار یک محصول و خط کسب‌وکار، درست در زمانی که بخشی از کلاسترها اضافه بار دارند، کلاستر سایر محصولات ممکن است فضای خالی بدون استفاده داشته باشند و این تکیه بر کلاسترهای اختصاصی نیز

[1] Obserability team
[2] cluster
[3] Batch
[4] Stateful
[5] Stateless

به این معنی است که Uber نمی‌تواند منابع محاسبه ای را بین آنها به اشتراک بگذارد.

برای استفاده بهتر از منابع،Uber باید این بارهای کاری را در یک سیستم مدیریت بار واحد محاسبه کند. افزایش کارایی‌های حاصل باعث کاهش هزینه فنی هر سفر در زیرساخت های Uber می‌شود و باعث افزاش حاشیه سود سفرها در Uber و درنهایت به نفع رانندگان و مسافران خواهد شد.

راه حلی که Uber برای مدیریت کلاستر ها ارائه کرده است نرم‌افزار مدیریت یکپارچه کلاستر ها به نام Peloton است[۲۶]؛ Peloton یک برنامه‌ریز یکپارچه است که برای مدیریت منابع در بارهای متمایز طراحی شده است و مدیریت کلاستر ها در Uber را تجمیع می‌کند. Peloton با استفاده از یک پلتفرم مشترک، توازن در استفاده از منابع، به اشتراک گذاری گسترده منابع و پیش‌بینی استفاده از منابع برای نیازهای آینده در Uber را پشتیبانی می‌کند.

از سایر محصولات مشابه با Peloton می‌توان به Google Borg[۵۴]، kubernetes[۱۶] ، Hadoop YARN[۱۰] و Apache Mesos/Aurora[۶][۱۹] اشاره کرد.

۹ـ۳ـ۳ منابع ذخیره‌سازی داده در Uber

پیش از سال ۲۰۱۶ شرکت Uber از پایگاه داده Postgres استفاده می‌کرد [۲۷]، اما امروز با استفاده از روشی بدون طرح[1][۹] با ترکیب تکنولوژی‌هایی نظیر Riak[۲۹] و cassandra[۷] به همراه MySql داده‌هایی که برای طولانی مدت نیاز به نگهداری دارند، را ذخیره سازی می‌کند.در طول زمان Postgres و Mysql جای خود را به موتورهای Nosql دادند و همچنین به جهت دست‌یابی هر چه بهتر به ویژگی کیفی کارایی در بسیاری از کاربرد ها با افزایش حجم داده‌ها Riak جایگاه خود را به cassandra می‌دهد.مهندسانی که ابزارهای کاربردی و چند منظوره را برای پذیرش در سراسر Uber ایجاد می کنند،از Cassandra و Go با شدت بیشتری نسبت به تیم های دیگر در Uber استفاده می‌کنند که دلیل اصلی آن سرعت است.همچنین Uber از Redis[۲۸] هم برای ذخیره سازی داده در حافظه‌های سریع و هم برای صف بندی استفاده می کند و با استفاده از [۳۶]Twemproxy مقیاس پذیری لایه حافظه پنهان را بدون از بین بردن نسبت برخورد[2] حافظه نهان از طریق الگوریتم هش کردن سازگار فراهم می کند.

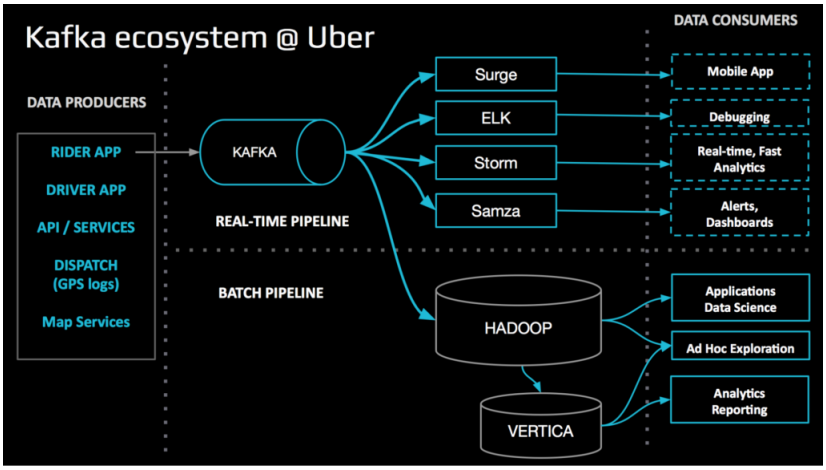
برای ذخیره سازی توزیع شده و تجزیه و تحلیل داده های پیچیده ،از انبار داده Hadoop[۱۰] به جهت ذخیره بهینه داده ها استفاده می‌شود.

[1] Schemaless
[2] Hit Ratio

۹ـ۳ـ۴ ثبت وقایع در Uber

در Uber،از Apache Kafka[۳] به عنوان یک گذرگاه پیام برای اتصال قسمت های مختلف اکوسیستم استفاده می‌شود.Uber لاگ[1] مربوط به سیستم و برنامه ها و همچنین داده های رویداد را از برنامه های رانندگان و مسافران را جمع آوری می‌کند و سپس این داده‌ها را از طریق کافکا[2] در دسترس مصرف کنندگان پایین دست قرار می‌دهد.

داده ها در کافکا توسط خطوط لوله real-time و خطوط دسته‌ای[3] تغذیه می‌شوند. داده های خطوط لوله مربوط به فعالیت هایی مانند محاسبه معیارهای تجاری ،رفع باگ[4] ،هشدار و داشبورد است.شکل ۹ـ۳ نمایی از نحوه ارتباط کافکا با سایر اجزای معماری رد Uber را نشان می‌دهد.[۳]



شکل ۹ـ۳: کافکا

۹ـ۳ـ۵ تست در Uber

برای اطمینان از اینکه سرویس‌ها قادر به پاسخگویی به تقاضای محیط واقعی هستند، مهندسین در Uber دو نرم افزار uDestroy و Hailstom را برای تست میکروسرویس‌ها ایجاد کرده‌اند.Hailstorm تست های ادغام را انجام می‌دهد و زمان اوج بار را نیز شبیه‌سازی می‌کند، در حالی که uDestroy عمداً کارها را خراب می کند تا سیستم بتواند در کنترل خرابی های غیر منتظره بهتر عمل کند.

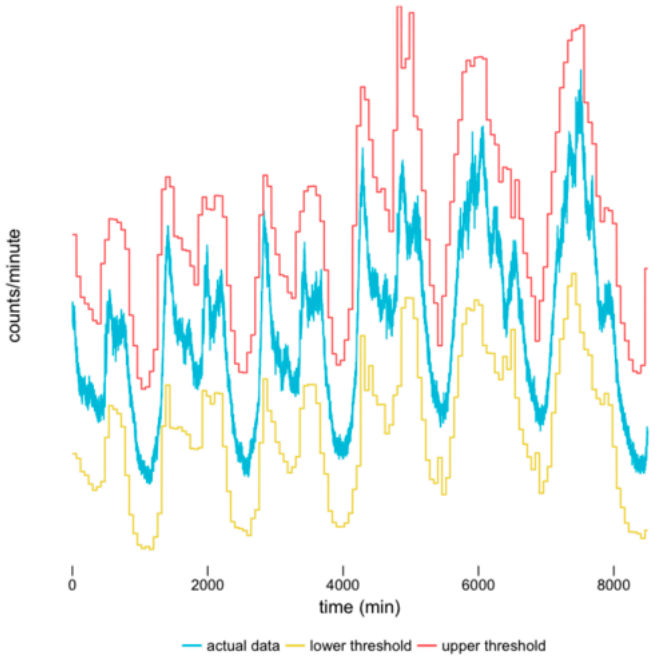
کارمندان Uber قبل از اینکه برنامه به دست کاربران برسند، از نسخه بتا برنامه برای آزمایش مداوم تحولات جدید استفاده می کنند و در طی استفاده از نسخه بتا باگ ها را گزارش و ثبت می‌کنند.[۳۷]

[1] Log
[2] Kafka
[3] batch
[4] Bug

۹ـ۴ قابلیت اطمینان و مشاهده‌پذیری در Uber

شرکت Uber برای نظارت از سامانه‌ی Nagios[۲۴] استفاده می‌کند، که به یک سیستم هشدار برای اعلان‌ها متصل است.در صورتی که کدی در بک‌اند منجر به شکست سیستم شود به مهندسان مربوطه اعلان ارسال می‌شود.

شرکت Uber به منظور مانیتورینگ شرایط بخش‌های مختلف سازمان نرم‌افزار M3[۱۷] را گسترش داده است به صورتی که هر سخت افزار و کد توسط این نرم‌افزار تحت نظر قرار دارد.همچنین این شرکت از نرم افزارهای تشخیص ناهنجاری نظیر Argos[۵] برای تشخص ناهنجاری‌ها بر اساس مدلی پیش‌بینی‌کننده مبنی بر داده‌های گذشته استفاده می‌کند که در شکل ۹ـ۴ نمایی از آن را مشاهده می‌شود.



شکل ۹ـ۴: Argos

۹ـ۴ـ۱ امنیت در Uber

بخش امنیت در Uber بر روی نوآوری‌هایی به منظور امن و مطمئن‌سازی محصولات این شرکت هم زمان با خدمت رسانی در بستر مجازی و فیزیکی تمرکز دارد.تیم‌های امنیتی در Uber وظیفه حفظ مردم و داده‌های آن ها در تقاطع دنیای دیجیتال و فیزیکی را بر عهده دارند. از عمده فعالیت‌های شرکت Uber در حوزه امنیت می‌توانیم به فعالیت‌ها در حفظ حریم خصوصی اشاره کنیم:

- استانداردهای حفظ حریم خصوصی داده‌ها را با همکاری ذینفعان حقوقی، امنیتی و سایر افراد تعریف می‌کنند.

- چارچوب حاکمیت داده ها و دستورالعمل های فنی در مورد طبقه بندی، رمزگذاری، کنترل دسترسی ،حفظ و به اشتراک گذاری داده‌ها.
- ابتکارات فنی خصوصی را در زمینه های استراتژیک از جمله داده های مکان، بستر تبلیغاتی و اشتراک داده های خارجی، با مشارکت تیم های مختلف مهندسی، سیاست های عمومی[1] و انطباق[2] ، هدایت کنید.

شرکت Uber نیز با وجود همه‌ی تلاش‌هایی که در جهت تامین امنیت محصولات خود کرده است از آسیب هکرها در امان نمانده است؛ به عنوان مثال در سال ۲۰۱۶ اطلاعات نزدیک به ۵۷ میلیون از کاربران این سرویس به سرقت رفته است.[۳۹] دست‌یابی به امنیت همانند سایر ویژگی‌های کیفی نسبی است و تیم Uber نیز سعی کرده‌است تا با جذب افراد مستعد و تشکیل تیم‌های متمرکز بر روی موضوعات خاص امنیتی به حد مناسبی از امنیت دست یابد.

۹-۵ نتیجه‌گیری

در این فصل ابتدا به بیان معماری قالب در شرکت Uber پرداخته شد و در ادامه روند رشد یکی از بخش‌های مهم معماری میکروسرویس مورد استفاده یعنی دروازه رابط برنامه‌نویسی کاربردی مورد بررسی قرار گرفت. در انتهای بخش نیز با بررسی مستندات منتشرشده موجود تحلیل مختصری از نحوه‌ی ارضای نیازهای کیفی مهم تعیین‌کننده در معماری ارائه شده است.

شرکت Uber توانسته است با حفظ سرعت رشد متناسب با رشد بازار هدف همواره سهم بزرگی از بازار را در دست بگیرد و چابکی این شرکت غول‌پیکر متاثر از معماری صحیح سازمانی و در ادامه نرم‌افزار این شرکت است.آنچه مشخص است و مستندات منتشر شده در دامنه مهندسی این شرکت آن را تائید می‌کند بازنگری همیشگی معماری این سیستم و منطبق‌سازی آن با نیازهای روز و اهمیت‌دهی به شرکت در پروژه‌های متن باز[۳۸] باعث شده است تا این شرکت از یکی از بهترین ساختارهای معماری و لایه‌های فناوری در میان سازمان‌های مشابه برخوردار باشد.

یکی از اقدامات جالب شرکت Uber این است که در زمان پیاده‌سازی یک امکان تازه ابتدا در صورت وجود به سمت نمونه‌های متن‌باز و یا با قیمت مناسب می‌رود و پس از توسعه در صورت نیاز در آینده خود شرکت اقدام به پیاده‌سازی نرم‌افزار و تکنولوژی‌های خاص برای رسیدن به اهداف خود می‌کند و در این راه از تاکتیک‌های موجود در معماری نرم‌افزار به جهت پوشش نقاط ضعف معماری استفاده می‌شود.

[1] public policy
[2] compliance team

فصل دهم

معماری نمونه مطالعاتی برنامه اشتراک سفر

۱۰ـ۱ زمینه برنامه اشتراک سفر

هدف ایجاد یک بازار برخط میان رانندگان و مسافران است تا از یک سمت نیاز مسافران برای یافتن رانندگانی که مقاصد و شرایط نزدیک به دلخواه آن برای سفر دارند رفع شود و از سوی دیگر رانندگان قادر باشند تا مسافران باب میل خود را از میان مسافرین انتخاب کنند.در این میان برنامه باید اطلاعات سفر ها را لحظه به لحظه رصد کند تا از بروز هر گونه مشکل جلوگیری شود.

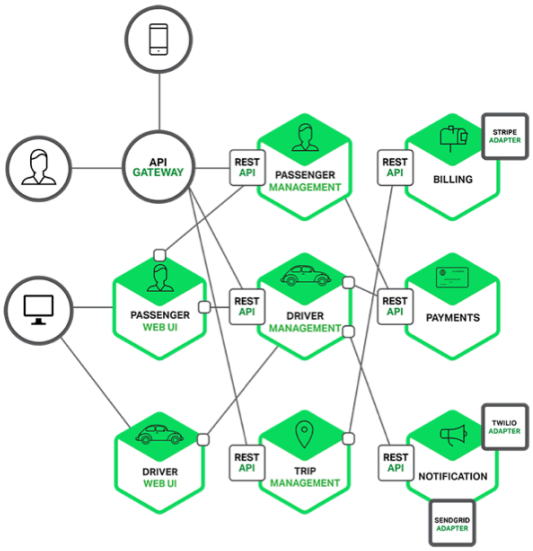
معماری پیشنهادی برای چنین زمینه[1] ای معماری micro-service [۲۰] است تا با شکستن برنامه بر اساس عملکردها[2] به میکرو سرویس های متناظر، نیازهای کیفی و غیرکیفی را مرتفع سازد.به این صورت با شکستن برنامه به مولفه‌های تقریبا مستقل بر حسب منطق تجاری به برنامه قابلیت حمل، انعطاف‌پذیری، رشد و گسترش‌پذیری بیشتری داده می‌شود. نمونه معماری مورد نظر برای برنامه اشتراک‌گذاری خودرو را در شکل ۱۰ـ۱ نمایش داده شده است [۲۱] . در مورد بحث دردسترس‌پذیری در معماری میکروسرویس، تاب‌آوری[3] مهم‌ترین فاکتور است؛ حفظ دسترس‌پذیری با مدیریت و مانیتورینگ صحیح سرویس‌ها امکان‌پذیر خواهد بود.گسترش‌پذیری در

[1] Context
[2] Functionalities
[3] Resilience

معماری میکروسرویس در گرو تخصیص بهینه منابع به سرویس ها توسط مولفه توزیع‌بار[1] است.

۱۰ـ۲ معماری پیشنهادی

الگو معماری میکروسرویس‌ها روشی برای توسعه برنامه در قالب سرویس‌های کوچک است که هر کدام به صورت تقریبا مستقل فعالیت می‌کنند.این الگو امکان تحویل/استقرار مداوم[2] برنامه های پیچیده و بزرگ را فراهم می کند و همچنین به یک سازمان امکان می دهد تا به تدریج پشته فناوری خود را توسعه دهد.



شکل ۱۰ـ۱: معماری میکروسرویس برنامه اشتراک سفر، عکس از [۲۱]

طبق معماری تعیین‌شده برنامه‌های سمت کاربر از طریق API Gateway درخواست‌های خود را ارسال می‌کنند و سرویس‌ها در ارتباط با یکدیگر به درخواست‌های کاربران پاسخ می‌دهند.

۱۰ـ۳ نیازمندی‌های پوشش‌داده شده توسط معماری

در ادامه به تجزیه و تحلیل مشخصات معماری رایج برای الگوی معماری میکروسرویس ها و میزان توانایی این الگو در رفع نیازهای کیفی پرداخته می‌شود.

چابکی و توانایی پاسخ سریع به تغییرات محیطی در کسب‌وکارهای رقابتی نظیر صنعت حمل‌ونقل اهمیت بالایی دارد.با توجه به مفهوم واحدهای مستقل و کوچک در الگوی میکروسرویس برنامه با چنین الگویی می‌تواند با کمترین هزینه تغییرات لازم را در خود اعمال کند و از قابلیت تغییرپذیری[3] بالایی برخوردار است.

به دلیل تفکیک عملکرد و دغدغه‌های تجاری در برنامه‌ها با الگوی میکروسرویس، می توان تست را محدود

[1] LoadBalancer
[2] Continouse Delivery/Development
[3] Modifiability

کرد، و این امر باعث می شود که آزمایشات هدفمندتر انجام شوند. همچنین آزمون برای یک سرویس خاص بسیار آسان تر و عملی تر از نوشتن آزمون برای یک معماری Monolithic است و چون در معماری میکروسرویس مولفه‌ها loosely coupled هستند امکان این‌که یک تغییر در یک سرویس منجر به شکست سایر سرویس‌ها شود،کم است.

به دلیل ذات توزیع‌شده و نیاز به پیام‌رسانی تحت شبکه در الگوی میکروسرویس نوشتن سرویس‌ها با کارایی بالا در خروجی مناسب بسیار اهمیت دارد و یکی از چالش‌های استفاده از الگوی میکروسرویس حفظ کارایی در هنگام گسترش برنامه است اما در مقابل چالش کارایی گسترش‌پذیری برنامه تحت این الگو به راحتی صورت می‌پذیرد و با ساختار خود پیچیدگی مسائل رو با شکستن به ریزسرویس ها کاهش می‌دهد. همچنین وجود امکان ارتباط راحت با سرویس‌های دیگر نیز به قابلیت همکاری در این معماری نیز کمک می‌کند.

پویایی معماری میکرو سرویس این امکان را می‌دهد که بتوان به چالش‌های امنیت نیز پاسخ داد، با استفاده از این معماری می‌توان تمامی ارتباطات را رمز کرد، تمامی درخواست‌ها را احراز هویت کرد و به چالش‌های رمزگذاری و دنبال کردن نشست کاربران و جلوگیری از حملات منع سرویس با استفاده از فیلترها نیز پاسخ داد [۱۲] .

از انجا که در این معماری می‌توان رنج وسیعی از API ها را برنامه ریزی کرد بنابراین در صورت طراحی مناسب API ها می‌توان به نیازمندی‌هایی که موجب افزایش قابلیت استفاده کاربر می‌شود نیز پاسخ داد.

در پترن میکروسرویس هر سرویس پایگاه داده خود را دارد. با این وجود برخی از تراکنش‌های تجاری شامل سرویس های مختلفی هستند بنابراین به مکانیزمی برای اجرای تراکنش‌ها نیاز است. اجرای هر تراکنش تجاری که شامل چندین سرویس باشد یک Saga است، در واقع Saga شامل چندین تراکنش محلی است. هر تراکنش محلی که پایگاه داده یک سرویس را به روز می کند،یک پیام یا رویدادی را منتشر می کند تا تراکنش محلی بعدی را در saga آغاز کند.یکی از خوبی های استفاده از Saga این است که اگر یکی از تراکنش‌های محلی شکست بخورن saga مجموعه اقداماتی را جهت rollback تراکنش‌ها انجام می‌دهد.[۳۰]

۱۰ـ۴ نتیجه‌گیری

معماری میکروسرویس پیشنهادی، مزایای زیر را با خود به همراه خواهد داشت:

- اول ، این که میکروسرویس مشکل پیچیدگی زیاد را حل می کند. پترن میکروسرویس به عنوان مثال یک معماری یکپارچه[1] را به مجموعه ای از خدمات تجزیه می کند و بنابرین با حفظ کل عملکرد سیستم،

[1] Monolithic

برنامه به چند بخش سرویس قابل مدیریت تقسیم شده است و هر سرویس دارای یک مرزبندی کاملاً مشخص به واسطه‌ی RPC یا message-driven API است. الگوی معماری میکروسرویس‌ها سطحی از ماژولاریتی[1] را اعمال می کند که دستیابی به آن با پترین یکپارچه بسیار دشوار است. در نتیجه ، توسعه سرویس‌ها بسیار سریعتر و درک و نگهداری آنها بسیار آسان تر است.

- دوم ، این معماری امکان توسعه هر سرویس را به طور مستقل، فراهم می‌کند و تیم‌های توسعه حول سرویس‌ها شکل می‌گیرند. توسعه دهندگان، به شرطی که این‌که قرارداد ها پیرامون API ها را رعایت کنند، در انتخاب هرگونه فناوری آزاد هستند؛ البته ، اکثر سازمان ها مایلند از هرج و مرج کامل جلوگیری کرده و گزینه های فناوری را محدود سازند. با این حال ، این آزادی به معنای آن است که توسعه دهندگان دیگر ملزم به استفاده از فن آوری های احتمالاً منسوخ شده ای که در آغاز یک پروژه جدید وجود داشته است نیستند و هنگام نوشتن سرویس جدید ، امکان استفاده از فناوری‌های روز را دارند. علاوه بر این ، از آنجا که خدمات نسبتاً کوچک هستند، بازنویسی یک سرویس قدیمی با استفاده از فناوری روز امکان پذیر است.

- سوم ، در الگوی معماری میکروسرویس می‌توان هر میکروسرویس را مستقلاً مستقر[2]کرد. توسعه دهندگان هرگز نیازی به هماهنگی در استفاده از تغییراتی ندارند که از منظر سرویس محلی باشد.

- چهارم ، الگوی معماری میکروسرویس ها مقیاس بندی هر سرویس را به طور مستقل امکان پذیر می کند.

- پنجم، سرویس‌ها به دلیل اندازه کوچک و وظیفه مشخص بسیار آزمون پذیر هستند.

در کنار مزایایی که یک معماری با خود به همراه دارد نقاط ضعفی نیز وجود خواهند داشت که باید در ادامه بر اساس ASR های مطرح برای ذی نفعان با استفاده از سایر پترن‌ها و تاکتیک‌ها در معماری به حل آن‌ها پرداخته شود.از جمله معایب استفاده از معماری میکروسرویس می‌توان به موارد زیر اشاره کرد:

- اول، تاکید زیاد بر اندازه‌ی سرویس ها و مشخص نبودن دقیق اندازه مناسب سرویس ها از جمله معایبی است که در ابتدای کار با میکروسرویس‌ها با آن برخورد خواهید داشت.

- دوم، در این پترن،تعیین حدود و مرزبندی میانن سرویس ها مشکل است.

- سوم، به دلیل ماهیت توزیع شده پیچیدگی‌های جدیدی به سیستم اضافه می‌شود.

---

[1] Modularity
[2] Deploy

- چهارم، كارايي پائين (ارتباطات شبكه اي) در معماری میکروسرویس مشهود است.

استفاده از معماری میکروسرویس به تیم‌ها اجازه می‌دهد تا با سرعت افزایش حجم بازار هدف خود را منطبق سازند و از طرفی پا به پای پیشرفت‌های تکنولوژی حرکت کرده و خود را به روز کنند.

فصل یازدهم

معماری نمونه مطالعاتی برنامه پارکینگ

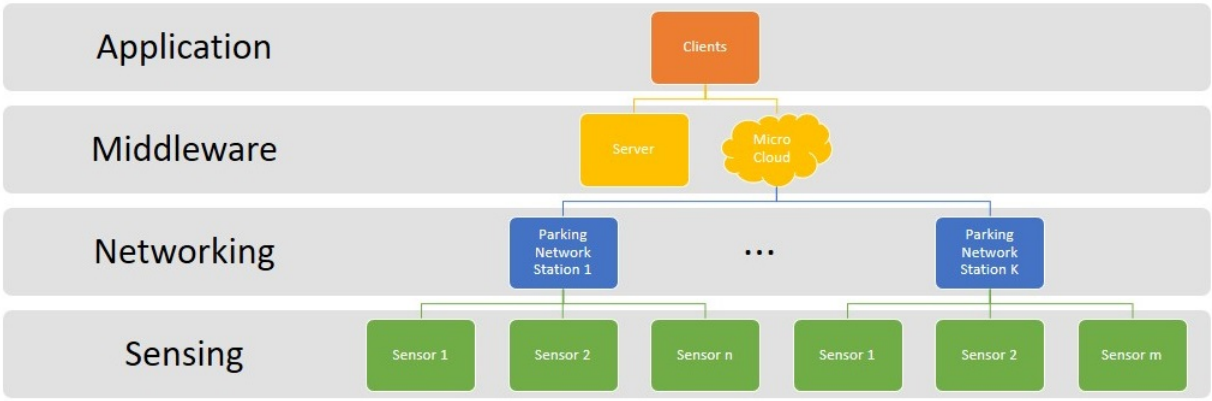
۱۱ـ۱ زمینه برنامه پارکینگ

هدف از برنامه برقراری ارتباط کاربران با پارکنیگ‌های مختلف و فضاهای خالی در هر پارکینگ است؛ کاربران در این برنامه ها باید بتوانند از مکان‌های خالی در پارکینگ‌های سطح شهر آگاه شوند و امکان رزرو مکان‌های خالی را داشته باشند.در این برنامه تعداد زیاد پارکینگ‌ها در سطح شهر و تغییرات زیاد در وضعیت پارکینگ‌ها سبب پیچیدگی مدیریت سنسورها می‌شود، در حالی که این پیچیدگی نباید خود را در پایگاه کد و سایر مولفه‌های سیستم نشان دهد.به منظور تفکیک دغدغه‌ها، معماری پیشنهادی برای این برنامه معماری لایه‌ای است.

در معماری لایه‌ای، لایه ها مستقل هستند بنابرین گروهی از تغییرات در یک لایه بر لایه های دیگر تأثیر نمی گذارد. با توجه به این ویژگی هر گونه تغییر در ساختار هر لایه این امکان را به ما می‌دهد تا مستقل از سایر لایه‌ها به بررسی تغییرات انجام شده بپردازیم. به عنوان مثال تغییر در پروتکل مورد استفاده در سنسور ها در معماری لایه‌ای صدمه ای به ساختار لایه‌های بالاتر نمی‌زند.

۱۱ـ۲ معماری پیشنهادی

با توجه به زمینه مطرح شده برای برنامه پارکینگ‌های شهری، معماری لایه ای پیشنهادی باید دارای لایه‌های شکل ۱۱ـ۱ باشد. این شکل با توجه به مقاله [۴۶] کشیده شده است. در معماری لایه‌ای پیشنهادی شده حیطه



شکل ۱۱ـ۱: معماری لایه‌ای پیشنهادی برای برنامه پارکینگ‌های شهری

عملکردی هر لایه به شرح زیر است:

- لایه‌ی برنامه[1] : برنامه‌های سمت کاربر شامل برنامه‌ی تلفن همراه و نسخه‌های وب در این لایه پیاده سازی می‌شوند و از طریق رابط ها از لایه‌ی پایینی خود خدمات را دریافت می‌کنند.کاربران قادر به جستجو در مکان های مورد نظر خود برای پارکینگ هستند و می توانند فضای مورد نظر خود را رزرو کنند.

- لایه میان‌افزار[2] : در این لایه اطلاعات پارکینگ ها به صورت لحظه‌ای دریافت می‌شود و پس از پردازش خدماتی که کاربران در لایه‌ی برنامه نیاز دارند، در اختیارشان قرار داده می‌شود. خود سرور در این لایه از معماری میکرو سرویس تبعیت می‌کند که پیشتر ویژگی‌های آن مطرح شد.

- لایه شبکه [3] : لایه‌ی شبکه به منظور ارتباط آسان تر سنسورها با لایه‌ی میان افزار تعبیه شده است. اگر قرار بود هر یک از سنسورها خود مستقلا به لایه‌ی میان‌افزار وصل شوند پیچیدگی شبکه و مدیریت آن بسیار دشوار بود.لایه‌ی شبکه با تجمیع اطلاعات سنسورها ضمن کشف خرابی یا اطلاعات ناصحیح در میان سنسورها با پروتکل‌های مطمئن تری ضمن هزینه کمتر اطلاعات را در اختیار لایه میان‌افزار قرار می‌دهد.

لایه شبکه ارتباط بی نقص بین مراکز مختلف پارکینگ و سیستم میان‌افزار و در نهایت کاربران را تضمین می کند. داده های کاربر و مرکز پارک از طریق این لایه به سیستم میان‌افزار منتقل می شود. این لایه شامل

[1] Application
[2] Middleware
[3] Networking

انواع مختلف فن آوری های ارتباطی نظیر شبکه LAN و WAN است.

- لایه سنسورها[1] : سنسورها از نوع های مختلف در این لایه قرار دارند و این لایه با محدودسازی پیچیدگی‌های سخت‌افزاری، مانع از انتشار آن به لایه‌های بالاتر خواهد شد.

۱۱ـ۳ نیازمندی‌های پوشش‌داده شده توسط معماری

در معماری لایه‌ای از انجا که هر لایه مستقل از دیگری است موجب اصلاح پذیری بالا و قابلیت آزمون در این سیستم می‌شود. چرا که می‌توان برای مثال در لایه سنسورها، پروتکل آن‌ها و دیگر موارد را تغییر داد یا هر لایه را مجزا آزمون کرد. همچنین در این معماری می‌توان امنیت را بر بر روی ارتباطات بین کاربران تا سرور، یا سنسورها تا میکروکلاد به راحتی داشت.

از انجا که خود سرور از معماری میکرو کرنل استفاده می‌کند نیز خصوصیات معماری میکروکرنل نیز به آن نیز اضافه می‌شود.

به راحتی می‌توان نیازمندی‌های مختلف را با API ها برآورده کرد که به پوشش نیازمندی‌های احراز هویت و پیاده‌سازی قابلیت استفاده پاسخ می‌دهد. آزمون راحت سرویس‌ها و تغییر آن‌ها نیز به دلیل ویژگی loosely coupled مولفه‌ها نیز امکان‌پذیر است.

همچنین بدیل ذات توزیع شده معماری میکرو کرنل و مستقل بودن معماری لایه گسترش‌پذیری نیز امکان پذیر است که از نتایج این گسترش پذیری می‌توان به افزایش کارایی و دردسترس‌پذیری این سرویس نیز اشاره داشت. وجود امکان ارتباط راحت با سرویس‌های دیگر نیز به قابلیت همکاری در این معماری نیز کمک می‌کند.

۱۱ـ۴ نتیجه‌گیری

معماری لایه‌ای با ایده‌گیری از آن چه در زیرساخت اینترنت در سراسر جهان وجود دارد یک پیشنهاد بسیار مناسب برای سیستم پارکینگ است که از یک سو با سخت افزارهای خاص و لایه‌ی فیزیکی مشابه آن چه در شبکه اینترنت شاهد هستیم مواجه است و سمت دیگر در لایه‌ی برنامه ممکن است تغییرات بر خلاف لایه‌های پایین‌تر با سرعت بیشتری روی دهند.در لایه برنامه می‌توان الگوهای خاصی جهت همکاری با لایه‌های پایین در استفاده کرد و معماری لایه‌ای دست معماری را برای ابتکار عمل در هر لایه باز خواهد گذاشت. استفاده از معماری لایه ای ممکن است تغییرات در سطح وسیع به صورتی که چندین لایه را درگیر کنید را دشوارد کند و همچنین لایه ها ممکن است عملکرد برنامه را تحت تأثیر قرار دهند زیرا باعث ایجاد سربار در اجرا می شوند؛

[1] Sensing

هر لایه در سطوح بالا برای هر عملکرد در سیستم باید به لایه های پایین تر متصل شود.

References

[1] 24/7 access to parkwhiz. https://help.parkwhiz.com/en_us/247-access-HJagl1gZWPW. Accessed: 2021-06-02.

[2] 6 advantages and disadvantages of carpooling. https://connectusfund.org/6-advantages-and-disadvantages-of-carpooling. Accessed: 2021-06-02.

[3] Apache kafka. http://kafka.apache.org/. Accessed: 2021-06-02.

[4] Apache kafka at uber. https://eng.uber.com/kafka/. Accessed: 2021-06-02.

[5] Argos. https://eng.uber.com/argos-real-time-alerts/. Accessed: 2021-06-02.

[6] aurora. http://aurora.apache.org/. Accessed: 2021-06-02.

[7] cassandra. https://cassandra.apache.org/. Accessed: 2021-06-02.

[8] Designing edge gateway, uber's api lifecycle management platform. https://eng.uber.com/gatewayuberapi/. Accessed: 2021-06-22.

[9] Designing schemaless, uber engineering's scalable datastore using mysql. https://eng.uber.com/schemaless-part-one-mysql-datastore/. Accessed: 2021-06-02.

[10] hadoop. https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html. Accessed: 2021-06-02.

[11] How do we measure the costs of software testing. https://www.softwaretestinggenius.com/how-do-we-measure-the-costs-of-software-testing/. Accessed: 2021-06-02.

[12] How to secure microservices architecture. https://securityintelligence.com/posts/how-to-secure-microservices-architecture/. Accessed: 2021-06-02.

[13] Integrating waze with arcgis. https://working-with-waze.hub.arcgis.com/. Accessed: 2021-06-02.

[14] Interoperability and the digital payment ecosystem. https://valid.com/digital-payment-ecosystem. Accessed: 2021-06-02.

[15] Introducing domain-oriented microservice architecture. https://eng.uber.com/microservice-architecture. Accessed: 2021-06-02.

[16] kubernetes. https://kubernetes.io/. Accessed: 2021-06-02.

[17] M3. https://www.youtube.com/watch?v=89H48IwFeV4. Accessed: 2021-06-02.

[18] M3: Uber's open source, large-scale metrics platform for prometheus. https://eng.uber.com/m3/. Accessed: 2021-06-02.

[19] mesos. http://mesos.apache.org/. Accessed: 2021-06-02.

[20] Microservice architecture. https://microservices.io/. Accessed: 2021-06-02.

[21] Microservices: an agile architecture for carpooling application. https://donetechno.com/en/microservices-an-agile-architecture-part-1/. Accessed: 2021-06-02.

[22] Modular agile: Loosely coupled, highly cohesive ceremonies. https://dzone.com/articles/modular-agile-loosely-coupled. Accessed: 2021-06-02.

[23] Myway: Requirement specification and analysis. https://www.sonarsource.com/why-us/code-quality/. Accessed: 2021-06-02.

[24] Nagios. https://www.nagios.org/. Accessed: 2021-06-02.

[25] Parkwhiz: A design review. https://medium.com/@corresrachel/parkwhiz-a-design-review-98c15033af36. Accessed: 2021-06-02.

[26] Peloton: Uber's unified resource scheduler for diverse cluster workloads. https://eng.uber.com/resource-scheduler-cluster-management-peloton/. Accessed: 2021-06-02.

[27] Project mezzanine: The great migration. https://eng.uber.com/mezzanine-codebase-data-migration/. Accessed: 2021-06-02.

[28] redis. https://redis.io/. Accessed: 2021-06-02.

[29] Riak. https://riak.com/. Accessed: 2021-06-02.

[30] Saga distributed transactions. https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga. Accessed: 2021-06-22.

[31] Sonarqube, code quality. https://www.sonarsource.com/why-us/code-quality/. Accessed: 2021-06-02.

[32] Starnavpilot, navigation software. https://www.aragon-technologies.com/en/starnavpilot/interoperability_with_google_maps.php. Accessed: 2021-06-02.

[33] Support terms of parkwhiz. https://www.parkwhiz.com/support/terms/. Accessed: 2021-06-02.

[34] The technological race to find you a place to park. https://www.nytimes.com/2017/11/30/business/car-parking-apps.html. Accessed: 2021-06-02.

[35] Three tenets of information security. https://www.lbmc.com/blog/three-tenets-of-information-security/. Accessed: 2021-06-02.

[36] Twemproxy. https://github.com/twitter/twemproxy. Accessed: 2021-06-02.

[37] The uber engineering tech stack, part i: The foundation. https://eng.uber.com/tech-stack-part-one-foundation/. Accessed: 2021-06-02.

[38] Uber github. https://github.com/uber. Accessed: 2021-06-22.

[39] Uber paid hackers to delete stolen data on 57 million people. https://www.bloomberg.com/news/articles/2017-11-21/uber-concealed-cyberattack-that-exposed-57-million-people-s-data. Accessed: 2021-06-22.

[40] Uber usability analysis. https://medium.com/@monikaadarsh/does-uber-app-have-an-uber-user-experience-e054189e60e5. Accessed: 2021-06-02.

[41] Uber's real-time push platform. https://eng.uber.com/real-time-push-platform/. Accessed: 2021-06-02.

[42] Under the hood of uber's experimentation platform. https://eng.uber.com/xp/. Accessed: 2021-06-02.

[43] Using waze and scoop apps for carpool commuting. https://cubicletherapy.com/waze-scoop-carpool/. Accessed: 2021-06-02.

[44] Waze carpool. https://www.waze.com/carpool/. Accessed: 2021-06-02.

[45] Wikipedia: Availability. https://en.wikipedia.org/wiki/Availability. Accessed: 2021-06-02.

[46] Ahmed, S., Rahman, M. S., Rahaman, M. S., et al. A blockchain-based architecture for integrated smart parking systems. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)* (2019), IEEE, pp. 177–182.

[47] Ataie, E., Babaeian, M., and Aghaei, F. Improving software quality by pattern driven software architecture. In *5th Symposium on Advance in Science & Technologies* (2011).

[48] Bass, L., Clements, P., and Kazman, R. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[49] CIUBĂNCAN, D., and ȚIRBAN, P. Empirical validation of oo metrics impact on changeability. *Studia Universitatis Babes-Bolyai, Informatica 65*, 1 (2020).

[50] Evans, E., and Evans, E. J. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[51] Martin, R. C. *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall, 2018.

[52] Perrey, R., and Lycett, M. Service-oriented architecture. In *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on* (2003), IEEE, pp. 116–119.

[53] Sudmanns, M., Tiede, D., Lang, S., and Baraldi, A. Semantic and syntactic interoperability in online processing of big earth observation data. *International journal of digital earth 11*, 1 (2018), 95–112.

[54] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems* (2015), pp. 1–17.