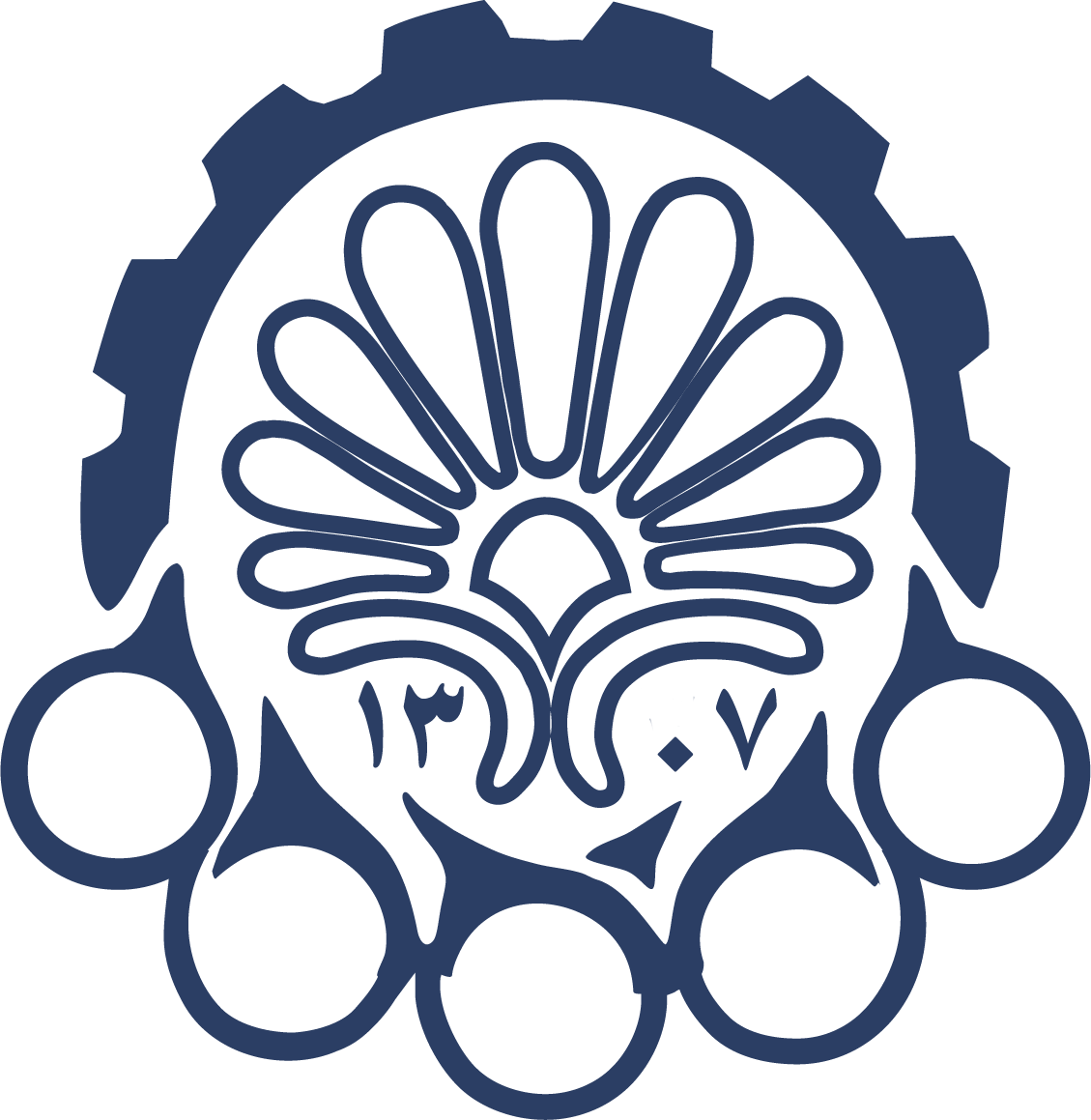
به نام خدا

دانشگاه صنعتی امیرکبیر ( پلی تکنیک تهران )

دانشکده مهندسی کامپیوتر



پروژه درس مهندسی نرم افزار 1

موضوع پروژه : **سیستم مدیریت رستوران**

نام گروه : Mamo Masa

سرپرست تیم : **سعید معروف**

نماینده مشتری : **مهدی آفریده**

سایر اعضای گروه :

**محمد طلابی**

**مهدی نیک نژاد**

استاد درس : علی کمالی

نیمسال دوم 1398-99

**فهرست مطالب**

[تعریف پروژه 1](#_Toc47446143)

[**معرفی محصول 1**](#_Toc47446144)

[**قابلیت های محصول 1**](#_Toc47446145)

[**قالب های کاربران و دسترسی ها 2**](#_Toc47446146)

[**محدودیت ها 2**](#_Toc47446147)

[نمودار محتوا (Context Diagram) 3](#_Toc47446148)

[انتخاب متدولوژی و نحوه ی انجام کار 3](#_Toc47446149)

[**نیازمندی های کلیدی 3**](#_Toc47446150)

[**سناریوی پذیرش(Acceptance Test) 4**](#_Toc47446151)

[**انتخاب متدلوژی و Process Model 9**](#_Toc47446152)

[**Methodology 9**](#_Toc47446153)

[**Process Model 11**](#_Toc47446154)

[آنالیز نیازمندی های مبتنی بر سناریو 19](#_Toc47446155)

[**Use-Case Diagram 19**](#_Toc47446156)

[**Activity Diagram 19**](#_Toc47446157)

[Swimlane diagram 23](#_Toc47446158)

[Technical Review 38](#_Toc47446159)

[آنالیز نیازمندی ها جنبه ساختاری 40](#_Toc47446160)

[**رسم CRC Cards 40**](#_Toc47446161)

[Class diagram 47](#_Toc47446162)

[Sequence diagram 48](#_Toc47446163)

[Technical Review 55](#_Toc47446164)

[طراحی معماری 56](#_Toc47446165)

[**انتخاب و استدلال استایل های کاندید برای معماری سیستم 56**](#_Toc47446166)

[**انتخاب و استدلال الگوهای کاندید برای معماری سیستم 58**](#_Toc47446167)

[ارائه حداقل دو طرح برای معماری سیستم در قالب 61](#_Toc47446168)

[مدیریت پروژه 68](#_Toc47446169)

[**از چه ابزاری برای مدیریت پیکره بندی و CD/CI استفاده میکنید ؟ چرا ؟ 68**](#_Toc47446170)

[**در صورتی که از متدلوژی های چابک برای تولید نرم افزار خود استفاده میکنید ، موارد زیر را تکمیل کنید 71**](#_Toc47446171)

[**Sprint Map 71**](#_Toc47446172)

[**User Stories 72**](#_Toc47446173)

[**DoD 78**](#_Toc47446174)

[**DoR 79**](#_Toc47446175)

[**خروجی یکی از جلسات Sprint Review 80**](#_Toc47446176)

# **تعریف پروژه**

## **معرفی محصول**

در قرن 21 که همه چیز با سرعت هر چه تمام تر به سمت استفاده ی بیشتر از تکنولوژی و اتوماسیون و بردن کار ها در قالب اپلیکیشن و استفاده از اینترنت می رود ، بدیهی است که انسان ها به دنبال راهی ساده تر برای رفع پایه ای ترین نیاز خود یعنی تغذیه و سفارش غذا بصورت آنلاین هستند. نرم افزار ها و استارتاپ هایی در این زمینه فعالیت خود را شروع کرده اند و در حال توسعه و تکامل هستند مانند اسنپ فود ، چنگال ، ریحون ، دلینو و... . اما سطح خدمات رسانی این محصولات در قالب پیک موتوری و اطلاع به رستوران می باشد و فقط کار مشتری را ساده تر می کنند. حال بر آن شدیم تا با برقراری ارتباط و ایجاد لینک میان آشپز ، حسابدار ، مدیر نیز این سطح تعامل را ارتقا بخشیم و کاری کنیم که آشپز بتواند در هر جایی بیرون از رستوران یا در خانه ی خود ، مدیر و حسابدار نیز حتی در سفر بتوانند مشغول کار باشند و بر همه چیز دسترسی داشته باشند.

این محصول قصد دارد میان پیک ، مشتری ، مدیر ، آشپز و حسابدار ارتباط برقرار کند و آن ها را از کار های یکدیگر باخبر سازد. لذا میتوان گفت این محصول به افراد ذکر شده خدمات عرضه میکند و از مهم ترین ذینفعان محصول می باشند. از این محصول همانطور که ذکر شد موارد شبیه و مشابه وجود دارد. طرف قرارداد محصول با یک *رستوران زنجیره ای* می باشد و پس از اخذ قرارداد اپلیکیشن اختصاصی با لگو و مشخصات رستوران عرضه می شود و همه میتوانند با نصب آن ، با رستوران در ارتباط باشند. لازم به ذکر است که ثبت نام در قالبی بجز مشتری ( آشپز ، حسابدار و پیک )، منوط به اجازه و تایید مدیر رستوران می باشد.

## **قابلیت های محصول**

* پشتیبانی از قالب های مختلف کاربری مانند حسابدار ، مدیر ، آشپز ، مشتری و پیک
* سفارش غذا بصورت آنلاین
* ایجاد ارتباط و اعلان های مربوطه میان کارکنان رستوران ( آشپز ، حسابدار و پیک )
* مشاهده و آگاهی از وضعیت سفارش توسط مشتری
* اعلام نظرات مشتریان نسبت به غذا ( به سیستم)
* اعلام اعتراض های کارکنان به مدیر رستوران
* اعلام نیازمندی های رستوران توسط آشپز
* اعلام اقلام نیازمندی به پیک.
* پرداخت و ویرایش آنلاین حقوق کارکنان
* سایر

## **قالب های کاربران و دسترسی ها**

* **قالب مشتری -** که قالب بیشتر کاربران محصول از این نوع می باشد. مشتری میتواند بطورکامل با انتخاب شبعه ی رستوران ، منوی رستوران را بطور کامل همراه با تصویری از غذا و مبلغ قابل پرداخت مشاهده کند. قابلیت انتخاب یک یا چند مورد از آن لیست را دارد و نهایتا می تواند با پرداخت آنلاین هزینه ی سفارش هایی که به سبد خرید فرستاده شده ، منتظر آماده شدن سفارش خود بماند. همچنین مشتری می تواند نظر خود را درمورد غذا اعلام نماید. مشتری میتواند با انتخاب گزینه ی بیرون بر ، غذای خود را در درب منزل خود ، دریافت نماید .
* **قالب آشپز –** برای هر کدام از شعبات رستوران میتوان یک یا چند آشپز تعریف شده باشد که فعالیت آن ها منوط به تایید حساب کاربری توسط مدیر می باشد. آشپز میتواند منوی شعبه را تغییر دهد و غذا ها را اضافه یا کم نماید و در صورت اتمام غذا آن را در حالت ناموجود قرار دهد. همچنین آشپز میتواند اقلام مورد نیاز آشپزخانه در طول هفته را گزارش دهد. آشپز قادر است سفارشات در جریان مشتریان را رصد کند و در صورت آماده شدن یک یا چند سفارش ، اطلاع دهد به سیستم .
* **قالب حسابدار –** برای هر کدام از شعبات میتوان تنها یک حسابدار معین نمود و فعالیت آن منوط به تایید حساب کاربری توسط مدیر می باشد. حسابدار میتواند تراکنش های واریزی شده به حساب شعبات مختلف رستوران را ببیند. اقلام مورد نیاز آشپزخانه را تایید کند و یا رد کند. همچنین میتواند وصول اقلام مورد نیاز توسط پیک خاصی را تایید یا رد نماید. پرداخت حقوق کارکنان نیز منوط به تاییدیه ی حسابدار است.
* **قالب مدیر –** در کل تنها یک نقش مدیر در اپلیکیشن موجود است. طرف قرارداد اپلیکیشن مدیر می باشد. مدیر میتواند حساب کاربری دیگر کاربران مربوط به رستوران را تایید و یا رد نماید. همچنین میتواند حقوق تعیینی آن ها را کم یا زیاد نماید. تمامی اعتراضات مربوط به کارکنان و نظرات مشتریان برای مدیر قابل مشاهده است. همچنین مدیر میتواند از طریق اینترنتی و آنلاین حقوق کارکنان بصورت ماهانه پرداخت نماید. مدیر قابلیت دسترسی به تراکنش های انجام شده بصورت کلی را نیز داراست و میتواند ریزتراکنش ها را مشاهده نماید.
* **قالب پیک –** برای هر کدام از شعبات میتوان یک یا چند پیک را تعریف نمود و فعالیت آن ها منوط به تایید حساب کاربری توسط مدیر می باشد. پیک قابلیت مشاهده ی اقلام نیازمندی آشپزخانه ی شعبه را داراست و میتواند با خرید و اعلام آن ، منتظر تایید وصول اقلام توسط حسابدار شعبه شود. همچنین قابلیت مشاهده ی سفارشات بیرون بر مشتریان را داراست و میتواند مسئولیت رساندن هر کدام از آنها را با توجه به آدرس مربوطه بر عهده گیرد.

## **محدودیت ها**

با توجه به اینکه محصول باید بتواند به راحتی در اختیار عموم قرار گیرد و همگان از آن استفاده کنند و همه جا قابل دسترس باشد ، میتوان گفت بهترین پلفترم برای پروژه ، پلتفرم اندروید و وب(web application) است که در دسترس همگان است و همچنین باید برای اتصال به محصول و استفاده از آن به اینترنت متصل شد.

# **نمودار محتوا (Context Diagram)**

مدیر

پشتیبانی مالی سیستم

خرید اشتراک

حق اشتراک

اجازه مدیریت

غذا خدمات

اطلاعات شرکت

مشتری

حسابدار

سیستم

پرداخت هزینه ی سفارش

مدیریت منابع

اطلاعات مشتری

تحویل سفارش

اطلاعات سفارش

غذای آماده شده

پیک

انجام تراکنش

درخواست پرداخت

آشپز

شاپرک

# **انتخاب متدولوژی و نحوه ی انجام کار**

## **نیازمندی های کلیدی**

* **در قالب مشتری:**
* نمایش شعبات رستوران
* انتخاب غذا از منو
* پرداخت وجه سفارش
* دریافت سفارش
* اعلام نظر
* **در قالب آشپز:**
* نمایش سفارشات
* اعلام آماده بودن سفارش
* ویرایش منوی شعبه (ویرایش غذا های موجود)
* اعلام اقلام موردنیاز آشپزخانه
* **در قالب مدیر:**
* ویرایش و پرداخت حقوق کارمندان
* مشاهده ی دخل و خرج های شعبات(تراکنش ها)
* مدیریت حساب های کاربری کارمندان ( تایید/رد حساب کاربری ، مشاهده اعتراض کارکنان و نظر مشتریان و ....)
* مدیریت شعبات ( اضافه کردن شعبه ، ویرایش کارمندان شعبه و ... )
* **در قالب پیک:**
* مشاهده ی اقلام مورد نیاز تایید شده ی شعبه
* مشاهده ی سفارشات آماده ی تحویل
* **در قالب حسابدار:**
* مشاهده ی دخل و خرج های شعبه
* مدیریت حقوق کارمندان
* امکان ویرایش منو ( تایید غذا های آپدیت شده آشپز و قیمت گذاری آنها )
* اعلام دخل و خرج به مدیر و حسابدار
* پیگیری وصول اقلام مورد نیاز توسط پیک
* **در قالب عام:**
* ثبت نام
* فراموشی رمز عبور
* ورود

## **سناریوی پذیرش(Acceptance Test)**

* **نمایش منو ها به تفکیک شعبه**
* Scope – پس از ورود مشتری به حساب کاربری خود باید لیست شعبات رستوران (شامل نام شعبه و آدرس شعبه به همراه تصویر بندانگشتی) برای مشتری به نمایش در بیاید. مشتری می تواند با انتخاب هر شعبه ، به صفحه ی منوی شعبه ی مورد نظر برود.
* Not-Scope – مشتری باید بتواند شعبه ای را انتخاب کند.

لیست شعبات باید دقیقا همانی باشد که در دیتابیس ذخیره شده است.

به محض انتخاب مشتری باید صفحه ی منوی شعبه ی مربوطه باز شود.

* **انتخاب غذا**
* Scope – پس از ورود مشتری به صفحه ی منوی شعبه ی مورد نظر خود ، منوی مربوطه برای او به نمایش در می آید. مشتری میتواند هر کدام از غذاهای درون منو را به سبد خرید خود اضافه کند. نهایتا با زدن دکمه ی پرداخت ، به صفحه پرداخت و دریافت سفارش میرود. (هر غذای درون منو شامل ، نام ، تصویر بندانگشتی ، قیمت می باشد. همچنین در پایین هر غذا و محصول یک آیکن "+" برای افزودن به سبد خرید و یک آیکن "-" برای حذف از سبد خرید وجود دارد)
* Not-Scope – اگر مشتری غذایی را انتخاب نکرده باشد نباید به سبد خرید او چیزی اضافه شود.

همچنین اگر مشتری غذایی را انتخاب کند ، باید دقیقا همان غذا به سبد خرید اضافه شود.

\*درصورتی که مشتری چند بار غذایی را انتخاب کند باید چند بار به سبد خرید اضافه گردد.

با ورود مشتری باید منوی شعبه ی مورد نظر برای مشتری به نمایش در بیاید. نه شعبات دیگر.

تصاویر غذاهای منو باید در قالب بندانگشتی برای مشتری به نمایش در بیایند.

\*اگر غذایی در سبد نباشد با زدن "-" نباید چیزی از سبد خرید کاسته شود.

* **پرداخت وجه سفارش**
* Scope – مشتری میتواند گزینه ی "ارسال با پیک" را انتخاب نماید (در اینصورت بر هزینه ی غذای او مبلغی افزوده میشود) و سپس به درگاه شاپرک برای پرداخت آنلاین هدایت میشود.
* Not-Scope – اگر مشتری گزینه ی "ارسال با پیک" را انتخاب کند باید مبلغ تعیین شده توسط مدیر در دیتابیس به مبلغ قابل پرداخت مشتری اضافه شود .

اگر در هنگام پرداخت مشکلی بوجود آمد و یا اینترنت قطع شد نباید به لیست انتظار برود.

اگر پرداخت آنلاین با موفقیت انجام نشد ، نباید پروسه تمام گردد.(مثلا باید پیغام خطا دهد)

* **دریافت سفارش**
* Scope – در حالت عادی اگر که سفارش آماده شده باشد ، برای کاربر پیغام "حاضر است"(به صورت یک notification) نمایش داده میشود.

حال اگر گزینه ی "ارسال با پیک" را زده باشد در پیام به نمایش در آمده ، اطلاعات پیک (شامل نام و شماره ی تماس) نیز موجود خواهد بود.

* Not-scope – اگر "ارسال با پیک" را نزده باشد ، پس از خروج از صف انتظار پیام "سفارش شما حاضر است" باید به نمایش در بیاید. اگر "ارسال با پیک" را زده باشد ، پس از خروج از صف انتظار باید پیام "پیک در حال آوردن سفارش شماست" به نمایش در بیاید. (که شامل مشخصات پیک نیز میباشد)

اگر اینترنت کاربر قطع و وصل شود ، نباید روند کار مختل شود و کاربر با ورود به اپلیکیشن مجددا "صفحه ی انتظار" را مشاهده کند.

* **اعلام نظر**
* Scope – بعد از خروج از "دریافت سفارش" برای کاربر ، نظرسنجی به نمایش در می آید. کاربر میتواند به این نظرسنجی جواب داده یا جواب ندهد. نهایتا به صفحه ی منوی شعبات هدایت خواهد شد.
* Not-scope- اگر کاربر گزینه ای را انتخاب نکرد و گزینه ی "تایید" را زد ، مشکلی ایجاد نشود.

نظرسنجی همراه با تمام گزینه های تعیین شده در دیتابیس به نمایش در بیاید.

* **مشاهده ی سفارشات توسط آشپز**
* Scope - سفارش های پرداخت شده شعبه توسط مشتری ها به ترتیب زمان سفارش داده شده به آشپز شعبه ی مورد نظر در قالب یک لیست نمایش داده می شود.
* Not-scope – تنها سفارشات پرداخت شده باید وارد آن لیست بشوند.

سفارشات باید شامل نوع غذا و تعداد مربوطه شوند.

سفارشات باید مربوط به شعبه ی مورد نظر باشند. (یعنی برای آشپز شعبه ی 2 نباید سفارشات شعبه ی 1 برود)

* **آماده بودن سفارشات**
* Scope - در لیست نمایش داده شده به آشپز ، درکنار هر سفارش ، دکمه ای برای اعلام آمادگی برای تحویل سفارش وجود دارد. با انتخاب آن توسط آشپز، آن سفارش از لیست حذف میشود و مشتری مربوطه از صف انتظار خارج میگردد.
* Not-scope – قابلیت انتخاب چند سفارش توسط آشپز وجود داشته باشد.

در صورت پایان یک سفارش ، مشتری دیگری از صف خارج نشود.( اطمینان پیدا کردن از لیست آپدیت شده )

* **ویرایش منوی شعبه**
* Scope - با ورود به صفحه ی منو ها ، میتوان به تک تک آیتم های مربوط به صفحه شعبه ، دسترسی پیدا کرده و برای هر آیتم میتوان گزینه های حذف ، ویرایش و یا افزودن را انتخاب نمود.
* Not-scope – با زدن آیکن حذف غذا ، غذای انتخاب شده از دیتابیس نیز حذف گردد.

درصورت اعمال هرگونه تغییر ، در دیتابیس نیز تغییرات اعمال شود.

اگر درحین وارد کردن غذای جدید ، غذای مورد نظر در منو وجود داشت ، پیغام خطا بدهد و ذخیره نشود.

* **اعلام اقلام مورد نیاز**
* Scope – صفحه(form) ای وجود داشته باشد که در آن اقلام مورد نیاز آشپزخانه جهت خرید ارسال گردد. پس از تکمیل این فرآیند ، یک نوتیفیکیشن توسط آشپز برای حسابدار حاوی پیام درج شده ارسال شود .
* Not-scope – پیام ارسال شده به درستی و شامل همان کارکتر های وارد شده توسط آشپز باشد.
* **پرداخت حقوق کارمندان**
* Scope - لیستی از حساب کاربری کارمندان سیستم بهمراه مشخصات وارده توسط آن ها به مدیر نمایش داده شود ، فیلدی حاوی حقوق مورد نظر این ماه ، برای هر کاربر در جلوی حساب او وجود داشته باشد که در صورت صلاح دید مدیر بتوان آن را ویرایش کرد. همچنین بتوان با کلیک بر روی دکمه ی پرداخت در جلوی هر حساب کاربری ، حقوق درج شده پرداخت گردد.
* Not-scope – مبلغ قابل پرداخت منفی نمیتواند باشد و در صورت وارد کردن مبلغ منفی پیغام مناسب چاپ شود.

با پرداخت کردن حقوق هر کدام از کارمندان ، کارمند مورد نظر از صف پرداختی های ماه خارج گردد.

در صورت پایان پروسه پرداخت حقوق ، کارمند دیگری از صف خارج نشود.( اطمینان پیدا کردن از لیست آپدیت شده )

* **ویرایش حقوق کارمندان**
* Scope- لیستی از حساب کارمندان به نمایش در بیاید و حقوق مورد نظر آن ها نیز در کنار آن فیلد ها نمایش داده شود. حال با انتخاب کارمند مورد نظر و انتخاب گزینه ی "اصلاح" ، حقوق کارمند انتخاب شده را افزایش یا کاهش دهد و نهایتا ذخیره شود.
* Not-scope – مبلغ وارد شده نمیتواند منفی باشد.

در صورت تغییر دادن مبلغ حقوق آن کارمند نباید تاثیری بر روی کارمندان دیگر ایجاد شود. اطلاعات باید به درستی وارد دیتابیس نیز گردد.

* **مشاهده ی دخل و خرج ها**
* Scope - تمام تراکنش های انتقال یافته و یا واریزی مربوط به حساب شرکت و رستوران ، در قالب نمودار هایی به تفکیک شعبات ، هفتگی ، ماهانه و سالانه نمایش دهد
* Not-scope – نمودار ها باید به درستی رسم شده باشند و با داده ها همخوانی داشته باشند.

نمودار ها باید به تفکیک ماه و شعبات به نمایش در بیایند.

* **مدیریت حساب کاربری کارمندان**

Scope - لیستی از حساب کاربری کارمندان سیستم به همراه مشخصات وارد شده توسط آن ها نمایش داده شود که در صورت صلاح دید مدیر ، اکانت مورد نظر حذف و یا ویرایش گردد. همچنین با انتخاب گزینه ی افزودن بتواند یک کارمند جدید را وارد سیستم گرداند.

* Not-scope – اطلاعات کابران به درستی از دیتابیس خوانده و نمایش داده شود.

اطلاعات کارمند جدید وارد شده نباید با کارمندان دیگر مشابهت زیادی داشته باشد و یا یکسان باشد.

* **مدیریت شعبات**
* Scope – لیستی از شعبات همراه با مشخصات کامل آن ها نمایش داده شوند. در صورت انتخاب گزینه ی حذف ، شعبه ی مورد نظر حذف شود. در صورت انتخاب افزودن شعبه ، با وارد کردن نام ، شماره تماس ، تصویر ، آدرس و هزینه ی ارسال پیک بتوان شعبه ی جدیدی ایجاد کرد. همچنین با انتخاب آیکن ویرایش بتوان ، اطلاعات ذکر شده را تغییر داد.
* Not-scope – شعبات نباید از نظر هر کدام از فیلد های ذکر شده یکسان باشند .

اطلاعات به درستی به دیتابیس منتقل گردد و آنجا ذخیره شود.

شعبات کمتر و یا بیشتر از شعبات دیتابیس نباشند. ( آپدیتی بگیرد از دیتابیس )

اطلاعات مربوط به شعبات غلط نباشد.

* **مشاهده ی اقلام مورد نیاز تایید شده**
* Scope – لیست اقلام مورد نیاز آشپزخانه که قبل تر به تایید حسابدار رسیده است باید در یک لیست برای پیک (ها) نمایش داده شود.در صورت انتخاب هر کدام ، نام و اطلاعات پیک بهمراه سفارش انتخاب شده ، به حسابدار گزارش شود و آن قلم از لیست اقلام حذف گردد
* Not-scope – اطلاعات پیک به درستی به حسابدار فرستاده شود.

با انتخاب و تایید هر کدام ، آن قلم در لیست اقلام باقی نمانده باشد و حذف گردد.

* **مشاهده ی سفارشات آماده ی تحویل**
* Scope – سفارشات آماده شده که مشتریانش "ارسال با پیک" را انتخاب کرده اند بصورت یک لیست برای پیک (ها) به نمایش در بیاید و هر پیک بتواند با انتخاب آن ، مسئولیت آن را برعهده بگیرد. موارد درون لیست باید شامل نام اقلام ، تعداد آنها ، قیمت شان و نام و آدرس مشتری باشد .
* Not-scope – انتخاب چند سفارش بطور همزمان وجود نداشته باشد .

سفارش انتخاب شده باید از لیست سفارشات حذف گردد .

* **تایید اقلام مورد نیاز آشپزخانه**
* Scope - نیازمندی های ارسال شده از هر آشپز و مربوط به شعبه ی کار بصورت یک لیست به نمایش در می آید. در کنار هر آیتم یک فیلد برای وارد کردن مبلغ حدودی وجود دارد و با زدن دکمه ی تایید برای هر نیازمندی ، اقلام به لیست نیازمندی اضافه گردد.
* Not-scope – مبلغ ورودی نمی تواند منفی باشد و در صورت وارد کردن عدد منفی اخطار دهد.

نیازمندی ها در صورت تایید شدن ، به درستی به لیست نیازمندی ها اضافه گردد.

مبلغ هر نیازمندی ، برای فقط همان نیازمندی نمایش داده شود.

* **تایید رسید اقلام مورد نیاز**
* Scope - لیستی از نیازمندی های در حال تهیه توسط پیک برای حسابدار بصورت یک لیست در جریان است ، که در کنار هر آیتم یک گزینه برای تایید رسید نیازمندی توسط پیک به انباری آشپزخانه می باشد. درصورتی که تایید شود ، مبلغ موجود در آیتم حقوق این ماه پیک اضافه می گردد
* Not-scope – مبلغ اضافه شده به حقوق ماهانه ، درست همان مبلغ وارد شده از طرف حسابدار باشد.
* **ورود کاربران**
* Scope – بعد از ورود به اپلیکیشن و انتخاب گزینه ی ورود کاربران ، سه فیلد حاوی "نام کاربری" ، "رمز عبور" و "عنوان" برای پر کردن وجود داشته باشد. با پر کردن آن ها و گزینه ی "تایید" ، کاربر وارد حساب کاربری خود شود. (عنوان = آشپز / پیک / حسابدار / مدیر )
* Not-scope – درصورت مغایرت داشتن اطلاعات با اطلاعات دیتابیس (بعد از اولین ثبت نام کاربر) ، پیام مناسب چاپ شود.

درصورت نبودن نام کاربری مربوطه ، پیغام مناسب چاپ گردد.

در صورتی که حساب کاربری مربوط به غیر مشتری ، تایید نشده باشد ، پیام مناسب چاپ گردد

* **ثبت نام کاربران**
* Scope – بعد از ورود به اپلیکیشن و انتخاب گزینه ی ثبت نام ، فیلد های نام ، نام کاربری ، رمز عبور ، عنوان ، آدرس و شماره ی تماس ظاهر شود و با پر کردن آن ها ، اطلاعات کاربر به دیتابیس فرستاده شود.
* Not-scope – درصورت وجود نام کاربری از قبل ، پیغام خطا به نمایش در آید.

در صورت تطابق هرکدام از فیلد ها ، پیغام خطای مربوطه به نمایش در آید.

در صورت خالی بودن هر کدام از فیلد ها ، دکمه ی تایید عمل نکند و پیغام خطای مناسب نمایش داده شود.

* **فراموشی رمز عبور**
* Scope – هر کاربر با ورود به اپلیکیشن ، میتواند با انتخاب گزینه ی "فراموشی رمز عبور" به صفحه ای وارد شود و در آنجا شماره ی همراه خود را وارد کند. سپس یک رمز یکبار مصرف(کد امنیتی) برای تلفن همراهش پیامک میشود و آنرا در صفحه وارد میکند تا بتواند وارد صفحه ویرایش رمز در داشبوردش شود .
* Not-scope – درصورتی که شماره همراه در دیتابیس موجود نباشد ، پیام مناسب دهد. ("تلفن موجود نیست")

پیامک به درستی به تلفن همراه فرستاده شود.

رمز عبور مربوط به حساب کاربری تلفن همراه وارد شده باشد. نه حساب کاربری دیگری.

## **انتخاب متدلوژی و Process Model**

### Methodology

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| آیتم / متدلوژی | XP | ASD | DSDM | Scrum | Crystal | FDD |
| **اندازه ی تیم** | 2-10 نفر | متغیر و وابسته به پروژه | 2-10 نفر | 5-9 نفر | متغیر | 4-20 نفر |
| **مدت زمان هر Iteration** | دقیقا 14 روز | 4-8 هفته | نامشخص | ماکزیمم 30 روز | تا 4 ماه | 14 روز |
| **روش توسعه** | Incremental | Iterative | Iterative | Incremental | Incremental | Iterative |
| **پروژه ی قابل توسعه** | کوچک | کوچک | همه نوع | همه نوع | همه نوع | پیچیده |
| **دخالت مشتری** | کاملا درگیر | در هر release | در هر release | در قالب PD | در هر release | طی گزارشات |
| **مستند سازی** | خیلی ساده و کلی | خیلی ساده و کلی | وجود دارد | خیلی ساده و کلی | خیلی ساده و کلی | بسیار مهم و ریز |
| **ارتباطات تیم** | غیررسمی و در قالب Daily stand-up meetings | غیر رسمی و  رو در رو | بر اساس مستندات | غیررسمی و در قالب Daily stand-up meetings | غیر رسمی و  رو در رو | بر اساس مستندات |
| **مشکلات** | مستندات ضعیف – حضور اجباری و دائم مشتری | متود ضعیف مستند سازی | مستندات پیچیده | مستندات ضعیف و کنترل ضعیف بر روی پروژه | عدم هماهنگی و نظم در گروه های کوچکتر | عدم فایده برای پروژه های کوچک ، مالکیت انفرادی کد ها |
| **استفاده از** | User Stories , Refactoring | Learning cycle | Prototyping | Planning poker , sprint and backlog , scrum master | Adaptable methods family | UML diagram |
| **سرعت روند کار** | بسیار سریع | سریع | بسیار سریع | سریع | سریع | سریع |
| **روند مدیریت پروژه** | Planning game | Adaptive cycle planning | - | توسط Scrum master | Monitoring هر progress | بررسی نتایج بوجود آمده و گزارشات |
| **جمع آوری نیازمندی ها** | User Story | JAD | Functional  Model  Iteration | Product backlog | Use case | Feature |
| **تمرکز روی** | TDD  Pair Programming  Refactoring | Self organization | Incremental prototyping | Product backlog  Sprint Review | Communication | Feature |

در جدول بالا، اطلاعاتی در مورد تمامی متودولوژی/فریمورک های اصلی مهندسی نرم افزار آمده است،که با استفاده از آن ها ،به مقایسه متودولوژی/فریمورک ها با هم می پردازیم و یکی از آن ها را برای استفاده در پروژه انتخاب می کنیم.مقایسه های انجام شده در غالب توضیحات و همچنین توضیحات بیشتر در مورد متودولوژِی/فریمورک ها در پایین آمده اند:

* **ارتباط اعضا:** با توجه به اینکه تیم توسعه، تیم کوچکي است که اعضای آن هم روی بک اند و فرانت اند کار می کنند، شفافیت در کار های اعضای تیم بسیار مهم می باشد و اینکه همه ی اعضای تیم از وضعیت پروژه با خبر باشند ،و درنتیجه باید ارتباط نزدیک و مکرر بین اعضای تیم وجود داشته باشد که به خاطر همین نمی توانیم از DSDM و FDD استفاده کنیم،چرا که در این دو فریمورک ارتباطات از طریق مستندات برقرار می شود.
* **تغییرپذیری:** چون اندازه پروژه بزرگ می باشد و نیازمندی های جدید به صورت مکرر(که باعث تغییرات در پروژه می شود) به وجود می آیند، متودولوژِی/فریمورک های XP و ASD در مدیریت این پروژه بزرگ کارامد نیستند و همچینین متودوبوژِی/فریمورک هایی مثل FDD و DSMD به خاطر استفاده از توسعه تکراری،نمی توانند نیازمندی های جدید آن را به طور مکرر پیاده سازی کنند.
* **نیازمندی به مستندسازی:** به خاطر تمرکز روی قابل فهم بودن کد و شفافیت،و امکان ایجاد شدن نیازمندی های جدید در توسعه پروژه،مستند سازی در اولویت کمتری قرار دارد و به صورت ساده کفایت می کند،که در این جهت استفاده از متودولوژی/فریمورک های(به ترتیب اولویت از کم تا زیاد) XP ، Crystal ، و Scrum بهتر است .
* **مدیریت:** با توجه به کارکردن full-stack اعضای تیم و اشنایی آن ها با گزارش نویسی ، استفاده از use case و user story و رسم دیاگرام activity ، UML وswim-lane ، روش های مدیریت پروژه در متودولوژی/فریمورک هایXP ، Crystal ،ASD با پروژه همخوانی بیشتری دارند و فریمورک هایی مانند DSDM(که در آن روش مدیریت خاصی موجود نیست،و چون پروژه بزرگ و complex می باشد،مناسب نیست) و Scrum (که در آن Scrum Master مدیریت پروژه و بهترین اسفاده از Scrum را بر عهده دارد،و کسی در اعضای تیم نیست که بتواند این نقش را به خوبی ایفا کند) از جنبه مدیریت برای پروژه مناسب نیستند.
* **حضور مشتری و کاربر:** چون دخالت کاربران در این پروژه باید زیاد باشد و توسعه آن نیازمند دریافت بازخورد از مشتری می باشد، متودولوژی/فریمورک هایی مانند FDD ، DSDM وASD برای توسعه این پروژه توصیه نمی شوند.
* **افزایشی بودن و خروجی سریع:** سرعت روند کار و درنتیجه تحویل هر جزء(increment (برنامه پروژه به صورت سریع (که در فریمورک های ،ASD،FDD، ScrumوCrystal دیده می شود) کافی است، زیرا ممکن است به خاطر سرعت زیاد از حد و بسیار سریع متودولوژِی/فریمورک ، در معماری برنامه و تقسیم بندی پروژه در ابتدای روند کار تصمیمات نادرستی گرفته شود( که باعث ایجاد مشکلات وافزایش هزینه توسعه می شود) و همچنین امکان دارد به اندازه کافی وقت نباشد که بتوان به طور مکرر جزء های قابل استفاده و نمونه های اولیه(prototype) به کاربر ارائه داد،که با توجه به ماهیت پروژه،برای ما بسیار مهم است.
* **سایر:** استفاده از متودوبوژِی xp در توسعه پروژه به شدت توصیه نمی شود،چرا که پروژه نسبتا بزرگ می باشد و در فریمورک cynefin ، در قسمت complex قرار دارد،که باعث می شود نتوانیم به طور کامل از بهترین روش ها ((best practicesاستفاده کنیم. همچنین پیداکردن ذینفعانی برای دخالت شدید در توسعه پروژه سخت می باشد.

**نتیجه:** در نهایت با درنظر گرفتن انعطاف پذیری و کاربرد **scrum** برای استفاده در پروژه های startup ای که به صورت complex با ذات adoptive هستند و همچنین به خاطر افزایشی بودن توسعه در آن که باعث پایداری پروژه در برابر تغییرات و نیازمندی های جدید می شود،و در نهایت با توجه به داشتن شرایط و تجربه لازم برای استفاده از **scrum** ،از آن برای توسعه پروژه استفاده می شود.

### Process Model

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sum | Management of Premade part  (0,5) | User **interface**  (0,5) | Risk management  (0,5) | Security management  (0,5) | Adoptability  (0,5) | Time  Management  (0,5) | Cost  Management  (0,5) | Process  Model |
| 5 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | Code and fix |
| 16 | 0 | 1 | 4 | 3 | 0 | 5 | 3 | The waterfall |
| 12 | 0 | 1 | 3 | 4 | 1 | 2 | 1 | V-model |
| 15 | 0 | 2 | 2 | 3 | 3 | 3 | 2 | Incremental |
| 11 | 0 | 3 | 2 | 2 | 2 | 1 | 1 | Prototyping |
| 12 | 0 | 2 | 3 | 2 | 2 | 2 | 1 | The spiral |
| 4 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | Concurrent |
| 20 | 5 | 3 | 2 | 3 | 1 | 4 | 2 | RAD |
| 18 | 5 | 0 | 2 | 3 | 1 | 4 | 2 | Component  based |
| 16 | 4 | 1 | 3 | 2 | 2 | 3 | 1 | Reuse  oriented |
| 8 | 0 | 0 | 4 | 4 | 0 | 0 | 0 | Formal |
| 13 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | Aspect oriented |
| 13 | 0 | 3 | 4 | 3 | 2 | 1 | 0 | The unified process |

**معیار های مقایسه :**

در جدول بالا،تمامی مدل فرآیند ها( process models ) با توجه به معیار های لازم پروژه ارزیابی شده اند ، و به هر خانه جدول(به غیر از خانه های استفاده شده برای نامگذاری)امتیازی بین 0 تا 5 نسبت داده شده است،که نشان دهنده ارزیابی مدل فرآیند مربوط به سطر خانه جدول،با توجه به معیار مربوط به ستون خانه جدول می باشد،که توضیح آن ارزیابی در پایین آمده است:

0: بسیار بد/سخت،مشخص نشده است

1:بد/سخت

2:متوسط

3:خوب/آسان

4:بسیار خوب/آسان

5:عالی/کاملا آسان

در ادامه توضیح های بیشتری در مورد هر از معیار ها و اهمیت آن ها برای پروژه ما آورده شده است.

**Cost Management**

با توجه به اینکه پروژه ما یک پروژه استارت آپ می باشد،و درنتیجه بودجه آن محدود می باشد،مدلی که برای ما کمترین هزینه را داشته باشد،اهمیت زیادی دارد.

* Code and fix: هیچ روشی برای مدیریت هزینه لازم برای پروژه را ارائه نمی کند وهزینه با افزایش چرخه های فرآیند،از کنترل خارج می شود.
* The waterfall: هزینه لازم برای پروژه در مرحله اول روند تا حدی مشخص می شود،اما با توجه به مشکلات در حین اجرای قدم های روند، مدریت هزینه کمی سخت می شود.
* V-model: به خاطر تست های زیاد لازم،و اینکه در صورت شکست تست ها باید تغییراتی در پروژه ایجاد کنیم و دوبار تست نویسی کنیم،هزینه دارد و مدیریت هزینه باید به دقت انجام شود،اما کمی از این هزینه به خاطر زود مشخص شدن خطا ها و اصلاح آن ها (که باعث می شود خطا های کمتری در آینده داشته باشیم) جبران می شود.
* Incremental: چون پروژه به صورت جزء (increment) هایی تحویل داده می شود و در توسعه هر جزء از دو تست اعتبار سنجی و صحت سنجی (V&V)استفاده می شود،خطا ها زود مشخص می شوند و هزینه کمتری بابت اصلاح خطا خواهیم داشت و مدریت هزینه راحتر خواهد بود.
* Prototyping: با توجه به ساخت مکرر نمونه اولیه یا Prototype ،هزینه کمی زیاد و مدیریت هزینه اندکی سخت می شود.
* Spiral: به خاطر تحلیل ریسک در هر چرخه،هزینه زیادی را باید پرداخت کنیم،که بخشی از آن به خاطر جلوگیری از ریسک های خطرناک ،جبران می شود و به مدریت درست هزینه کمک می کند.
* Concurrent: در مورد مدریت هزینه،روشی و ایده ای ارائه نمی دهد.
* RAD: به خاطر استفاده از جزء های آماده،هزینه زیاد می شود که باعث سختی مدیریت هزینه می شود،اما همزمان به خاطر مشخص بودن هزینه جزء ها،کمی از سختی مدریت هزینه کاسته می شود.

توجه: در ایران، component های آماده عمدتا رایگان استفاده میشوند !

* Component based: مانند RAD، به خاطر استفاده از جزء های آماده،هزینه زیاد می شود که باعث سختی مدیریت هزینه می شود،اما همزمان به خاطر مشخص بودن قیمت جزء ها،کمی از سختی مدریت هزینه کاسته می شود.
* Reuse oriented: باز هم مانندRAD و Component based ، به خاطر استفاده از جزء های آماده،هزینه زیاد می شود که باعث سختی مدیریت هزینه می شود، وهمچنین در قسمت های دیگر آن،درگیر پیکر بندی و یا ساختن و پیدا کردن جزء های جدیدی هستیم که باعث افزایش بیشتر هزینه و سختی در تقریب هزینه می شود، اما همزمان به خاطر مشخص بودن قیمت جزء ها،کمی از سختی مدریت هزینه کاسته می شود.
* Formal: به خاطر استفاده از زبان ریاضی در همه مراحل آن،این مدل پر هزینه است و همچنین چون این مدل تمرکزی روی مدیریت هزینه ندارد، مدریت آن سخت است.
* Aspect oriented: با توجه به اینکه core-concern ها دسته بندی می شوند و به صورت جدا توسعه داده(که از مشکلاتی مانند tangling و scattering جلوگیری می کند و باعث کاهش هزینه توسعه می شود(می شوند،به صورت نسبتا راحتی می توان هزینه لازم برای توسعه آن ها را تخمین زد و برای این مدل به صورت نسبتا راحتی مدیریت هزینه داشت،اما در مقابل آن چون این مدل تکراری است، و پیدا کردن core-concern ها و cross-cutting concern ها کمی هزینه براست،کمی در تقریب هزینه و مدیریت هزینه دقت را از دست می دهیم.
* The unified process: به دلیل استفاده از UML ، مستند سازی بین هر دو قدم روند و ساخت نمونه های اولیه،از خطا ها تا حدی جلوگیری می شود (که باعث کاهش هزینه می شود)، اما چون این روند کند است و به صورت تکراری می باشد،و همچنین در آن نمونه های اولیه به طور مکرر در فاز های inception و elaboration تولید می شوند،هزینه توسعه زیاد است و همچنین تقریب زدن و مدیریت هزینه لازم برای گام های روند وپایان پروژه سخت می شود.

**Time Management**

با توجه به اینکه کوتاهی زمان برای عملی کردن این ایده و ارائه آن به بازار برای ما حائز اهمیت می باشد ، استفاده از مدل هایی که در آن ها کنترل و مدیریت زمان به آسانی انجام می شود برای ما بسیار مطلوب است.

* Code and fix: هیچ تقریبی برای زمان پایان پروژه و زمان پیاده سازی قسمت های پروژه در آن موجود نیست و همچنین به خاطرتعداد زیاد چرخه های آن، مدریت زمان بسیار سخت است.
* The waterfall: زمان پایان پروژه در مرحله اول روند مشخص می شود و با توجه به دانستن اینکه هر قدم ازروند چه قدر طول می کشد، مدریت زمان به راحتی امکان پذیر است.
* V-model: در ابتدا تقریبی از زمان پایان پروژه و زمان لازم برای هر یک از قدم های روند می توان به دست آورد،اما با توجه به اینکه پروژه ساخته شده از اصول مهندسی به درستی استفاده کرده باشد و تعداد کل تست ها و تست های شکست خورده زیاد باشد، این تقریب دقیق نیست و به خاطر آن مدریت زمان کمی سخت شود.
* Incremental: با برنامه ریزی دقیق،می توان تقریب خوبی از زمان تحویل جزء ها داشت و در مدیریت زمان دچار مشکل نشد.
* Prototyping: چون روندی تکراری می باشد،و همچنین در آن نمونه های اولیه به صورت مکرر تولید می شود،برای همین این روند کند بوده وتقریب دقیقی برای زمان در آن امکان پذیر نمی باشد و همچین مدیریت زمان مشکل است.
* Spiral: با scheduling،می توانیم تقریبی برای زمان داشته باشیم و مدیریت زمان را در هر چرخه انجام دهیم، اما باز چون روندی تکراری می باشد، تعداد چرخش ها زیاد است و نمی توانیم تقریب زمانی مناسبی برای پایان پروژه داشته باشیم و مدیریت زمان مشکل است.
* Concurrent: در آن هیچ تقریبی و مدیریتی برای زمان مشخص نشده است.
* RAD: چون در آن از جزء (component) های آماده استفاده می شود ،مدریت و تقریب زدن زمان نسبتا راحت می باشد.
* Component based: مانند RAD، چون در آن از جزء (component) های آماده استفاده می شود ،مدریت و تقریب زدن زمان نسبتا راحت می باشد.
* Reuse oriented: باز هم مانندRAD و Component based ، چون در آن از جزء(component (های آماده استفاده می شود ،مدریت و تقریب زدن زمان امکان پذیر می باشد،اما با توجه به اینکه در قسمت های دیگر آن،درگیر پیکر بندی و یا ساختن و پیدا کردن component های جدیدی هستیم،نسبت به RAD و Component based مدریت زمانی سختر می باشد.
* Formal: به خاطر استفاده از زبان ریاضی در همه مراحل آن،این مدل زمان بر است و به راحتی نمی توان مدیریت زمان را درآن انجام داد و تقریبی برای زمان پایان پروژه در آن داشت.
* Aspect oriented: با توجه به اینکه core-concern ها دسته بندی می شوند و به صورت جدا توسعه داده (که از مشکلاتی مانند tangling و scattering جلوگیری می کند و باعث کاهش زمان توسعه می شود) می شوند ، می توان در این مدل زمان را مدیریت کرد،اما چون این مدل تکراری است، و پیدا کردن core-concern ها و cross-cutting concern ها کمی وقت گیر است،اندکی دقت در تقریب زمان و توان در مدیریت آن از دست داده می شود.
* The unified process: به دلیل استفاده از UML و مستند سازی بین هر دو قدم روند، و همچنین به خاطر تولید نمونه های اولیه مکرردر فاز های inception و elaboration ، این روند کند است و چون به صورت تکراری می باشد، درنتیجه تقریب زمان لازم برای گام های روند وپایان پروژه،و همچنین مدیریت زمان سخت است.

**Adoptability**

با توجه به طبیعت پروژه و اینکه به آن نیازمندی ها و ویژگی ها جدید اضافه می شود(که باعث تغییرات در پروژه می شود)،برای ما مهم است که از مدلی استفاده کنیم که به تغییرات به خوبی واکنش نشان می دهد و آن را به راحتی قبول کند.

* Code and fix: در مقابل تغییرات پایدار است و آن ها را تا حد خوبی می تواند کنترل کند.
* The waterfall: به خاطر عدم توانایی بازگشت به قدم های قبل روند (مگر اینکه مدل iterative باشد) ، انعطاف پذیری آن بسیار پایین است.
* V-model: به خاطر تست های زیاد لازم برای اعمال تغییرات و اضافه کردن قسمت های جدید به پروژه،و اینکه در صورت شکست تست ها،باید تغییراتی در پروژه ایجاد کنیم و دوبار تست نویسی کنیم،در مقابل تغییرات پایدار نیست،که البته به خاطر تکراری بودن آن ،این ناپایداری کمی جبران می شود.
* Incremental: چون پروژه به صورت جزء هایی تحویل داده می شود و این جزء ها می توانند به صورت موازی توسعه پیدا کنند،این مدل انعطاف پذیری بالایی دارد.
* Prototyping: با توجه به ساخت مکرر نمونه اولیه و دخالت کاربر در توسعه پروژه،ریتم تیم توسعه به هم می خورد (دخالت مداوم کاربر به دلیل مشاهده کردن نمونه اولیه و ارایه درخواست های جدید) و باعث کاهش انعطاف پذیری می شود،اما به خاطر تکراری بودن این مدل،این کاهش کمی جبران می شود.
* Spiral: به خاطر تکراری بودن این مدل،Spiral نسبت به تغییرات تطبیق پذیر است،که کمی از تطبیق پذیری آن،به خاطر آنالیز ریسک در هر چرخه آن (که در مواقعی ممکن باعث ایجاد مقاومت به تغییرات شود) ، کاهش می یابد.
* Concurrent: نسبت به تغییرات انعطاف پذیر است (چرا که هر task ای بعد از تمام شدن منتظر تغییرات است) ودر آن همه اعضای تیم توسعه از تغییرات با خبر هستند،که باعث کاهش نا پایداری پروژه در مقابل تغییرات می شود.
* RAD: به خاطر استفاده از جزء های اماده (که تغییر دادن آن ها سخت است) ،نسبت به تغییرات انعطاف پذیر نیست و همچنین به خاطر ساخت مکرر نمونه اولیه و دخالت کاربر در توسعه پروژه،ریتم تیم توسعه کمی به هم می خورد که باعث نپایداری بیشتر پروژه در مقابل تغییرات می گردد.
* Component based: مانند RAD، به خاطر استفاده از جزء های اماده (که تغییر دادن آن ها سخت است) ،نسبت به تغییرات انعطاف پذیر نیست.
* Reuse oriented: باز هم مانند RAD و Component based ، به خاطر استفاده از جزء های اماده(که تغییر دادن آن ها سخت است )،نسبت به تغییرات انعطاف پذیر نیست،اما به دلیل اینکه در این مدل ساخت جزء های جدید امکان پذیر است،کمی از انعطاف ناپذیری آن جبران می شود.
* Formal: به خاطر استفاده از زبان ریاضی در همه مراحل آن،و اینکه تغییرات باید به صورت زبان ریاضی اعمال شوند،انعطاف پذیری پایینی نسبت به تغییرات دارد.
* Aspect oriented: با توجه به اینکه core-concern ها دسته بندی می شوند و به صورت جدا توسعه داده (که از مشکلاتی مانند tangling و scattering جلوگیری می کند و باعث راحتی در اعمال تغییرات میشود) می شوند،و همین اینکه چون این مدل تکراری است،از انعطاف پذیری بالایی برخوردار است.
* The unified process: به دلیل استفاده از UML و مستند سازی بین هر دو قدم روند،تغییرات هزینه بر هستند و نسبت به تغییرات مقاوت خواهیم داشت (که باعث کاهش انعطاف پذیری می شود) ، همچنین تولید نمونه های اولیه مکرر در فاز های inception و elaboration باعث دخالت زیاد مشتری می شود،که ریتم تیم توسعه به هم می خورد و باعث کاهش بیشتر انعطاف پذیری می شود،و در کل در این مدل انعطاف پذیری پایین است،. اما به خاطر تکراری بودن این مدل،این پایین بودن کمی جبران می شود.

**Security Management**

با توجه به اینکه کاربران با اسفاده از حساب هایی با این نرم افزار حاصل از پروژه کار می کنند،که این حساب ها شامل اطلاعات خصوصی کاربر(مانند نام کاربری،پسورد،آدرس و...) می شود،و حفاظت از این اطلاعت مهم می باشد(که اگر امنیت برنامه پایین باشد، دزدیده می شوند)،و همچنین امکان حملات هکری به قسمت هایی از برنامه(مانند قسمت وارد شدن به حساب با ارسال requestهای زیاد به سمت سرور مثل حملات DDoS و ... ) وجود دارد( که اگر امنیت برنامه پایین باشد، این حملات موفق می شوند) و درنهایت به خاطر طبیعت پروژه،که درآن نیازمندی هایی جدیدی به طور مکرر روی برنامه اعمال می شوند(که با اعمال هر تغییر،باید مطمین شویم که امنیت برنامه پایین نیاید)،باید از مدل هایی استفاده شود که در آن امنیت برنامه به راحتی کنترل و مدیریت می شود.

* Code and fix: برای امنیت و مدیریت آن روشی در نظر نمی گیرد.
* The waterfall: به خاطر عدم توانایی بازگشت به قدم های قبل روند (مگر اینکه مدل iterative باشد)،و در نتیجه جمع آوری تمام نیاز ها و ویژگی های لازمه پروژه در مرحله اول،و مستند سازی قوی بین هر دو گام روند،دارای امنیت بالایی می بشد و مدیریت امنیت در آن راحت است.
* V-model: به خاطر تست های زیاد (که شامل تست های امنیتی هم می شود)،و اینکه در صورت شکست تست ها ،باید تغییراتی در پروژه ایجاد کنیم و در صورت لزوم امنیت را بالا ببریم،امنیت بالا و مدیریت ان راحت است.
* Incremental: چون در اخر توسعه هر جزء،تست های اعتبار و صحت سنجی انجام می شوند،که باعث می شود ببنیم امنیت همه جزء ها بالا هستند وامنیت آن ها را مدیریت کنیم و در نتیجه فقط امنیت جزء ها را کنار هم در انتها پروژه در نظر نگیریم (مانند waterfall) ،مشکلات امنیتی کاهش می یابند (چون خیلی از مشکلات امنیتی در خود جزء ها بهتر دیده شده و راحتر حل می شوند) و در نتیجه امنیت بالا تر می رود و مدیریت آن سهولت پیدا می کند.
* Prototyping: با توجه به ساخت مکرر نمونه اولیه و دخالت کاربر در توسعه پروژه،نقاط ضعف و گنگ شناسایی می شوند،که باعث افزایش اندک امنیت و راحتی در مدیریت آن می شود.
* Spiral: به خاطر تکراری بودن این مدل و آنالیز ریسک در هر چرخه ، باعث کاهش مشکلات امنیتی می شود و باعث مدیریت بهتر امنیت در پروژه است.
* Concurrent: برای ریسک و مدیریت آن روشی در نظر نمی گیرد.
* RAD: به خاطر استفاده از جزء های اماده (که تست شده اند و امنیت ان ها بالا است) ، و همچنین به خاطر ساخت مکرر نمونه اولیه و دخالت کاربر در توسعه پروژه (که باعث شناسایی نقاط ضعف و گنگ می شود) ، از امنیت نسبتا بالایی برخوردار است و مدیریت امنیت در آن مشکل نیست.
* Component based :مانند RAD، به خاطر استفاده از جزء های اماده(که تست شده اند و امنیت ان ها بالا است)، و همچنین به خاطر ساخت مکرر نمونه اولیه و دخالت کاربر در توسعه پروژه( که باعث شناسایی نقاط ضعف و گنگ می شود)،از امنیت نسبتا بالایی برخوردار است و مدیریت امنیت در آن مشکل نیست .
* Reuse oriented: باز هم مانند RAD و Component based ، به خاطر استفاده از جزء های اماده(که تست شده اند و امنیت ان ها بالا است)،از امنیت برخوردار است ،اما به خاطر اینکه بعضی از جزء ها ممکن هست آماده نباشند و ساخته شوند،مدیریت امنیت کمی سختر می شود.
* Formal: به خاطر استفاده از زبان ریاضی در همه مراحل آن(که باعث کاهش گنگی،ناکاملی و تناقض در تعریف پروژه می شود)،و اینکه تغییرات باید به صورت زبان ریاضی اعمال شوند و همچنین چون از قواعد

cleanroom software engineering استفاده می شود،باعث افزایش زیاد امنیت و همچنین سهولت در مدیریت امنیت می شود.

* Aspect oriented: با توجه به اینکه core-concern ها دسته بندی شده و به صورت جدا توسعه داده (که از مشکلاتی مانند tangling و scattering جلوگیری می کند و مشکلات امنیتی را کاهش می دهد) می شوند و همچنین دسته بندی cross-cutting concern(که امنیت شامل آن ها می شود و باعث می شود که بتوان آن ها را به صورت جدا توسعه ومدیریت کرد)،این مدل باعث ایجاد امنیت زیاد و راحتی در مدیریت امنیت می شود .
* The unified process: به دلیل استفاده از UML و مستند سازی بین هر دو قدم روند (که باعث می شود اثر تغییرات در پروژه بهتر دیده شوند و در نتیجه مشکلات امنیتی کاهش یابند) و همچنین تولید نمونه های اولیه مکرردر فاز های inception و elaboration (که باعث می شود نقاط ضعف و گنگ شناسایی شوند) ، از خطا های امنیتی به خوبی جلوگیری می شود،که باعث افزایش زیاد امنیت و سهولت مدیریت آن می شود.

**Risk Management**

با توجه به complexity و بزرگی پروژه،که باعث ایجاد ریسک های مختلفی در طول توسعه برنامه می شود،ما به دنبال مدلی هستیم که بتواند این ریسک ها را کاهش دهند و آن ها را به نحوه ی خوبی مدیریت کند،چرا که با مدیریت درست ریسک ها ،هزینه توسعه کاهش می یابد،و چون بودجه پروژه محدود و کم می باشد(با توجه به اینکه پروژه،یک پروژه استارت آپ است)،این برای ما بسیار مهم است. ( انتخاب راه حل مناسب)

* Code and fix: برای ریسک و مدیریت ان روشی در نظر نمی گیرد.
* The waterfall: به خاطر عدم توانایی بازگشت به قدم های قبل روند(مگر اینکه مدل iterative باشد)،و در نتیجه جمع آوری تمام نیاز ها و ویژگی های لازمه پروژه در مرحله اول(که شامل خطا های ممکن و ریسک ها می شود)،و مستند سازی قوی بین هر دو گام روند،دارای ریسک پایین می باشد و مدیریت ریسک در آن راحت است،اما چون این ریسک وجود دارد که پروژه با نیازمندی های مشتری همخوانی نداشته باشد(که این ریسک بزرگی است،با توجه به ماهیت پروژه)ریسک اندکی افزایش می باید و مدیریت ان اندکی سخت می شود.
* V-model: به خاطر تست های زیاد در آن ،که باعث اگاهی از نقاط ضعف پروژه و بعضی از ریسک های مربوط به معماری نرم افزار حاصل از پروژه و کد آن می شود ، تا حد کمی باعث کاهش ریسک و سختی در مدیریت ریسک می شود.
* Incremental: چون در اخر توسعه هر جزء، دو تست اعتبار و صحت سنجی انجام می شوند،که باعث دیدن مشکلات هر جزء می شود، و در نتیجه فقط خطاهای جزء ها در کنار هم در نظر گرفته نمی شود(مانند قدم اخر waterfall (،خیلی از اشکال ها زود شناسایی می شوند وچون این خطا ها فقط در یک جزء هستند و نه در کل پروژه(که می توان راحتر آن ها را اصلاح کرد و از خطا های بیشتر در آینده جلوگیری کرد)، باعث کاهش خطا ها در کل پروژه می شود ،که تا حدی از ریسک ها جلوگیری کرده و در مدیریت ریسک کمک می کند،اما چون امکان دارد که در اول پروژه بخش بندی جزء ها و معماری به طور درست انجام نشده باشد،مزیت های آن کمی کاهش می یابد.
* Prototyping: با توجه به ساخت مکرر نمونه اولیه و دخالت کاربر در توسعه پروژه،نقاط ضعف و گنگ و همچینین خطا ها زودتر شناسایی می شوند،که باعث کاهش ریسک و سختی در مدیریت آن می شود.
* Spiral: به خاطر تکراری بودن این مدل و تمرکز این مدل بر ریسک و آنالیز ریسک در هر چرخه آن، ریسک کاهش و مدیریت آن به آسانی انجام می شود.
* Concurrent: برای ریسک و مدیریت ان روشی در نظر نمیگیرد.
* RAD: به خاطر استفاده از جزء های آماده، این ریسک وجود دارد که جزء های آماده همه نیازمندی ها را برآورده نکنند،که باعث افزایش ریسک در پروژه و سختی مدریت آن می شود، اما به خاطر ساخت مکرر نمونه اولیه و دخالت کاربر در توسعه پروژه (که باعث شناسایی نقاط ضعف و گنگ و خطا ها می شود) ، کمی از افزایش ریسک جبران می شود و مدیریت آن اندکی سهولت می یابد.
* Component Based: مانند RAD، به خاطر استفاده از جزء های آماده، این ریسک وجود دارد که جزء های آماده همه نیازمندی ها را برآورده نکنند،که باعث افزایش ریسک در پروژه و سختی مدریت آن می شود ،اما به خاطر اینکه رابطه بین این جزء ها و معماری برنامه ای که این جزء در آن قرار خواهند گرفت مورد توجه بیشتری قرار می گیرند، کمی از افزایش ریسک جبران می شود و مدیریت اندکی سهولت می یابد.
* Reuse oriented: باز هم مانند RAD و Component based ، به خاطر استفاده از جزء های آماده، این ریسک وجود دارد که جزء های آماده همه نیازمندی ها را برآورده نکنند،که باعث افزایش ریسک در پروژه و سختی مدریت آن می شود ،اما به خاطر اینکه در صورت نیاز جزء هایی جدید ساخته می شود ،بخشی از ریسک جبران می شود (البته با در نظر گرفتن ریسک وجود خطا ها در جزء های جدید) و مدریت آن کمی آسان می گردد .
* Formal: به خاطر استفاده از زبان ریاضی در همه مراحل آن (که باعث کاهش گنگی،ناکاملی و تناقض در تعریف پروژه می شود)،و همچنین استفاده از قواعد cleanroom software engineering ،و خوش فرم بودن زبان فرمال در آن(که باعث می شود رابطه های بین نیازمندی ها به راحتی مشخص باشد)،و درنتیجه دیده شدن بهتر اثر تغییرات در پروژه،ریسک آن کم بود و مدیریت ریسک به راحتی امکان پذیر است .
* Aspect oriented: با توجه به اینکه core-concern ها دسته بندی می شوند و به صورت جدا توسعه داده(که از مشکلاتی مانند tangling و scattering جلوگیری می کند و خطا ها را کاهش می دهد (و همچنین دسته بندی cross-cutting concernها (که باعث می شود که بتوان آن ها را به طور مستقل توسعه ومدیریت کرد و چون خیلی از مشکلات امنیتی و دیگر خطا ها از تغییرات در این concern ها ایجاد می شود ، باعث می شود که این مشکلات و خطا ها کاهش بیابند) ،این مدل باعث کاهش ریسک ها به طور خوبی(مخصوصا اگر این ریسک ها مربوط به cross-cutting concern ها باشد) می شود و در مدریت ریسک تا حد مناسبی کمک می کند.
* The unified process:به دلیل استفاده از UML و مستند سازی بین هر دو قدم این روند و تولید نمونه های اولیه مکرر(که باعث پیدا شدن نقاط ضعف و گنگ و باعث کاهش خطا ها می شود) ،از خطا ها تا حدی تا حد خوبی جلوگیری می شود (که در کاهش ریسک اثر دارد) و باعث اطلاع بیشتری از پروژه و اثر تغییرات در آن می شود(که باعث سهولت در مدیریت ریسک می شود)،و همچینین چون که در فاز inception هم از ارزیابی ریسک اسفتفاده می شود،در کل این مدل باعث افزایش ریسک پذیری و آسانی در مدیریت ریسک می شود.

**user Interference**

به خاطر ماهیت پروژه(که complex است و ذات adoptive دارد)،و اینکه این پروژه مدام به بازخورد و دخالت(interference ) کاربر نیازمند است(چرا که رضایت و درنتیجه بازخورد کاربر برای ما مهم است)،برای همین به مدل هایی نیازمندیم که در آن دخالت مشتری در توسعه تا حدی بالا است.

* Code and fix: میزان دخالت کاربر در آن مشخص نیست.
* The waterfall: به خاطر جمع آوری تمام نیاز ها و ویژکی های لازمه پروژه در مرحله اول،میزان دخالت کاربر در آن بسیار کم است.
* V-model: به خاطر آزمون قبولی و مدل کردن نیازمندی ها،دخالت کاربر به طور کم می باشد.
* Incremental: به خاطرارتباطات با مشتری و تعریف نیازمندی ها در ابتدای توسعه هر جزء و همچنین تولید جزء ها که کاربر می تواند از آن ها استفاده کند و بازخورد بدهد ،دخالت کاربر به طور متوسط می باشد.
* Prototyping: با توجه به جمع آوری نیازمندی ها و ساخت نمونه اولیه در هر چرخه آن،دخالت کاربر در آن بالا می باشد.
* Spiral: مانند incremental ، به خاطرارتباطات با مشتری و تعریف نیازمندی ها در ابتدای توسعه هر جزء و همچنین تولید جزء ها که کاربر می تواند از آن ها استفاده کند و بازخورد بدهد ،دخالت کاربر به طور متوسط می باشد.
* Concurrent: میزان دخالت کاربر در آن مشخص نیست.
* RAD: با توجه به جمع آوری نیازمندی ها (که یکبار در آغاز این مدل انجام می شود) و ساخت نمونه اولیه در هر چرخه User Design آن،دخالت کاربر در آن بالا می باشد.
* Component based: میزان دخالت کاربر در آن مشخص نیست.
* Reuse oriented: به خاطر جمع آوری نیازمندی ها و تصحیح کردن آن ها در مراحل ابتدایی این مدل، میزان دخالت کاربر در آن نسبتا کم است.
* Formal: میزان دخالت کاربر در آن مشخص نیست.
* Aspect oriented: میزان دخالت کاربر در آن مشخص نیست.
* The unified process: به خاطرارتباطات با مشتری و تعریف نیازمندی ها در ابتدای هر چرخه آن،و همچنین به خاطر تولید نمونه های اولیه مکرر (که کاربر با آن ها کار می کند) در فاز های inception و elaboration،و در نهایت به خاطر جمع آوری بازخورد و گزارش های تست بتا در فاز transition،دخالت کاربر در آن بسیار بالا است.

**Management of premade Parts**

با توجه به محدودیت زمانی ،می خواهیم از مدل هایی استفاده کنیم که در آن امکان استفاده از جزء(component) های آماده(مانند سامانه پرداخت شاپرک در این پروژه) وجود دارد و مدیریت آن ها راحت می باشد،چرا که استفاده از این جزء ها باعث کاهش زمان توسعه(چرا که دیگر لازم نیست آن ها را بسیازیم) و افزایش امنیت در نرم افزار حاصل از پروژه( چون که این جزء ها توسط شرکت های معتبر ساخته شده وتوسط توسعه دهندگان نرم افزار زیادی استفاده شده اند، و اگر مشکلات امنیتی داشتند،از آن مشکلات با خبر می شدیم.) می شود.

* Code and fix: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* The waterfall: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* V-model: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* Incremental: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* Prototyping: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* Spiral: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* Concurrent: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* RAD: به خاطر ماهیت آن،استفاده از جزء های آماده در آن زیاد و مدیریت آن جزء ها راحت می باشد.
* Component based: مانند RAD،به خاطر ماهیت آن،استفاده از جزء های آماده در آن زیاد و مدیریت آن جزء ها راحت می باشد.
* Reuse oriented: مانند Component based وRAD،به خاطر ماهیت آن،استفاده از جزء های آماده نسبتا در آن نسبتا زیاد است اما به خاطر ساخته شدن و استفاده از جزءهای جدید در آن، از Component based وRAD کمتر می باشد،و همچنین مدیریت جزء های آماده در آن نیز کمی سختر می باشد.
* Formal: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* Aspect oriented: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.
* The unified process: روش استفاده و مدیریت از جزء های آماده در آن مشخص نیست.

**نتیجه گیری و انتخاب مدل فرآیند:** با توجه به جدول امتیازات،مدل فرایند **RAD**،بهترین فرآیند برای پروژه می باشد و جنبه های مورد اهمیت پروژه را به بهترین نحو پوشش می دهد،اما چون در بخش هایی از پروژه ممکن است لازم به ساخت جزء های جدیدی باشیم(که در RAD پوشش داده نمی شود)،بهتر است از مدل Incremental هم در کنار مدل RAD برای ساخت جزء های جدید استفاده کنیم.

# **آنالیز نیازمندی های مبتنی بر سناریو**

## **Use-Case Diagram**

پروزه را در 3 سطح بررسی کردیم :

* **سطح کلی** : در این سطح فقط به use case های کلی پرداخته شده است و جزییات در نظر گرفته نشده اند .
* **سطح** **متوسط** : در اینجا یک سطح پایین تر میرویم و به بررسی use case ها می پردازیم .
* **سطح پایین** : در این سطح use case ها را با جزییات بیشتری بررسی میکنیم.

**High level use case diagram**

تصویر

**Middle level use case diagram**

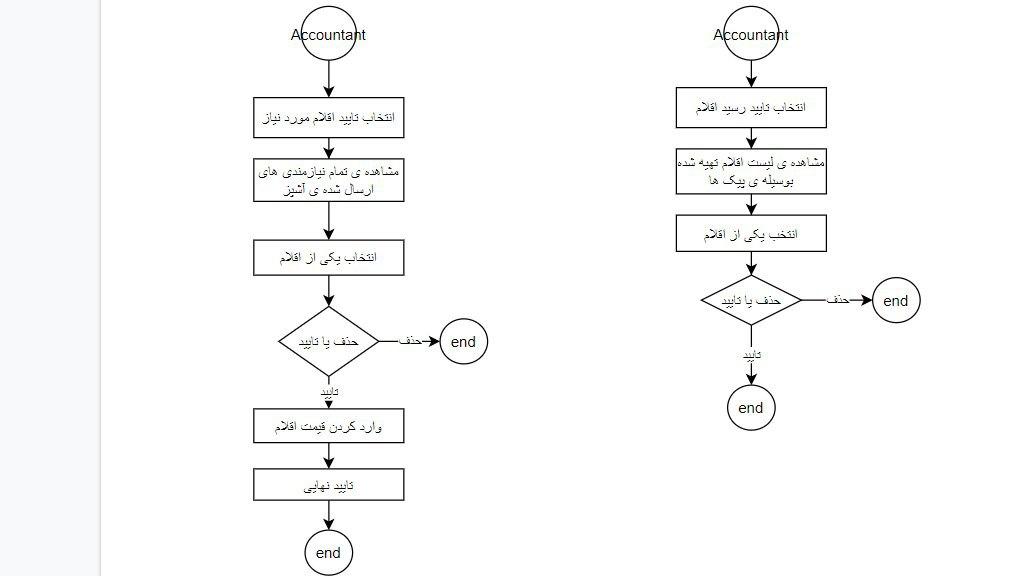
تصویر

**Low level use case diagram**

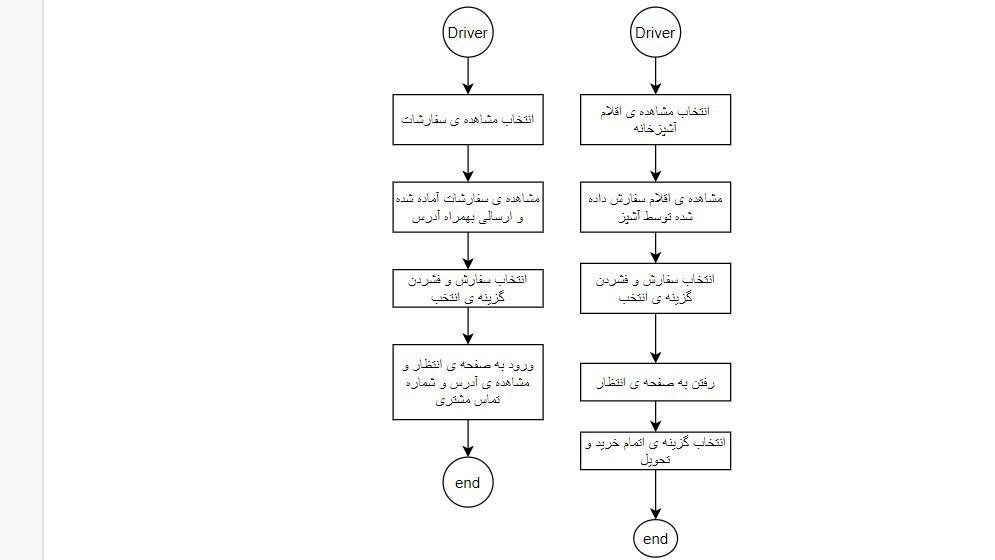
تصویر

## **Activity Diagram**

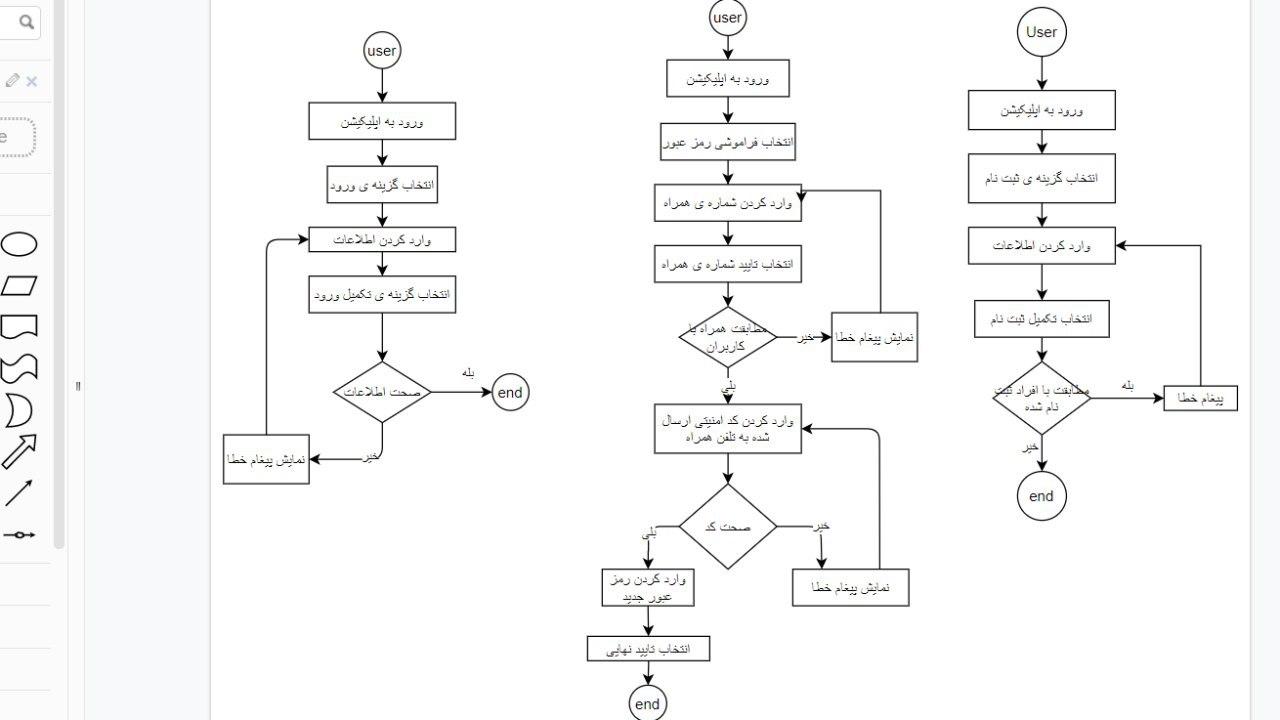
**نمودار حسابدار**



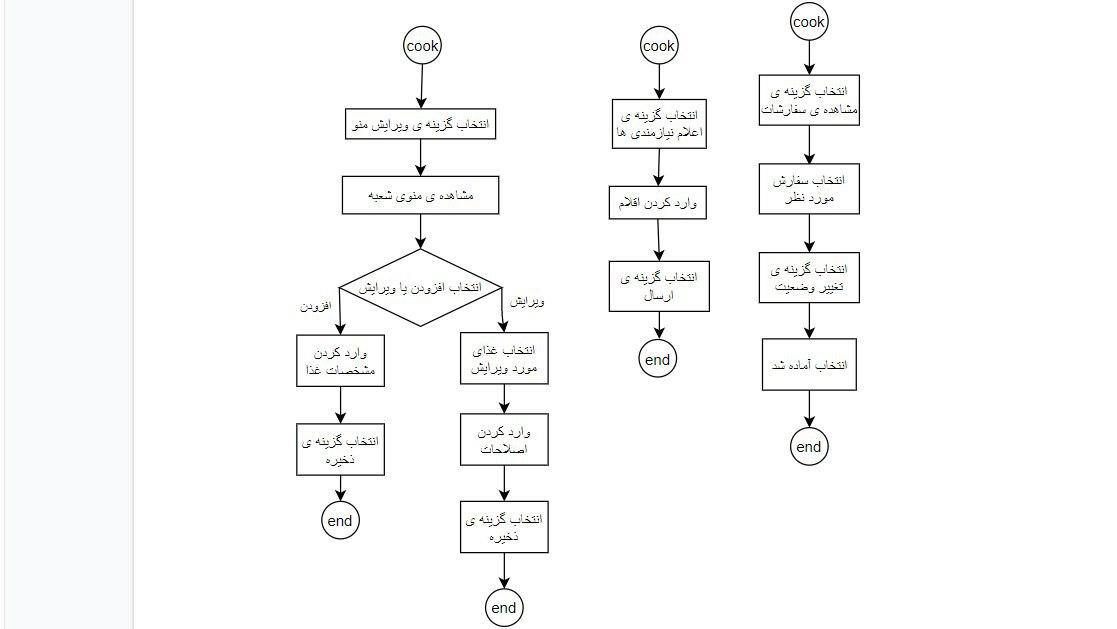
**نمودار پیک**



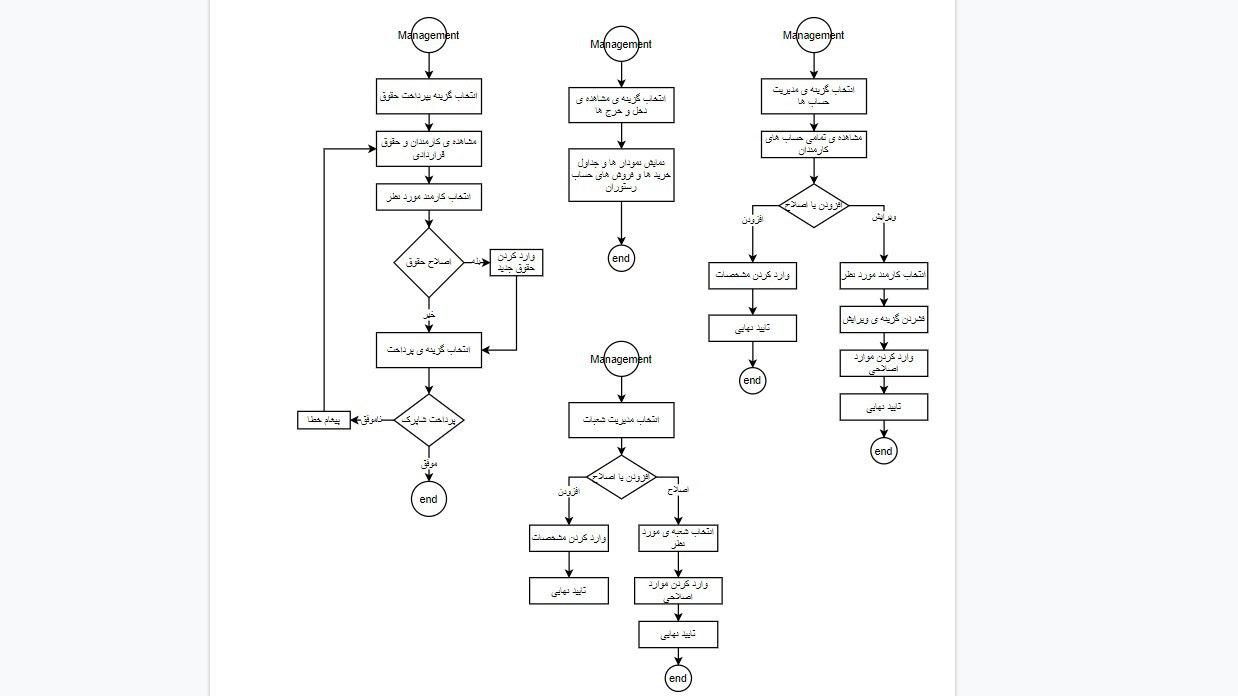
**نمودار کاربر(User) ]در حالت عام[**



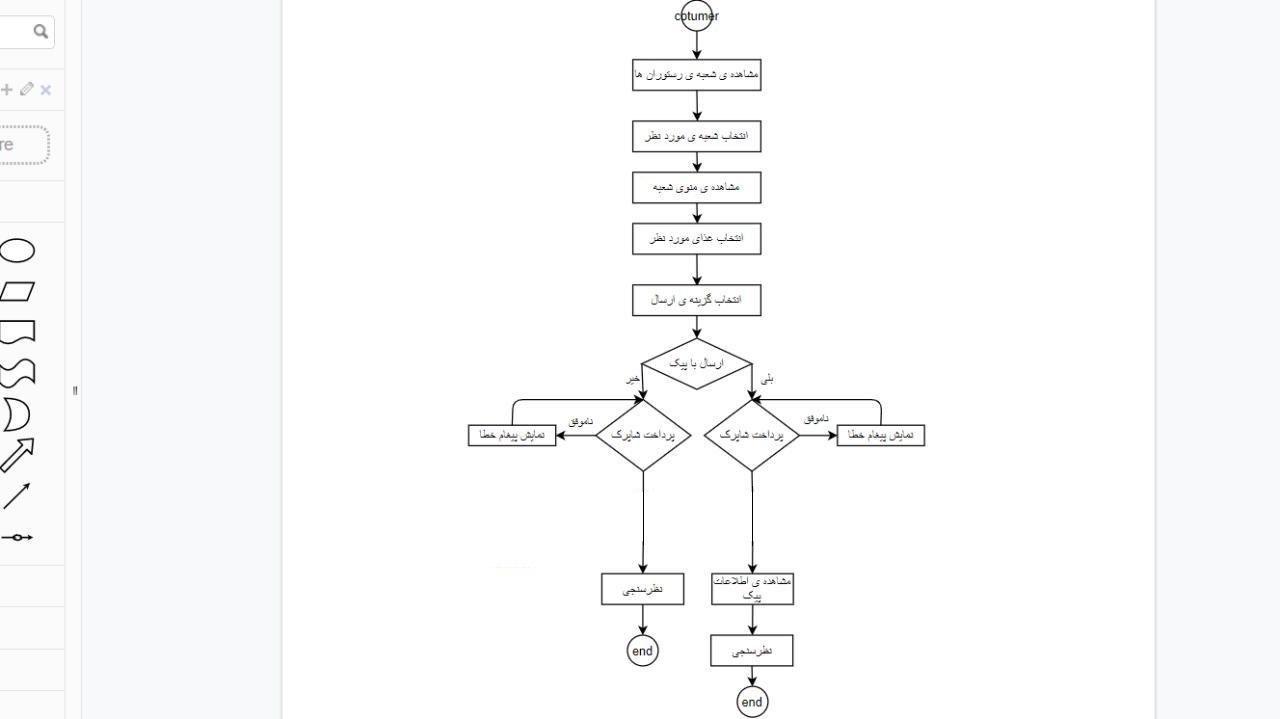
**نمودار آشپز**



**نمودار مدیر**



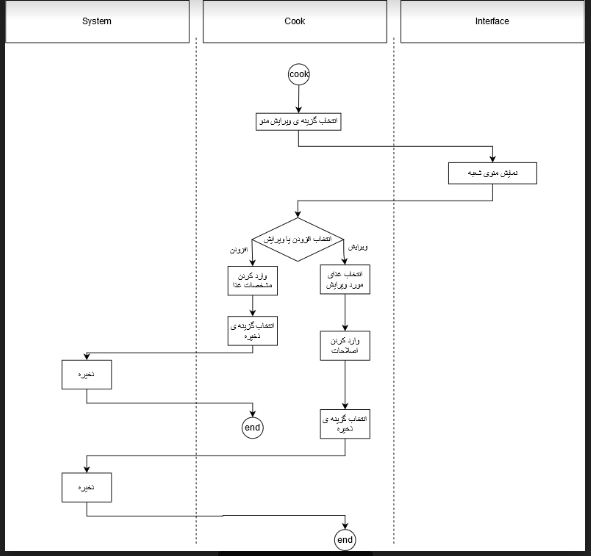
**نمودار مشتری**



# **Swimlane diagram**

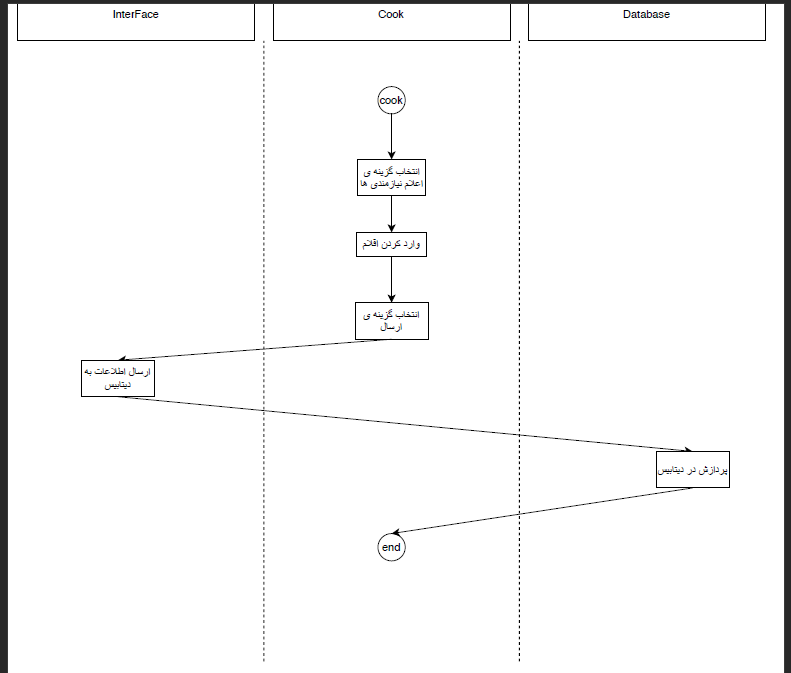
**نمودار شماره 1**

انتخاب گزینه ویرایش منو توسط آشپز



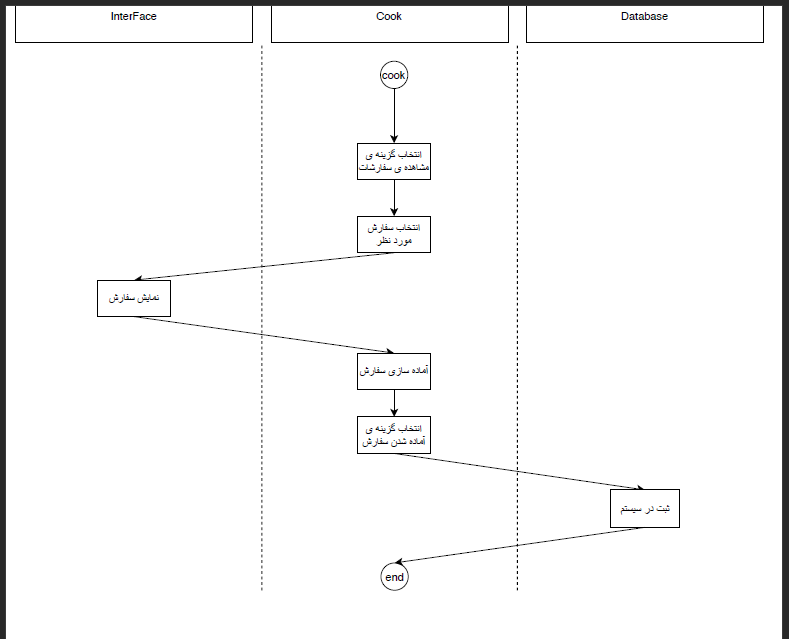
**نمودار شماره 2**

انتخاب گزینه اعلام نیازمندی ها توسط آشپز



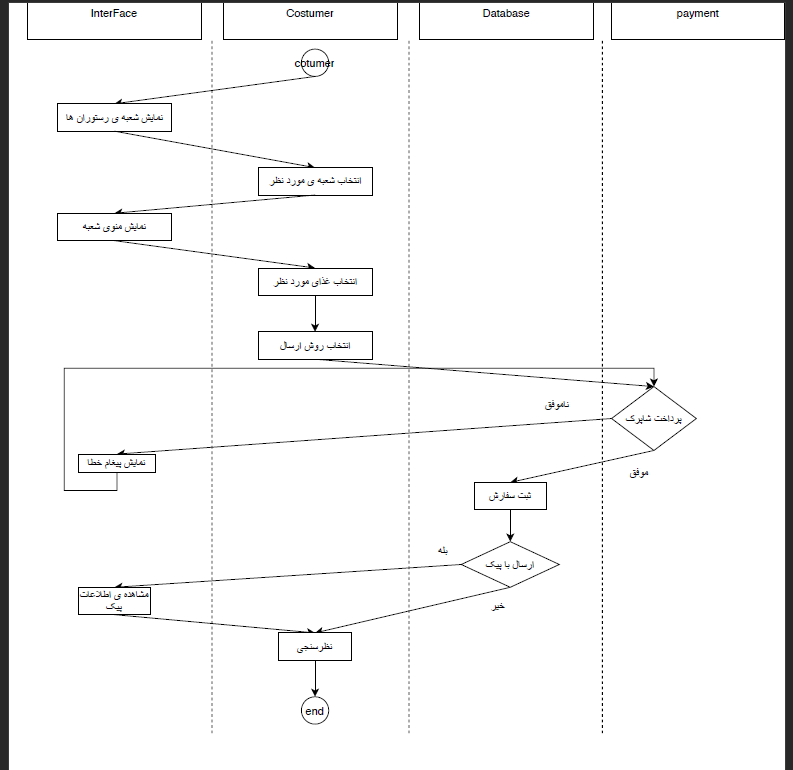
**نمودار شماره 3**

انتخاب گزینه مشاهده سفارشات توسط آشپز



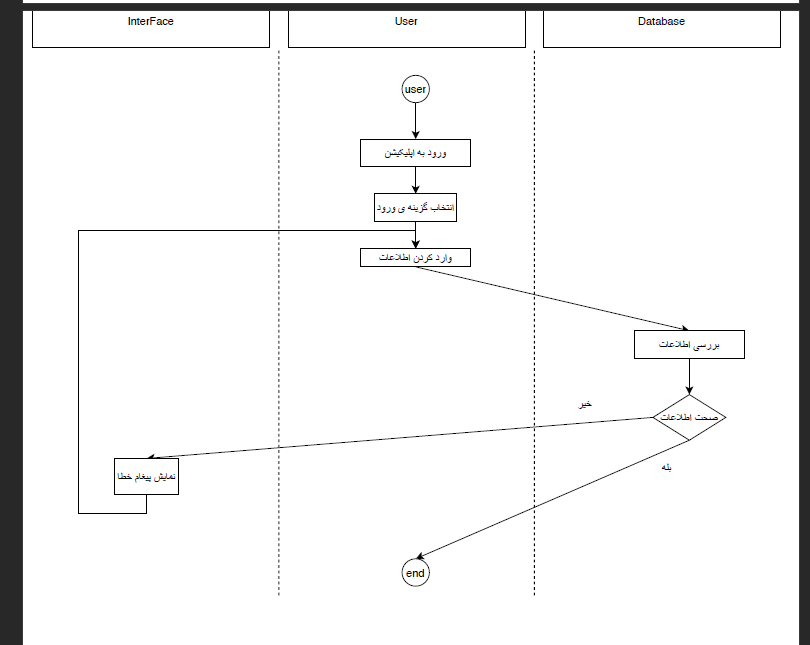
**نمودار شماره 4**

انتخاب شعبه و غذا از طریق منوی آن توسط مشتری



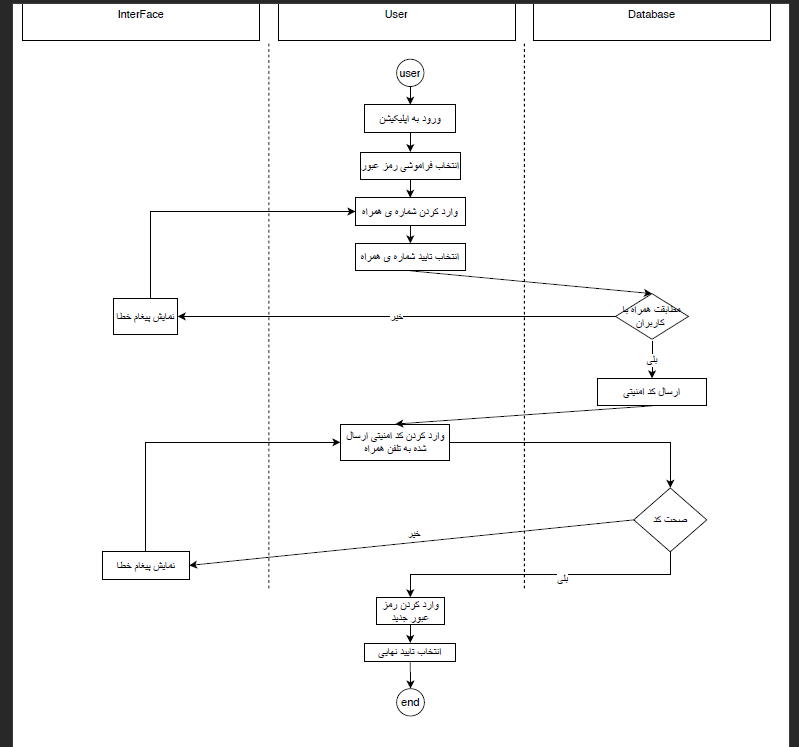
**نمودار شماره 5**

ورود به اپلیکیشن توسط کاربر سیستم(login)



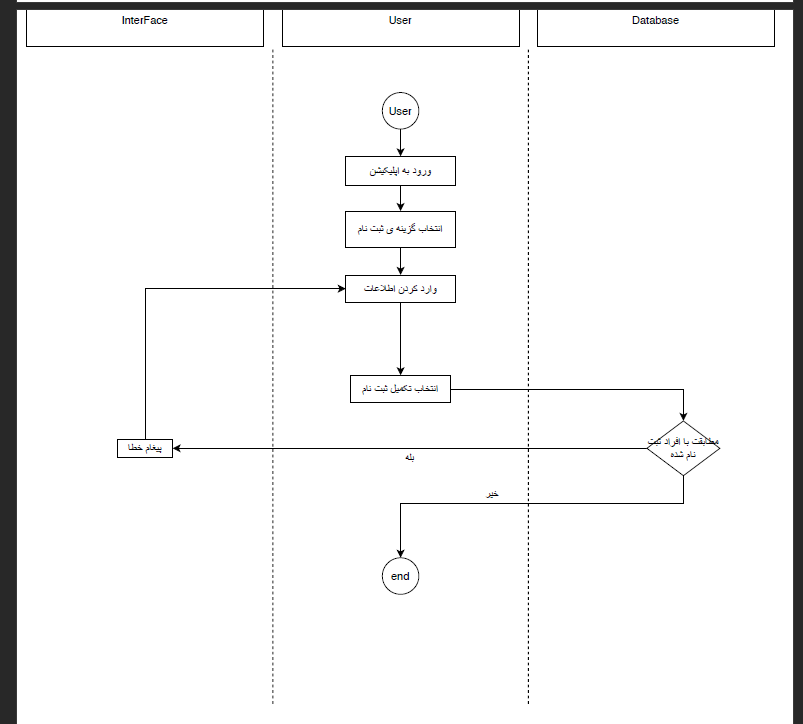
**نمودار شماره 6**

انتخاب گزینه فراموشی رمز عبور توسط کاربر سیستم



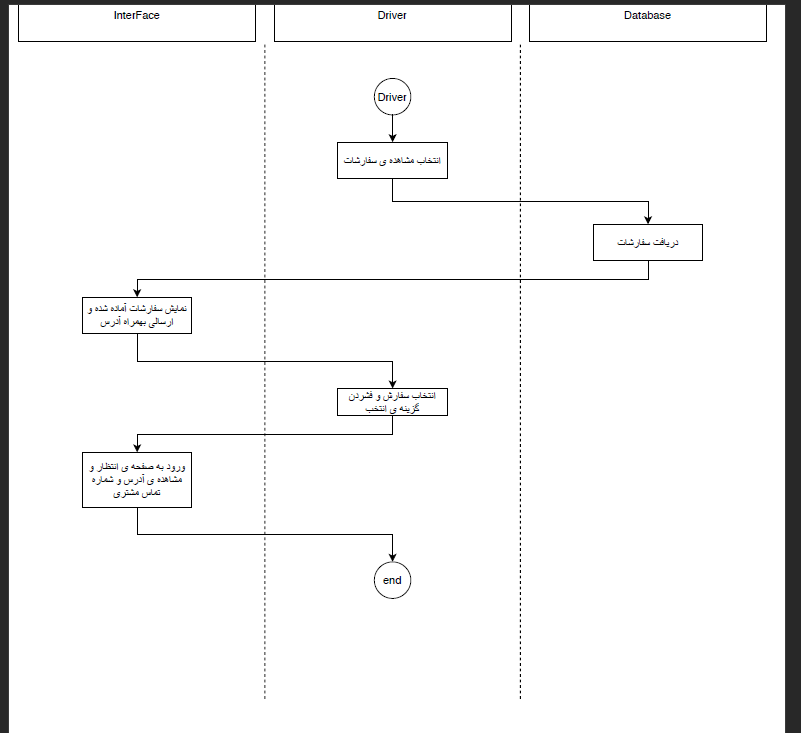
**نمودار شماره 7**

انتخاب گزینه ثبت نام توسط کاربر سیستم(sign up)



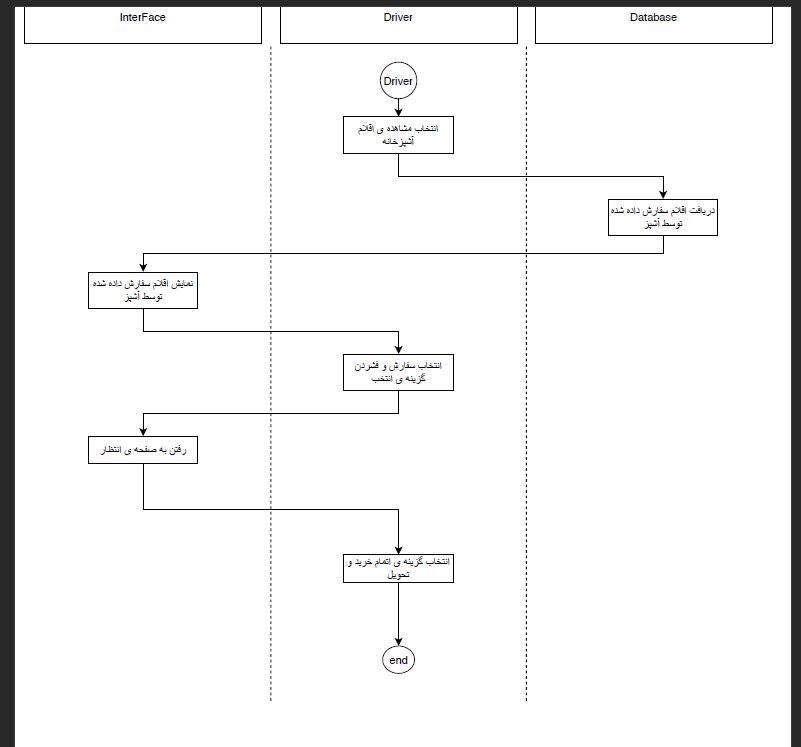
**نمودار شماره 8**

انتخاب گزینه مشاهده سفارشات (آماده ارسال) توسط پیک



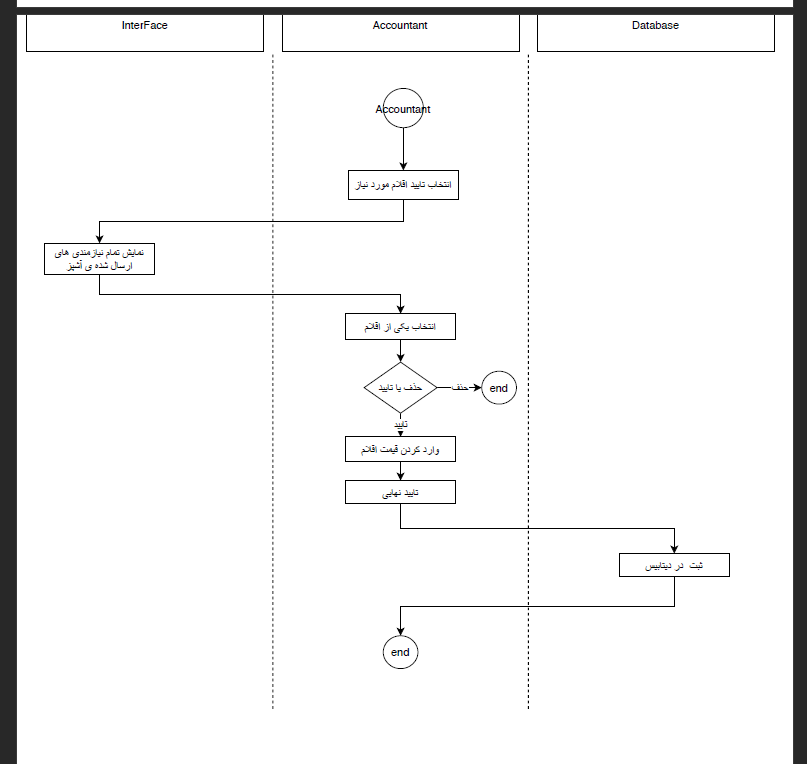
**نمودار شماره 9**

انتخاب گزینه مشاهده اقلام مورد نیاز آشپزخانه توسط پیک



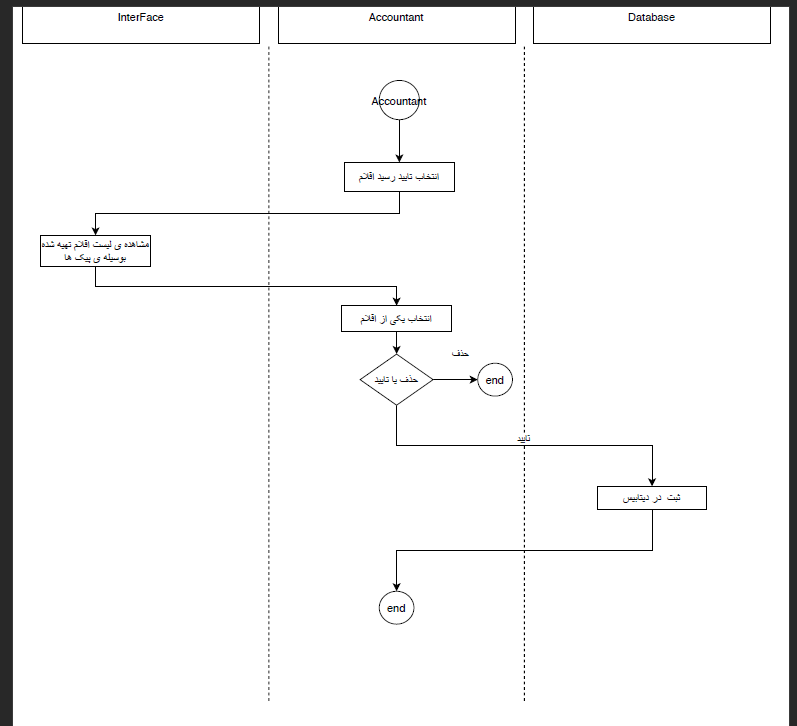
**نمودار شماره 10**

انتخاب گزینه تایید اقلام مورد نیاز آشپز توسط حسابدار



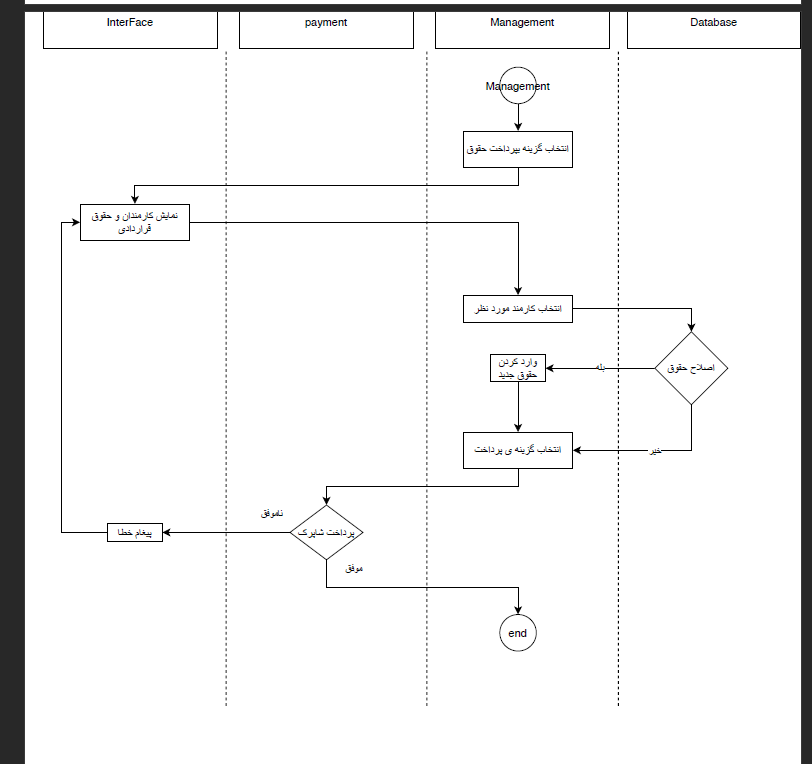
**نمودار شماره 11**

انتخاب گزینه تایید رسید اقلام از پیک به آشپز توسط حسابدار



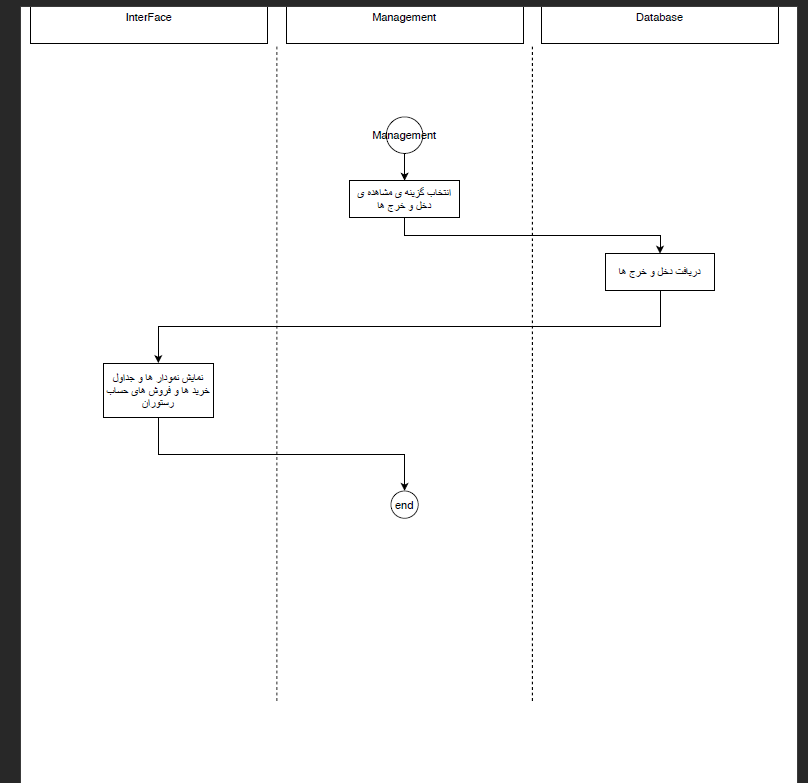
**نمودار شماره 12**

انتخاب گزینه پرداخت حقوق کارمندان توسط مدیر



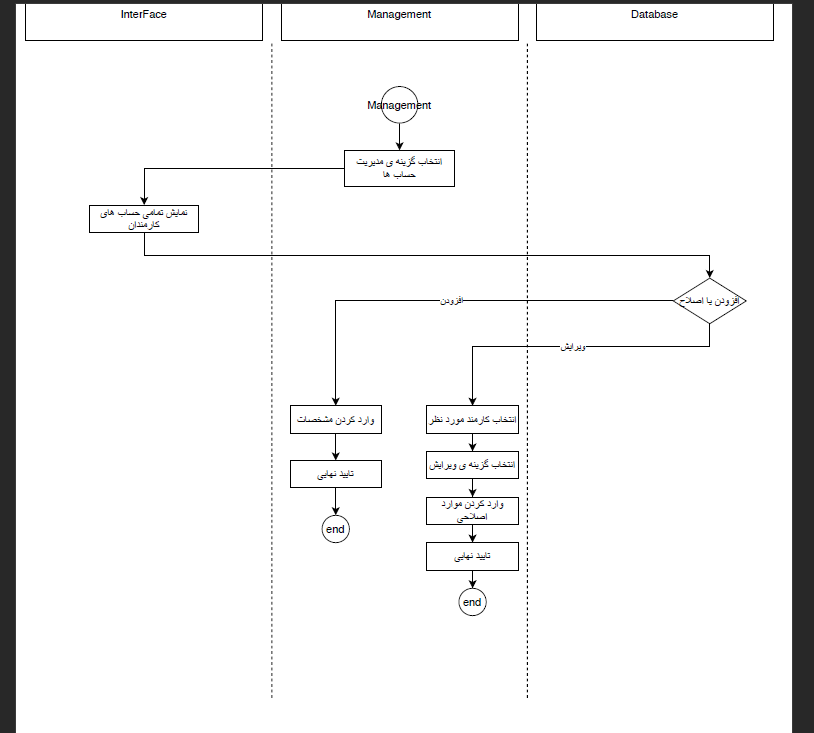
**نمودار شماره 13**

انتخاب گزینه مشاهده دخل و خرج ها توسط مدیر



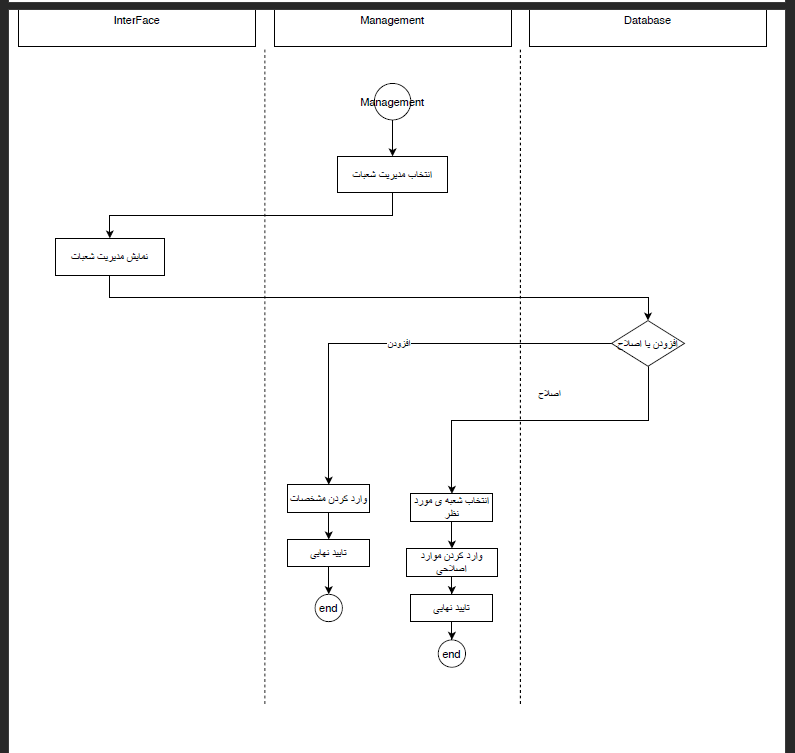
**نمودار شماره 14**

انتخاب گزینه مدیریت حساب کاربران توسط مدیر



**نمودار شماره 15**

انتخاب گزینه مدیریت شعبات توسط مدیر



# **Technical Review**

* شماره : 1
* مسیول جلسه : سعید معروف
* تاریخ جلسه : 28/04/1399
* نظرات و جمع بندی ها :

**جدای از اشتباهات تایپی که با ریزبینی دوستان تشخیص داده شد و تصحیح شد ؛ تغییراتی در متن نیازمندی ها و تصمیماتی اتخاذ شد که بر پروژه تاثیراتی گذاشت. در ادامه بطور مفصل به این تصمیمات خواهیم پرداخت.**

* + **در ابتدا بخش حقوق کارمندان مورد بحث و بررسی قرار گرفت که بنظر جمع ، دارای اشکالات و نقاط ضعفی بود. اصلی ترین نقطه ی ضعف این بود که در ابتدا قرار بود مدیر بتواند حقوق افراد را تعیین و پرداخت کند اما با توجه به اینکه حسابدار از دخل و خرج های شرکت بیشتر با خبر است و ورودی ها و خروجی ها را تحلیل و مشاهده میکند ، بهتر است حسابدار در این بخش نقشی ایفا کند. یکی از دوستان پیشنهاد داد که پس بهتر است بخش "اصلاح و پرداخت حقوق کارمندان" بطور خاص در دسترس حسابدار باشد اما از طرف دیگر اینکه پرداخت حقوق کارمندان در دست حسابدار صرف باشد درست نیست چراکه مدیر باید حرف اول و آخر را در این مورد بزند و تصمیم نهایی را اتخاذ کند و حقوق را پرداخت کند. فلذا بهتر است حسابدار نیز در به بخش ویرایش حقوق ها دسترسی های لازم را داشته باشد. بدین نحو که هر دو مدیر و حسابدار به بخش ویرایش حقوق ماهانه ی هر کارمند دسترسی دارند ، حال در آخر هر ماه مدیر میتواند پرداخت را هم انجام دهد. این کار در بخش خرید اقلام موردنیاز آشپزخانه توسط پیک هم به ما کمک میکند چراکه باعث میشود حسابدار با تایید کردن رسید اقلام مورد نیاز توسط پیک ، مبلغی که پیک بابت تهیه ی اقلام پرداخت کرده است را به حقوق ماهانه ی او اضافه کند.**
  + **در بخش بعدی در مورد غذا ها صحبت کردیم. گفتیم ممکن است در رستوران یک غذا به دلایلی از جمله در دسترس نبودن و یا اتمام مواد اولیه ی مورد نیاز ، قابل پخت نباشد. فلذا تصمیم بر آن شد که برای این مهم ، یک State برای هر غذا در منو در نظر بگیریم. اگر غذا در دسترس باشد ، که مشکلی نیست. اما اگر از دسترس خارج شود ، آشپز میتواند با ورود به بخش "ویرایش منوی شعبه" ، آن غذا را در حالت "اتمام موجودی" قرار دهد. با اینکار به پیچیدگی ها در پیاده سازی اضافه نمیشود و تصمیم درستی است که باید آن را نگاه داریم.**
  + **در قسمت نیازمندی ها اول به این صورت بود که کاربر با انجام خرید و پرداخت هزینه ی مورد نظر ، به صفحه ی سفیدی هدایت میشد و در آن قسمت lock میشد تا غذا به دست مشتری برسد. اما به پیشنهاد دوستان ، این حالت تغییر کرد. دلیل این تغییر آن بود که مشتری شاید بخواهد بار دیگر نیز سفارش بدهد و مجدد از اپلیکیشن استفاده کند. پس نباید مشتری در حالت lock قرار گیرد اما پیک باید در حالت lock قرار گیرد چراکه مادامی که یک کار را تا انتها انجام نداده است حق انجام کار دیگری نباید داشته باشد. بطور مثال مادامی که سفارش A را به دست مشتری نداده است نباید بتواند قبول کند که فلان سفارش و یا فلان اقلام مورد نیاز را تهیه میکند.**
  + **در قسمت اقلام مورد نیاز آشپزخانه ، توضیحات کم بود و باعث سردرگمی می شد بهمین واسطه با ارایه ی توضیحات بیشتر و بهتر و با همفکری اعضای تیم ، یک Data Flow برای آن مشخص کردیم که مورد تایید همه قرار گرفت. Data Flow در این بخش اینطور است که ابتدا آشپز اعلام میکند که فلان ماده ی اولیه در انبار نیست. بعد این اعلام نیازمندی توسط آشپز را حسابدار چک میکند که آیا این نیازمندی صحت دارد یا خیر. همچنین حسابدار مقدار هزینه ی تهیه ی این اقلام را وارد میکند. سپس این اقلام تایید شده ، به پشته ی اقلام مورد نیاز قابل مشاهده برای پیک میرود. در صورتی که پیک مسئولیت آن را به عهده بگیرد نام پیک به آن نیازمندی ها اضافه میشود به این معنی که این شخص مسئول تهیه ی آن ها خواهد بود. نهایتا حسابدار میتواند تایید کند که آیا پیک آن را تهیه کرده است یا خیر. گفتیم ممکن است اینکار و تعریف این صف و پشته باعث شود که وابستگی کلاس ها به همدیگر بسیار زیاد شود چراکه این نیازمندی ها توسط آشپز اعلام شده ، توسط حسابدار هزینه ها وارد شده ، توسط پیک نام وی بر آن ها جهت تهیه ثبت شده و نهایتا توسط حسابدار تایید میشود. حال برای رفع این وابستگی ها در این سه کلاس ، تصمیم بر آن شد که کلاس Integrate تعریف شود که رابطی بین این سه کلاس باشد و از وابستگی های آن ها بکاهد.**
  + **مورد بعدی که دوستان پیشنهاد کردند آپشن "لغو سفارش" برای مشتری است که موجب پیچیدگی و افزایش کار در پروژه میشود همچنین که اولویت بالایی برای پروژه ندارد. با در نظر گرفتن اینکه این خروجی ، بعنوان نسخه ی اولیه ی اپلیکیشن است ، در مجموع بر آن شدیم که این گزینه را به پروژه اضافه نکنیم.**
  + **در قسمت Context Diagram کمبودی وجود داشت و آن هم این بود که درگاه شاپرک را به عنوان سیستم خارجی در نظر نگرفته بودیم و درون خود نرم افزار نهفته میدیدیم. ولی با بحث و بررسی هایی که انجام شد ، بر آن شدیم که درگاه پرداخت شاپرک را به عنوان یک سیستم مجزا در نظر بگیریم که به برای سیستم ما "خدمات پرداخت" را فراهم میکند.**

# **آنالیز نیازمندی ها جنبه ساختاری**

## رسم CRC Cards

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class:Account   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the Account details(e.g phoneNumber,Accountname and password) |  | |  | DeliveryDriver | |  | Cook | |  | RestaurantAdmin | | Uses the functions in SystemInterfaceForAccount to Sign Up,Login and Change Password | SystemInterfaceForAccount | |  | Accountant | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class: EmployeeData   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the employee details(e.g  Accountname,wage,BankAccount and the  restaurants the employee is working  for) | RestaurantInterfaceForAdminAndAccountant | |  | DeliveryDriver | |  | Cook | | Allow Restaurant to edit food details | Restaurant | |  | Accountant | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class:Food   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the food details(e.g name,price  image and status,which shows if it is  available for ordering or not and Price status which shows if it’s price is approved) | RestaurantInterfaceForCookAndDeliveryDriver | | Allow Restaurant to edit food details | Restaurant | |  | RestaurantInterfaceForAdminAndAccountant | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class: Ingredients   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the Ingredient details(such as  name,price image and status,which shows if it is approved or not and validationStatus that shows if the purchase is validated)which could be used by the cook  (except the validationStatus which is set by Accountant) | Cook | | Allow Restaurant to edit ingredient details | Restaurant | |  | RestaurantInterfaceForAdminAndAccountant | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class: Accountant   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the restaurant which accountant  works for | Restaurant | | hold the employee data for the  accountant | EmployeeData | | Uses the functions in CookAndRestaurantAdminInterfaceForAccountant to have report on cash flow and request  Menu change | CookAndRestaurantAdminInterfaceForAccountant | | Uses the functions in  RestaurantInterfaceForAdminAndAccountant  to get cash flow ,approve ingredients and validate ingredient purchases. | RestaurantInterfaceForAdminAndAccountant | |  | Account | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class: Cook   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the restaurant which cook  works for | Restaurant | | hold the employee data for the  cook | EmployeeData | | Implements the functions in CookAndRestaurantAdminInterfaceForAccountant | CookAndRestaurantAdminInterfaceForAccountant | | Uses the functions in RestaurantInterfaceForCookAndDeliveryDriver to edit the ingridients,edit the menu and get the orders of the restaurant | RestaurantInterfaceForCookAndDeliveryDriver | |  | Account | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class: RestaurantAdmin   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the restaurants which the admin  works for | Restaurant | | Implements the functions in CookAndRestaurantAdminInterfaceForAccountant | CookAndRestaurantAdminInterfaceForAccountant | | Uses the functions in RestaurantInterfaceForAdminAndAccountant  To hire/fire employees, edit and pay their wages and get the cash flow of the restaurant | RestaurantInterfaceForAdminAndAccountant | | hold the cash flow reports from the accountant |  | |  | Account | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class:Customer   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the customers details(e.g name,address and their orders) | Order | | Uses the functions in SystemInterfaceForCustomer To take out ,  See the restaurants and leave a review | SystemInterfaceForCustomer | |  | Account | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class:order   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | hold the order details(e.g order items,the customer and status of it,i.e if it’s done or not) | Customer | | Allow Restaurant to edit order details | Restaurant | |  |  | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Interface: RestaurantInterfaceForAdminAndAccountant   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | Contain the methods which to be implemented in  Restaurant and which are used by RestaurantAdmin  And Accountant | Restaurant | |  | Accountant | |  | Restaurant Admin | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Interface: RestaurantInterfaceForCookAndDeliveryDriver   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | Contain the methods which to be implemented in  Restaurant and which are used by the Cooks  And DeliveryDrivers | Restaurant | |  | Cook | |  | DeliveryDriver | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Interface: CookAndRestaurantAdminInterfaceForAccountant   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | Contain the methods which to be implemented in  SystemDataBase and which are used by Accounts | SystemDataBase | |  | Account | |  |  | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Interface: SystemInterfaceForCustomer   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | Contain the methods which to be implemented in  SystemDataBase and which are used by Customers | SystemDataBase | |  | Customer | |  |  | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Interface: RestaurantInterfaceForAdmin   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | Contain the methods which to be implemented in  SystemDataBase and which are used by RestaurantAdmin. | SystemDataBase | |  | RestaurantAdmin | |  |  | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class:SystemDataBase   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | Contain the Restaurants registered in  The system | Restaurant | | Contain the AccountData of Accounts(i.e. password and  Accountname). |  | | Implement the functions in RestaurantInterfaceForAdmin which are used  By RestaurantAdmin. | RestaurantInterfaceForAdmin | | Implement the functions in  SystemInterfaceForCustomer  Which area used by Customer . | SystemInterfaceForCustomer | | Implement the functions in SystemInterfaceForAccount  Which are used by Account. | SystemInterfaceForAccount | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Interface: SystemInterfaceForAccount   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | Contain the methods which to be implemented in  SystemDataBase Which are used by Account | SystemDataBase | |  | Account | |  |  | |  |  | |  |  | |  |  | |  |  | |

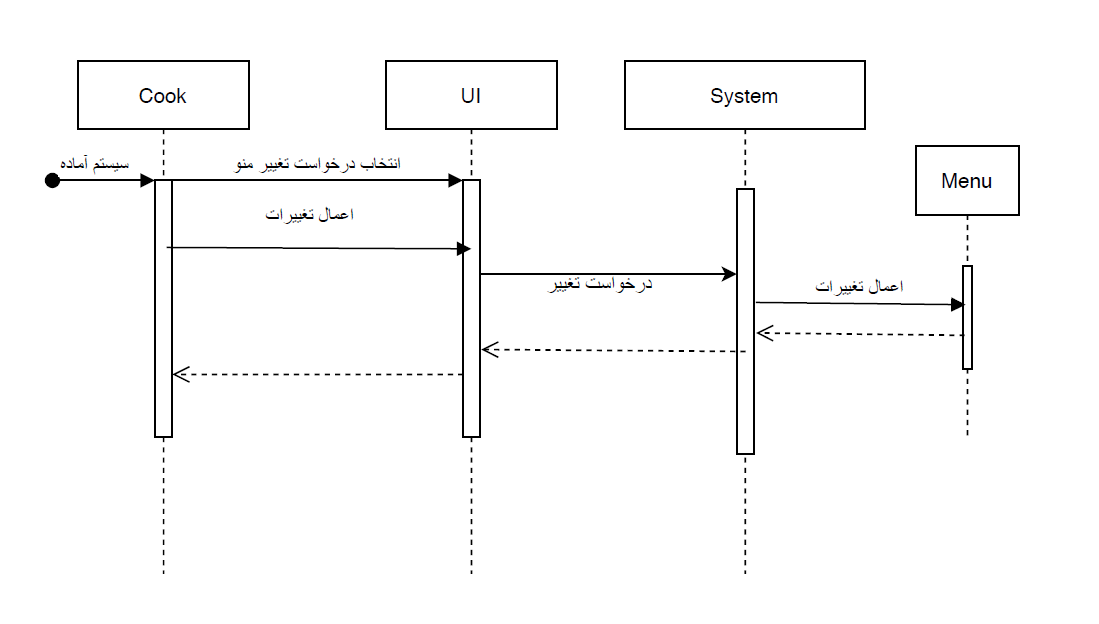
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class: Restaurant   |  |  | | --- | --- | | Responsibilities: | Collaborators: | | Contain the cooks of the Restaurant | Cook | | Contain the DeliveryDrivers of the Restaurant | DeliveryDriver | | Contain the RestaurantAdmin of the Restaurant | RestaurantAdmin | | Contain the Accountant of the Restaurant | Accountant | | Contain the data of employees | EmployeeData | | Contain the Ingredients of the Restaurants | Ingredient | | Contain the orders of the Restaurant | Order | | Contain the details of the Restaurant(i.e. reviews,cash flow,name and address) |  | | Contain the food of the Restaurant(menu) | Food | | Get Order From SystemDataBase | SystemDataBase | | Implement the functions in RestaurantInterfaceForCookAndDeliveryDriver  Which are used by Cook and DeliveryDriver | RestaurantInterfaceForCookAndDeliveryDriver | | Give Menu to SystemDataBase | SystemDataBase | | Implement the functions in  RestaurantInterfaceForAdminAndAccountant  Which are used by RestaurantAdmin and Accountant | RestaurantInterfaceForAdminAndAccountant | |

# **Class diagram**

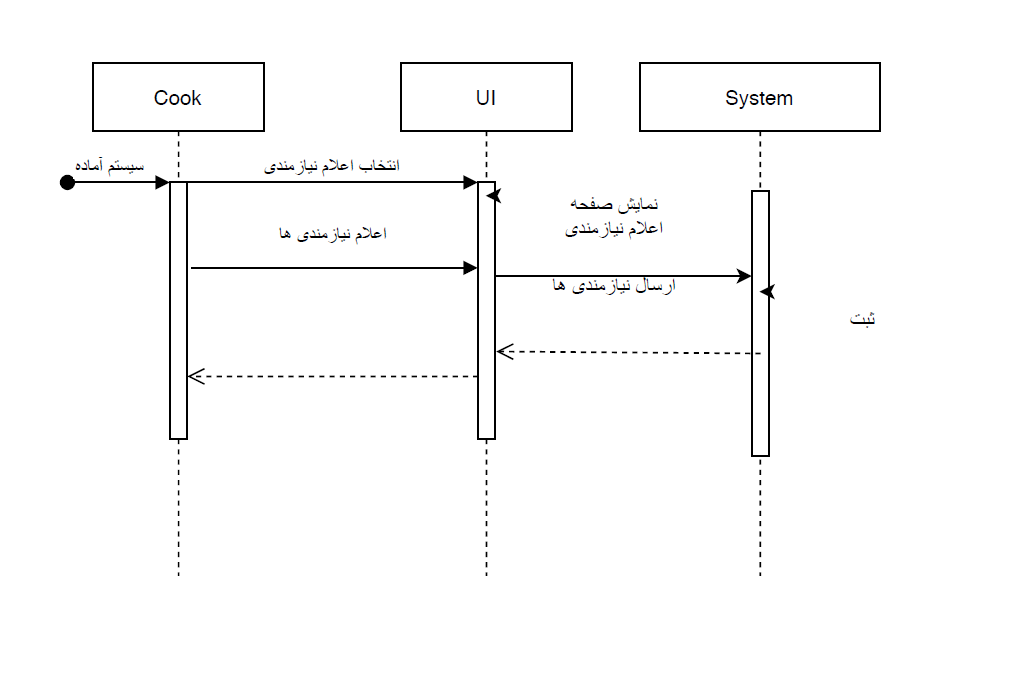
تصویر

# **Sequence diagram**

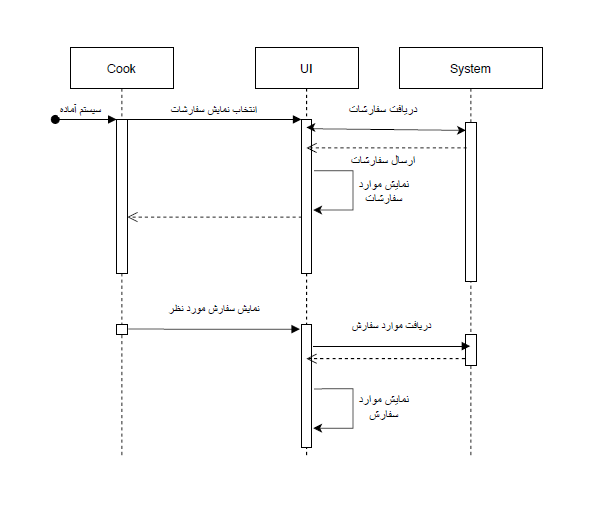
نمودار شماره 1



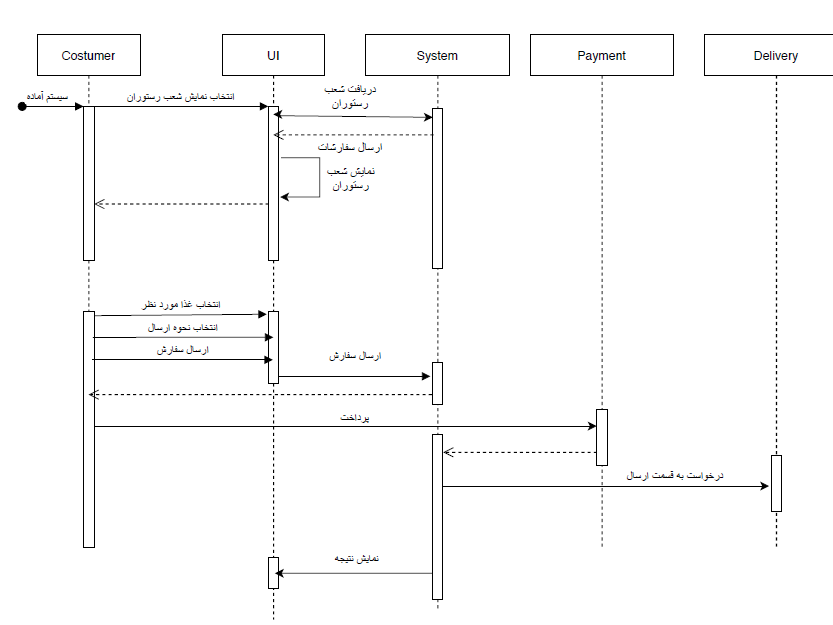
نمودار شماره 2



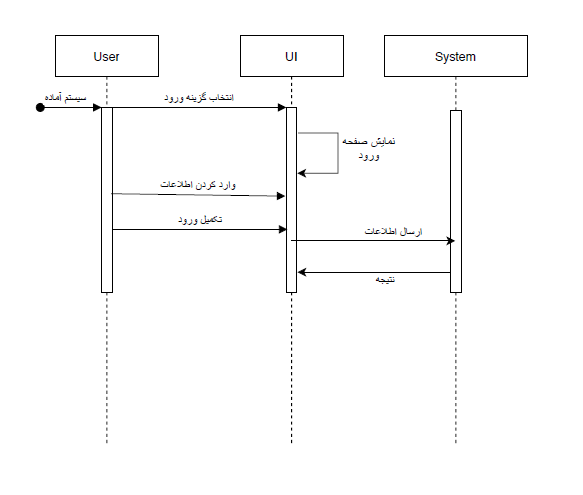
نمودار شماره 3



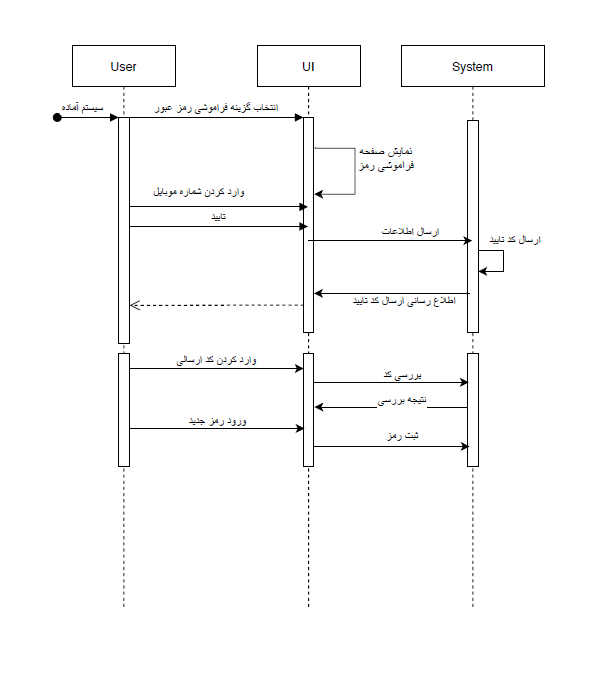
نمودار شماره 4



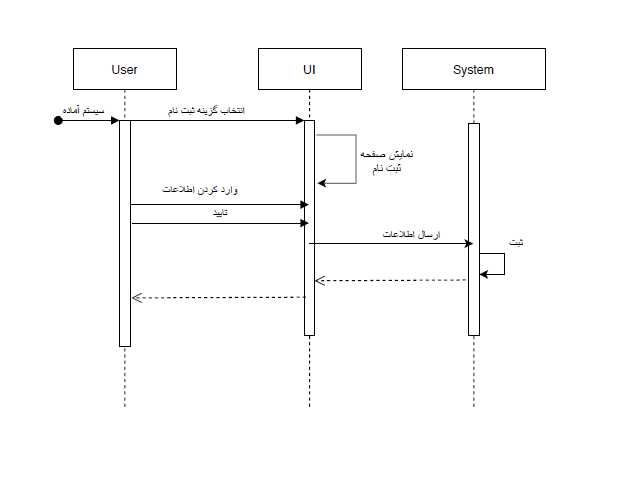
نمودار شماره 5



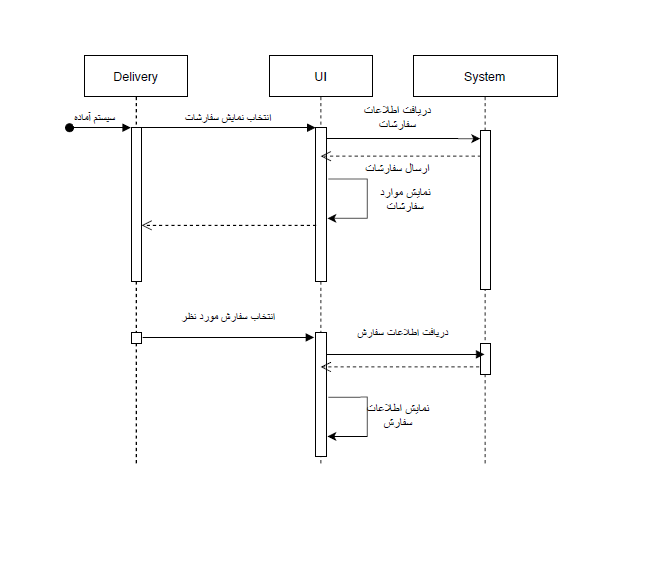
نمودار شماره 6



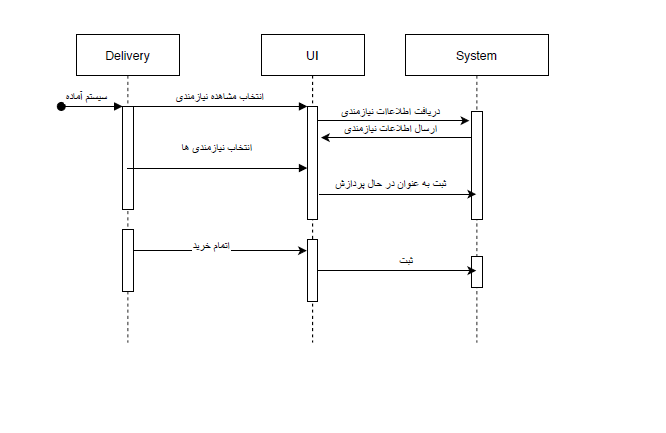
نمودار شماره 7



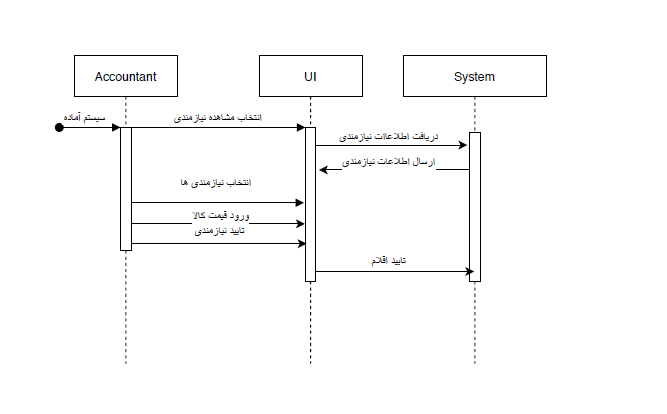
نمودار شماره 8



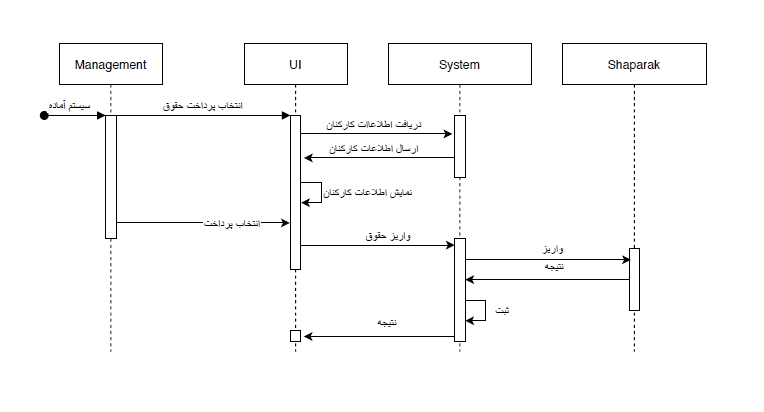
نمودار شماره 9



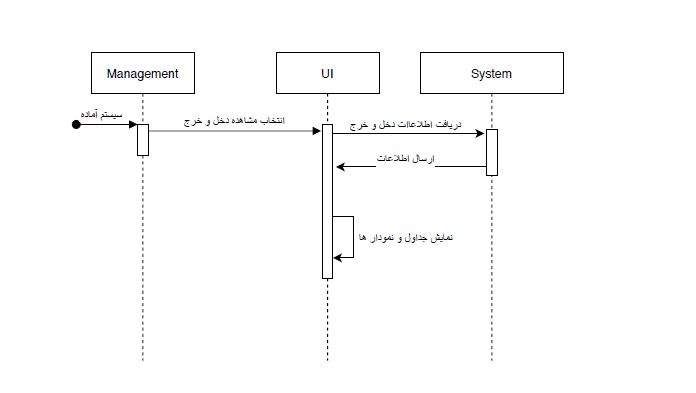
نمودار شماره 10



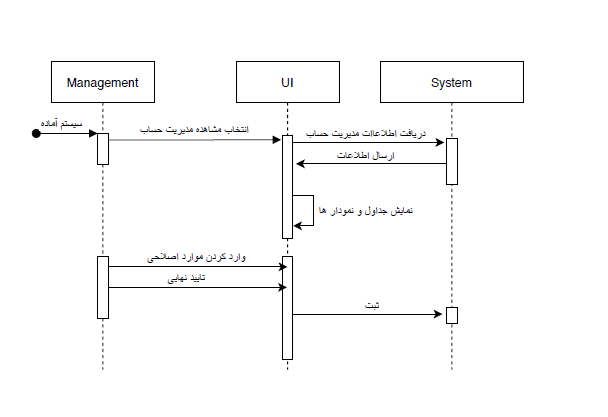
نمودار شماره 11



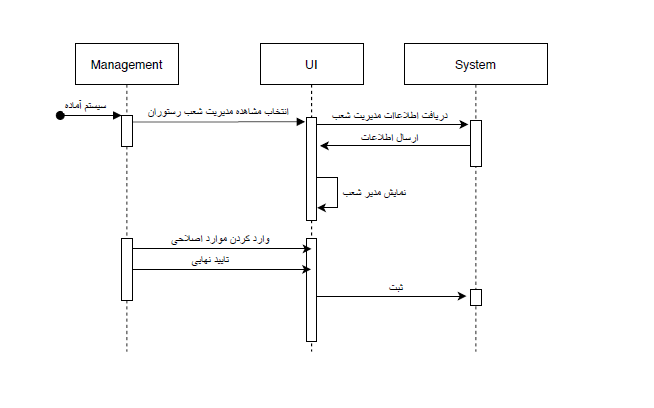
نمودار شماره 12



نمودار شماره 13



نمودار شماره 14



# **Technical Review**

* شماره : 2
* مسیول جلسه :
* تاریخ جلسه : 30/04/1399
* نظرات و جمع بندی ها :

**در این جلسه نیز مانند جلسه ی گذشته به موارد انجام شده و بررسی آن ها پرداختیم و برای تغییرات صحبت هایی کردیم که در ادامه بطور مفصل به آن خواهیم پرداخت.**

* + **در قسمت اول به یک Actor اضافه به نام Supplier رسیدیم. بدین شرح که قبل تر گفته بودیم در قسمت نیازمندی ها ، حسابدار و آشپز برای چک کردن و تعامل در قبال اقلام مورد نیاز آشپزخانه باید انبار را چک کنند که ببینند آیا موارد مورد نیاز در انبار موجود شده ، هست و یا بوده است. پس یک Actor نیاز داریم که مشخص کند که آیا اقلام در انبار موجود است یا خیر. و یا میتوان خود انبار را یک Actor در نظر گرفت که دیگر actor ها برای انجام برخی فعالیت ها به آن رجوع میکنند و وابسته هستند. پس این Actor اضافه شد.**
  + **در قسمت بعدی ، به Food پرداختیم و اینکه شایسته تر است که حسابدار قیمت غذا در منو را مشخص کند تا اینکه آشپز. چراکه حسابدار در جریان دخل و خرج های شرکت هست و همچنین میداند که سود شرکت در چیست. فلذا برای پررنگ کردن این نقش و دادن اختیارات در حیطه ی کاری به وی ، تصمیم بر آن شد که آشپز فقط بتواند منو را اصلاح کند و یا به آن موردی را اضافه کند. ولی نوشتن قیمت غذا ها و ویرایش آن ها بر عهده ی حسابدار خواهد بود. فلذا موارد اضافه شده توسط آشپز دارای قیمت نخواهند بود و حسابدار میتواند با گزینه ی "اصلاح قیمت منو" آن ها را تغییر دهد. برای این کار پیچیدگی زیادی به پروژه اضافه نمیشود و کار معقولی است که میصرفد آن را به پروژه افزود.**
  + **در قسمت بعدی ، یکی از دوستان پیشنهادی داد بابت آنکه پروژه را گسترده تر کنیم. یعنی این را در نظر بگیریم که میتوان از این پروژه استفاده های بهتر و جامع تری کرد. برای نمونه ماشین های Drive Through را مثال زدند که در کشور های خارجه خصوصا غرب از آن استفاده ی زیادی میکردند. به این شرح که این پروژه را بر روی ماشین های Drive Through میتوان پیاده کرد با این فرض که هر شخص با توجه به موفعیتی که دارد ، نزدیک ترین دستگاه را انتخاب کند و سفارش را تکمیل کند. حال برای انجام تحویل کامل سفارش میتوان به دستگاه مراجعه کرد و با ارایه ی کد ارسال شده غذا را از دستگاه تحویل گرفت.**

# **طراحی معماری**

## **انتخاب و استدلال استایل های کاندید برای معماری سیستم**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Layered | Call & return | Data Flow | Data Center |
| Purpose | Incremental delivery | Divide & conquer | Data modifiability | Integrality of data |
| میزان استقلال اجزا | متوسط | پایین | پایین | بسیار بالا |
| امکان کار همزمان | بالا | پایین | پایین | بسیار بالا |
| چالش اصلی | لایه بندی مستقل | شکستن سیستم | تاخیر و بازدهی سریع | تست و تغییر در اطلاعات اصلی |
| میزان Reusability | بالا | پایین | بالا | بسیار بالا |
| پل ارتباط اجزا | لایه ی میانی آن دو | Function returns | I/O | Data Center |

**معیار ها:**

1.**مدیریت داده ها و سرور اصلی**: در این پروژه مدیریت داده ها نقش مهمی دارند و برای مدیریت آن ها به یک سرور اصلی نیازمندیم. یعنی مهم ترین قسمت پروژه به این صورت است که بتوانیم در کنار اینکه بخش امنیت سرور را فراهم آوریم ، هر شخص یا actor بر روی داده های پردازش شده کار خود را انجام دهد. سه معماری که محوریت داده ها و سرور را در نظر دارند ، عبارتند از Database center و data flow و Layered.

2.**استقلال عمل**: همانطور که قبلتر در قسمت نیازمندی ها نیز بحث شد ، تعداد actor ها در این پروژه زیاد هستند و هر کدام هم نقش خاص خودشان را برعهده دارند که لزوما نیازی نیست از کار یکدیگر باخبر شوند. یعنی همین که هر شخص داده ها را بخواند ، تغییرات را روی آن اعمال کند و باز پس دهد ، کافیست. این استقلال در عملکرد بیشتر در دو معماری Layered و Data center دیده میشود.

3.**سادگی معماری**: یکی ازمعیار های بسیار مهم برای تمام پروژه ها ، پرهیز از افزودن پیچیدگی های غیرضروری به پروژه میباشد. در میان این چند نوع معماری ، معماری Data center ، Layered ، OOP از سادگی بیشتری برخوردارند و اعضای تیم بیشتر با این استایل ها آشنایی دارند. پس انجام پروژه در این سه سبک به صرفه تر است.

4.**اخلال در عملکرد**: از آنجا که پروژه ی ما یک پروژه ی اپلیکیشنی است که با اینترنت کار میکند ، امکان اخلال در اینترنت و بروز خطا برای اجرای ادامه ی عملکرد وجود دارد فلذا تا حد ممکن اطلاعات در هر قسمت باید در دیتابیس مجدد آپدیت گردد تا با بازگشت به اپلیکیشن ، کار از همانجا ادامه یابد. برای این امر Database Center بسیار مقرون به صرفه تر از دیگران است.

5**.مدیریت ریسک پیاده سازی**: چون پروژه ، پروژه ی بسیار بزرگی است ، باید کامپوننت ها و اجزای سیستم طوری قرار گیرند که در صورت درخواست برای ایجاد یک Feature جدید ، بتوان به راحتی آن را پیاده سازی کرد. از آنجا که در Database center این استقلال بسیار زیاد است ، میتوان صرفا در یک جزء آن تغییر را اعمال کرد و ادامه داد.

6**.ماهیت و بستر**: پروژه طوری است که میتوان آن را به بستر وبسایت انتقال داد. در اپلیکیشن های تحت وب ، بیشتر از معماری Layered استفاده میشود و باید در نظر داشت که پس میتواند Best Practice های بهتری در آن یافت که به پروژه ی ما بسیار نزدیک است.

**بررسی هر معماری:**

1.**Call & return**: پروژه اینطور نیست که بتوان آن را به قسمت های مختلف شکست. یعنی پروژه بیشتر از اجزای مستقل از هم تعریف میشود تا اینکه یکپارچه باشد و تکه تکه باشد. همچنین که در معماری Call & return که بر پایه ی Divide & conquer است اعمال تغییرات و مدیریت ریسک در آن کمتر است و از آنجا که پروژه دارای Feature های زیادی است و امکان تغییر و جا افتادن چند feature وجود دارد بهتر است که از این معماری پرهیز شود.

2.**Layered**: این معماری در بخش وبسایت و یا اپلیکیشن موبایل میتواند خیلی بدرد بخور باشد. چراکه لایه های مختلف یعنی Backend ، Frontend را از یکدیگر جدا میسازد و به راحتی میتوان سیستم را به سه لایه ی server ، Data interfaces و actors تقسیم بندی نمود و روی هر کدام بصورت مستقل وقت وانرژی گذاشت. همچنین که میدانیم امنیت برای هر لایه متفاوت است چراکه دست یابی به اطلاعات سرور و تغییر در آن ها باید بسیار سخت تر از دستیابی به اکانت یک شخص باشد چراکه بدلیل وجود تراکنش ها و مبادله ی هزینه ها و پول ، امنیت در این پروژه بسیار مهم است. پس این معماری میتواند معماری مناسبی باشد.

3.**Data Flow**: این معماری بدلیل اینکه جریان داده ها را با توجه به filter ها پیگیری میکند از این نظر میتواند به پروژه کمک شایانی کند اما بدلیل وجود مشکلاتی مثل امکان افزودن Feature های جدید و یا actor های جدید این کار برای ما با این معماری ممکن است دشوار باشد. همچنین که در پروژه ، با هر پردازش لزوما باید به سرور ارتباط گیری شود ؛ یعنی در انتهای هر فیلتر باید اطلاعات به سرور برود و سیو شود. پس منطقی تر است که از این معماری بدلیل دوباره کاری استفاده نکنیم.

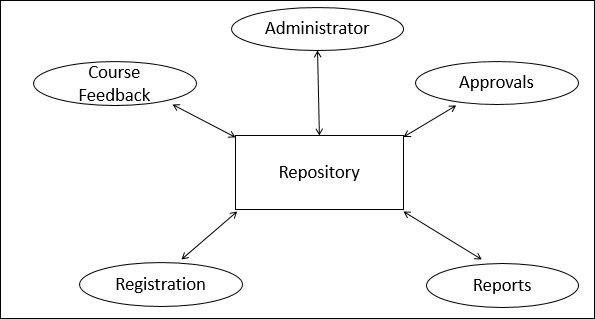
4.**Database Center**: اطلاعات پردازش شده در پروژه و سیستم ما ، تقریبا همه ی Actor ها را درگیر میکند. یعنی در قسمت سفارش مشتری ، نه تنها مشتری درگیر پردازش این اطلاعات است بلکه پس از آن به ترتیب آشپز و حسابدار هم بصورت مستقیم آن اطلاعات را پردازش میکنند و همچنین که حسابدار و مدیر هم بصورت غیر مستقیم این اطلاعات پردازش شده را مشاهده میکنند و باید به آن ها اجازه ی دسترسی داده َشود. پس این معماری میتواند بسیار برای پروژه مفید باشد.

**جمع بندی و نتیجه**: میتوانیم برای انجام پروژه از ترکیب دو معماری Layered و Database center استفاده کنیم. بدین شرح که سه لایه ی اصلی داریم (UI ، mediate و Server). لایه ی UI که لایه ی تعاملی ما با کاربران است. در لایه ی mediate تغییرات بر روی داده ها رخ میدهد و دیتاها به server میرود. حال در لایه ی mediate که شامل کلاس های Restaurant و ... میشود ، از آنجا که Restaurant تقریبا از همه ی اطلاعات پردازش شده ، آگاه هست ؛ میتواند بصورت center باشد و بعنوان database در لایه ی دوم به آن نگاه کرد.

## **انتخاب و استدلال الگوهای کاندید برای معماری سیستم**

**Repository Pattern**

این الگو به استایل Data-centered مربوط میباشد . در داخل این الگو ، یک Repository در وسط داریم که passive میباشد و یعنی خودش به تنهایی کاری نمیکند و client هایی که به آن وصل هستند به صورت active درخواست میدهند. در پروژه ما Restaurant ، به صورت Passive قرار دارد و بقیه اجزای سیستم با آن در ارتباط هستند .



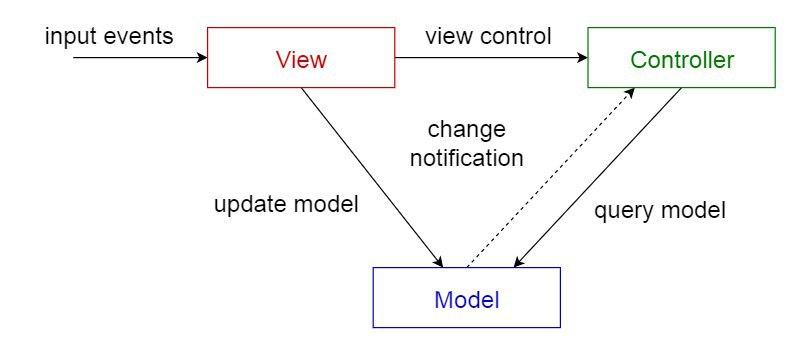
مزایا :

* + ارایه یک معماری انعطاف پذیر
  + جدا شدن منطق داده ، سرویس و بیزینس از هم ( مستقل بودنشان )
  + کاهش Redunadancy
  + خطای کمتری را تجربه میکنیم

**MVC Pattern**

این الگو از 3 بخش تشکیل شده است :

* Model : بخش منطق برنامه که با دیتا ها سروکار دارد.
* View: بخش UI و ارتباط با کاربر
* Controller : اینترفیس بین 2 بخش بالا



مزایا :

* قابل نگهداری (maintainable)
* قابل تست کردن
* آسانی در نسبت دادن نقش های جداگانه به لایه ها ( استقلال بالا)
* آسانی در اصلاح لایه ها

مناسب برای :

* برنامه های جدیدی که نیاز دارند تا سریع ساخته شوند (Built Quickly Apps)
* برنامه های سازمانی
* تیم های نوپا که هنوز با معماری های دیگر که عمدتا پیچیده تر هستند آشنا نمیباشند
* برنامه هایی که به استاندارد های دقیق نیازمند میباشند

در پروژه ما :

**View :**

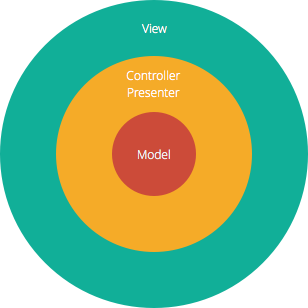
بخش UI برنامه است که به طور مستقیم با کاربر در تعامل است یعنی وقتی کاربر وارد صفحه sign up میشود، با چه صفحه ای روبرو خواهد بود و چه گزینه ها و دکمه ها و تکست باکس هایی را میبیند که برای انجام ثبت نام باید پر کند و تیک بزند و ... .

**Model :**

بخش منطق برنامه است که نشان میدهد مثلا وقتی کاربر وارد صفحه ثبت نام میشود با انتخاب دکمه ها و پر کردن تکست باکس ها و .... چه کارهایی در Back-end صورت میگیرد .

**Controller :**

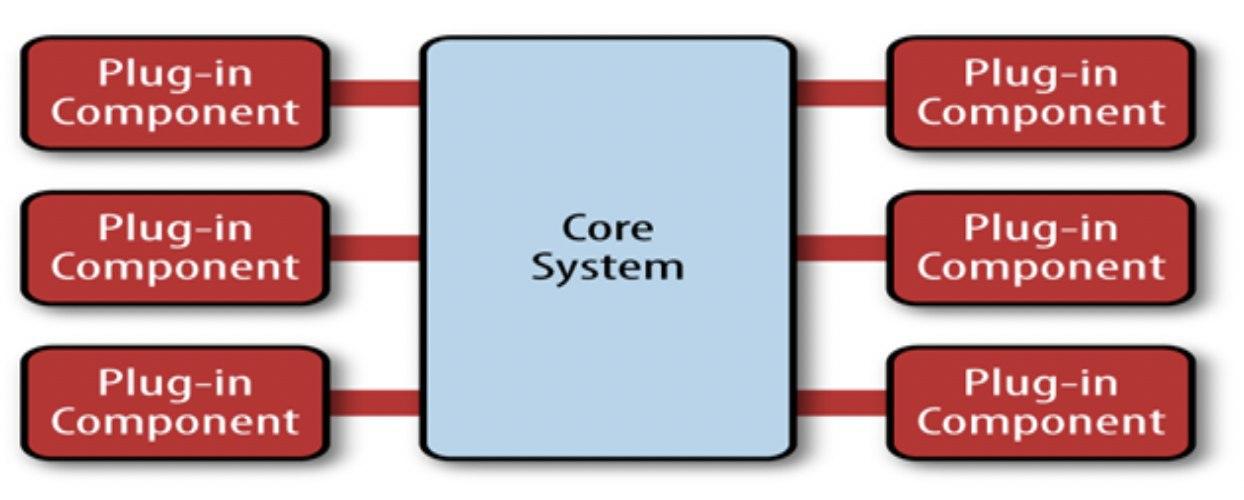
رابطه بین UI و منطق برنامه



**Microkernel Pattern**

این الگو برای سیستم های منعطف با تغییرات نیازمندی ها خوب است .در اینجا یک core system داریم که حداقل قابلیت ها را درونش پیاده سازی کرده ایم تا حدی که اپلیکیشن بالا بیاید (Main Program)

در اینجا از plugin-component استفاده میکنیم . یعنی در سایر الگو های معماری اگر بیاییم وبخشی را تغییر دهیم ، کل سیستم تحت تاثیر قرار میگیرد ولی اینجا قابلیت های اضافی برنامه را به صورت آماده از ببیرون می آوریم استفاده میکنیم . و این component ها به طور کاملا مستقل از هم طراحی میشوند .



مزایا :

* واکنش نشان دادن به تغییرات در ماژولهای plug-in در عین حداقل کردن تغییرات core-system
* Deploy کردن راحت تر به خاطر وجود plug-in ها
* تست کردن جداگانه هر ماژول
* امکان شخصی سازی در انتخاب plug-in ها وجود دارد - انتخای RAD

مناسب برای :

* برنامه هایی که طیف وسیعی از مردم با آن در ارتباط هستند
* برنامه هایی که به طور مشخص به 2 بخش روال معمول(هسته اصلی) و روال سطح بالا و دینامیک(متغیر) تقسیم بندی شده اند .

در پروژه ما :

**Core System :**

ارتباط بین Customer و Restaurant و انجام فرآیند خرید

**Plug-in components :**

مشاهده لیست شعبه ها و انتخاب از بین انها

مشاهده منوی شعبه

اعلام نظر راجع به غذا

انجام فرآیند پرداخت سفارش

و ....

نتیجه گیری :

طبق بررسی هایی که در بالا شکل گرفت با نگاه به مزایا و ویژگی های الگو های مختلف ، میتوانیم از ترکیب الگو های MVC و Microkernel استفاده کنیم.

# **ارائه حداقل دو طرح برای معماری سیستم در قالب**

* ***طرح 1***

در این طرح از استایل Layerd برای معماری استفاده شده است.

**Package diagram**

هدف از تعریف Package diagram ، نشان دادن نحوه شکسته شدن یک سیستم بزرگ به تعدادی زیر سیستم و نمایش چگونگی ارتباط بین این اجزا است . پس هنگامی که میخواهیم package diagram رسم کنیم ، سیستم را به چند زیرسیستم تقسیم میکنیم و وابستگی بین آنها را هم نشان میدهیم .

توجه : میتوانیم package های تودر تو هم داشته باشیم .

توجه : رابطه بین Package ها چند نوع هست :

* + **import** : مثلا package A می آید و از functionality ها و قابلیت های package B استفاده میکند .
  + **access** : مثلا package A می آید و برای اجرا شدنش به package B نیاز دارد .
  + **merge** : همان معنی extend را میدهد
  + **use** : مثلا A از تمام operation ها B استفاده میکند و به تمام قابلیت های آن نیاز دارند .

نحوه رسم package diagram :

* + شناسایی package ها
  + شناسایی وابستگی های بین package ها

سیستم اصلی :

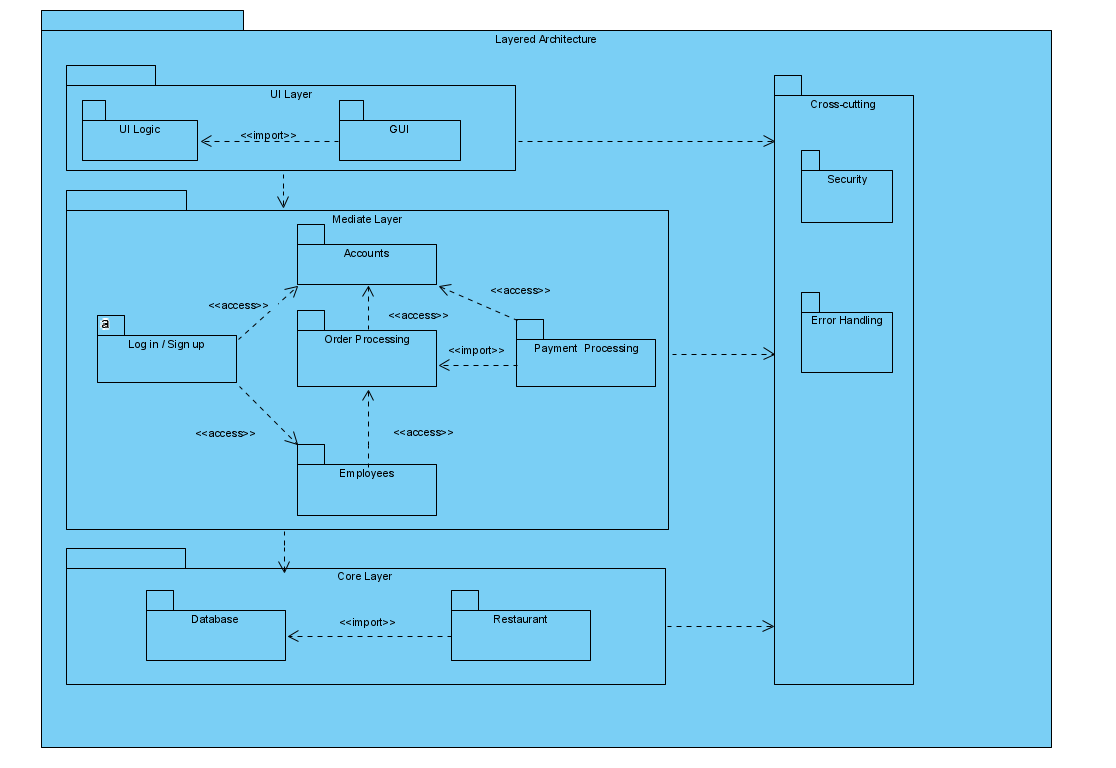
Restaurant

زیر سیستم ها :

UI , Log in – sign up , Order processing , Payment processing , Error handling for all other packages ,Security(authentication , … ) , Database

همانطور که صحبت کردیم در طرح 1 از استایل Layered استفاده میکنیم :

Package Diagram :



توجه :

UI Layer : کاربر (مشتری و غیر مشتری) به طور مستقیم با این لایه در ارتباط هست وبقیه لایه ها از نظر کاربر پنهان هستند .

Mediate Layer(Logic) : در این لایه component هایی وجود دارد که کاربر از طریق UI با اینها در ارتباط است . مثلا : پرداخت

Core Layer(Server) : لایه اصلی نرم افزار که در آن ابزار های اصلی قرار دارند .مثلا : دیتابیس

**deployment diagram**

هدف از رسم این دیاگرام این است که نشان دهیم که node(گره) ها چه طور با هم دیگر در ارتباط هستند . گره ها شی فیزیکی هستند که حافظه دارند و همچنینتوانایی پردازش هم دارند . مثلا : Server ها و Firewall ها و ....

در واقع این دیاگرام نحوه معماری اپلیکیشن را از بعد فیزیکی نشان میدهد .

ارتباط بین گره ها :

* **dependency** :

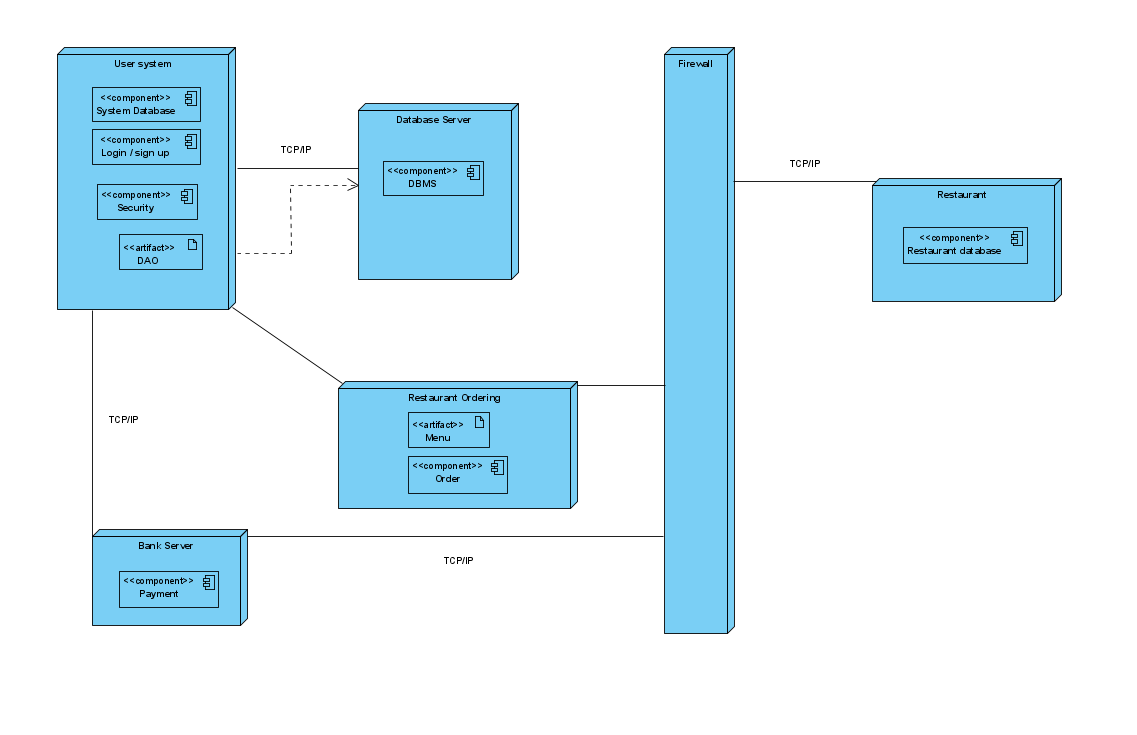
این ارتباط نشان میدهد که Source به Target وابسته هست یا نه . در واقع اگر تغییری در Target صورت گیرد ، Source هم تغییر میکند .

* **connection** :

این ارتباط مثل association است که در class diagram داشتیم . و 2 نوع دارد :

* + آسنکرون : وقتی داده ای را میفرستیم ممکن است که برنگردد و باید هندل شود.
  + سنکرون : در این حالت درگیر وصل شدن نیستیم و میسپاریم به لایه شبکه مثل TCP/IP
* **Artifact** :

هر المان فیزیکی که نتیجه یک develop هست میباشد که میتواند موارد مقابل باشد : 1. library 2. فایل exe 3. دیتابیس 4. فایل .config و .....

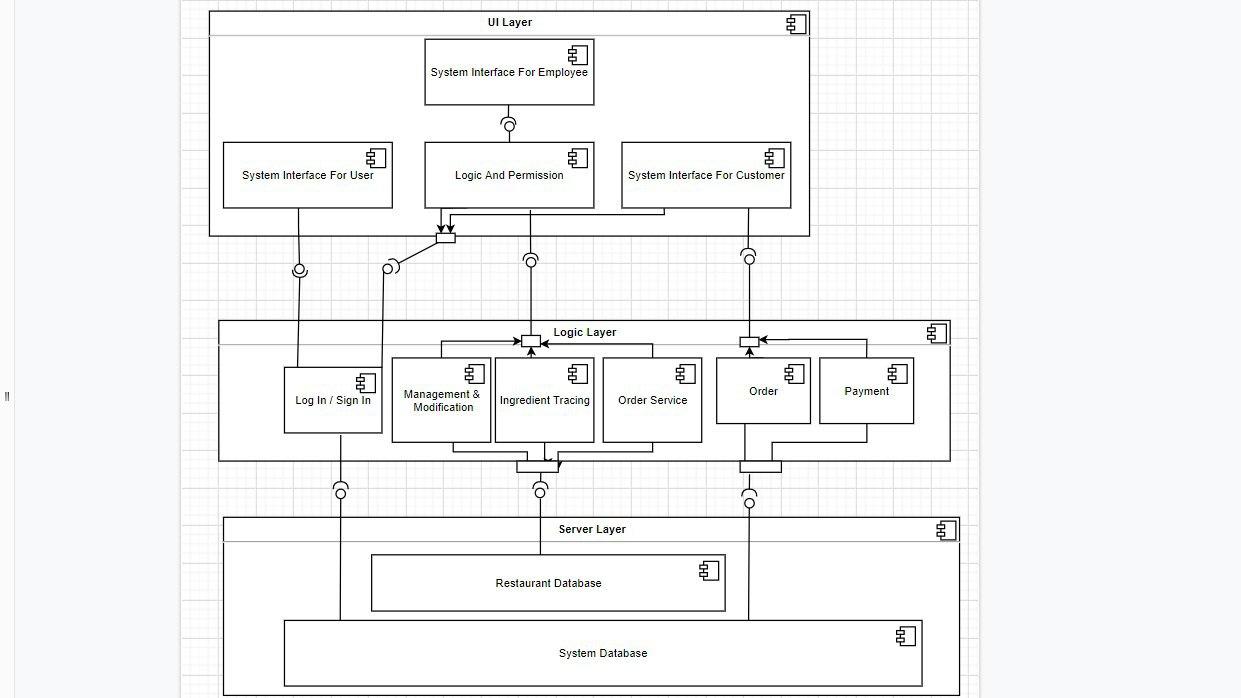


توجه : منظور از DBMS(Database Management System) ، system database در پروژه هست .

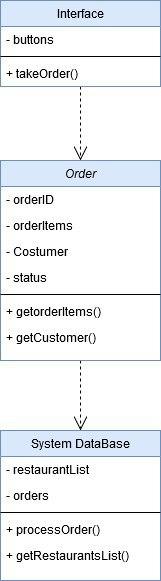
توجه : DAO به معنای Data Access Object میباشد و یک اینترفیس در اختیار کاربر قرار میدهد تا بتواند با دیتابیس کار کند .

توجه : علت استفاده از Security و Firewall ، به دلیل امنیت هنگام احزار هویت کابر در هنگام ورود به اپلیکیشن است . ( و در کل امنیت اطلاعات کاربر)

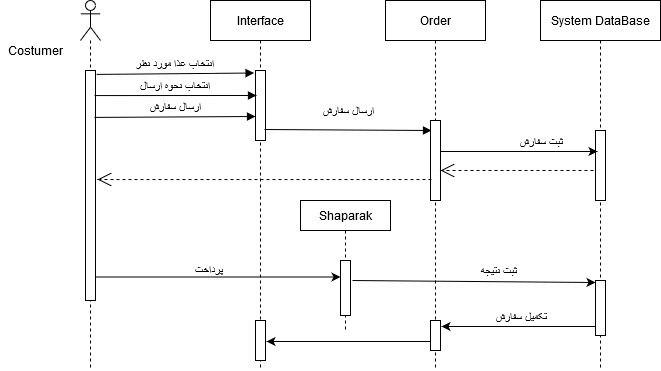
**component diagram**



**class diagram**



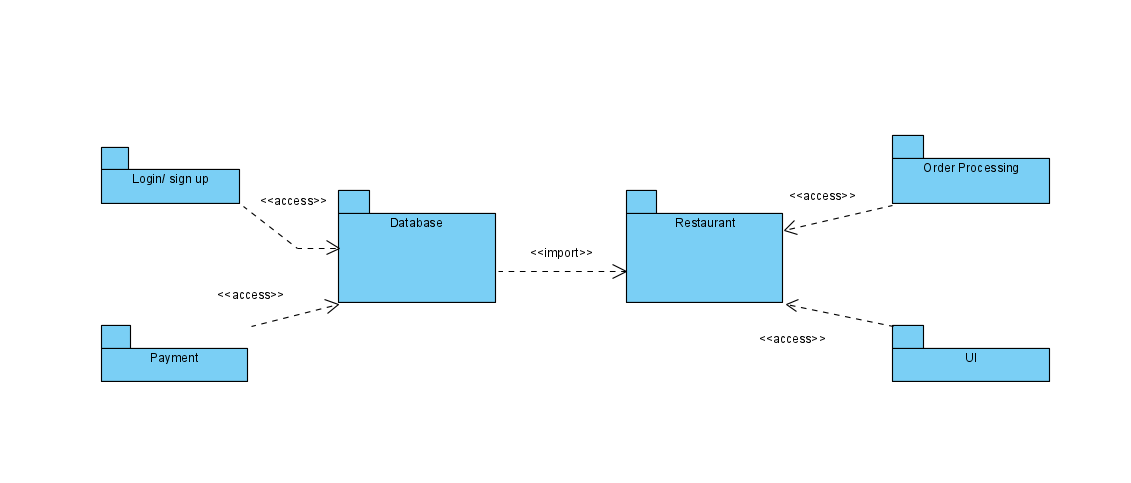
**sequence diagram**



* ***طرح 2***

در این طرح از استایل Data-centered برای معماری استفاده شده است.

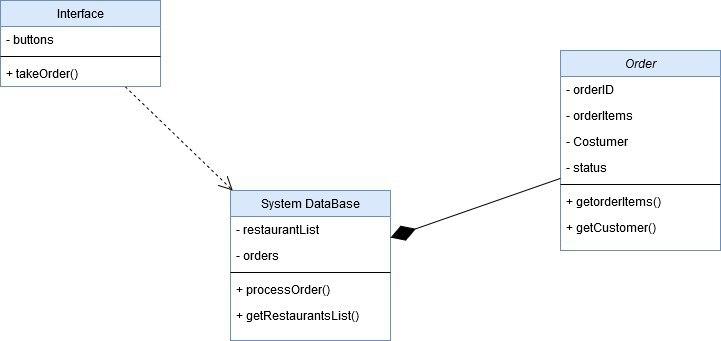
**Package diagram :**



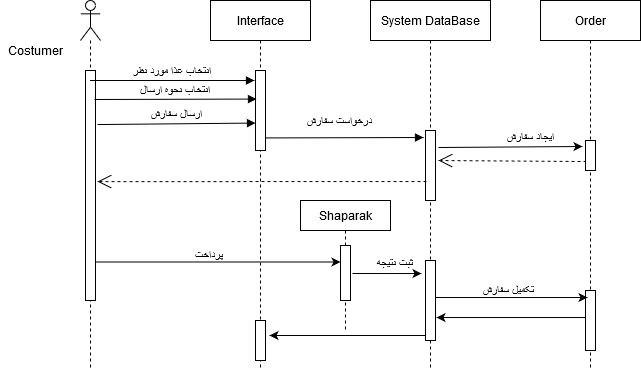
**deployment diagram :**

همان قبلی میباشد .

**class diagram :**

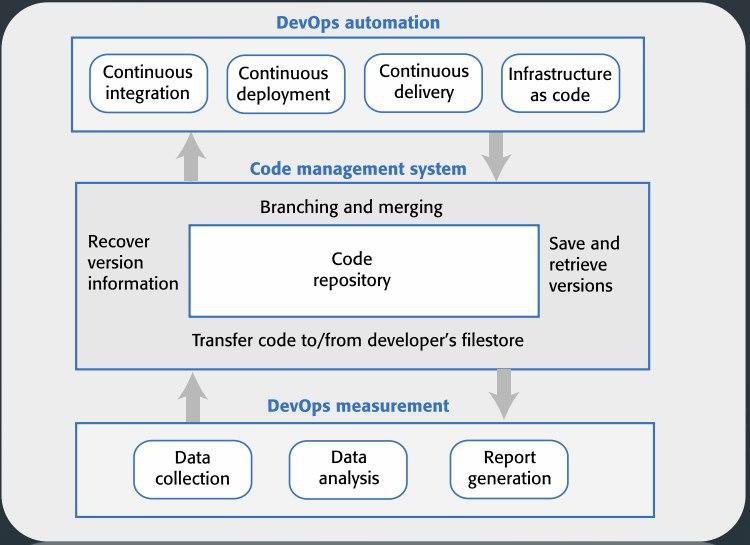


**sequence diagram :**

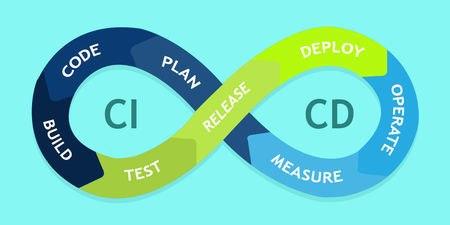


# **مدیریت پروژه**

## **از چه ابزاری برای مدیریت پیکره بندی و CD/CI استفاده میکنید ؟ چرا ؟**



در شکل بالا ، 3 بخش کلی Devops را میبینیم . که بخش بالایی ، لایه های مختلف automation را دارد یعنی همان CI/CD که میخواهیم ابزار های این قسمت را تعیین کنیم اما در ابتدا 3 بخش ابتدایی این قسمت را کمی توضیح میدهیم :



1. **CI (Continious Integration) :**

به طور خلاصه یعنی هر موقع تغییری رخ دهد ، باید بلافاصله تست شود و از صحت عملکرد آن اطمینان حاصل پیدا کنیم . طبیعتا این استراتژی باعث به حداقل رسیدن تعارض و ناسازگاری (conflict) بین جزء(component) های جدید و قدیم میشود .

1. **CD (Continious Delivery) :**

یعنی بتوانبم به ازای هر تغییری ، یک build از سیستم بگیریم که قابل ارایه به مشتری باشد علت هم این است که قبلا از پروژه تست های مختلف و زیادی گرفته شده است . (پروژه دایما آماده توسعه)

* توجه : در پروژه های بزرگ ، مرحله build گرفتن طولانی میشود ، لذا راهکاری که داریم این است که آنرا به بخش های کوچک میشکانیم و فقط آن بخش که تغییر داشته را build میگیریم .

در نتیجه با این استراتژی این قابلیت را خواهیم داشت تا با یک دوره زمانی خاصی اپلیکیشن را در دسترس کاربران واقعی قرار دهیم .

1. **CD (Continious Deployment ) :**

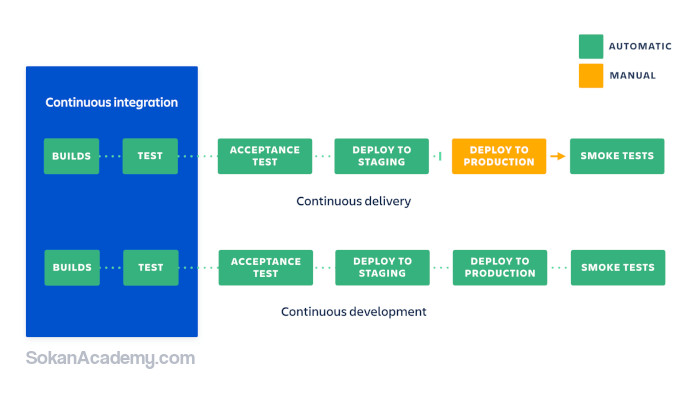
آن بخشی که تغییر داشته را build کرده ایم و حالا آن build را به production server میبریم .

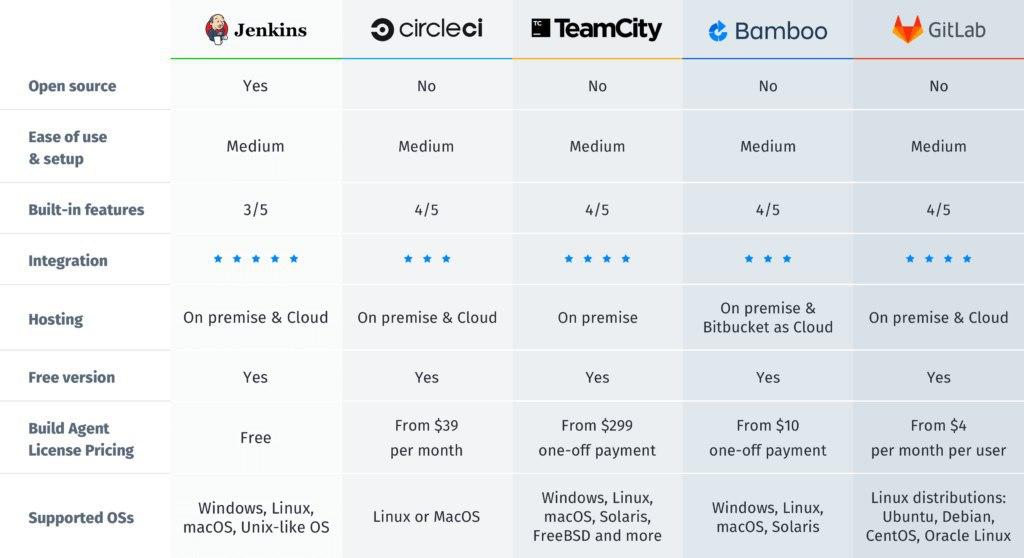
یعنی به صورت خودکار ، source code روی سرور ، deploy میشود . ( انتشار خودکار پروژه روی سرور)

در نتیجه این استراتژی میتوانیم سریع تر feedback های مرتبط با تغییرات را از کاربر بگیریم .

این CD همان CD ی قبلی است + اجرای خودکار

خلاصه :





در شکل بالا ابزار هایی که در سال 2019 برای CD/CI مناسب شناخته شده اند را آورده ایم .

[](https://jenkins.io/)

با بررسی هایی که انجام شد ، ابزار مناسب برای پروژه **Jenkins** انتخاب شد . دلایل :

* Jerkins ، یک ابزار متن باز (open source) است . و در ایران از نظر اخلاق حرفه ای مناسب به نظر میرسد .
* یکی از ویژگی های مهم این ابزار ، قابلیت تست به صورت relatime(بلادرنگ) است ؛ به طوریکه error های احتمالی را به صورت خودکار و سریع تشخیص میدهد . و این مورد باعث میشود تا تیم توسعه سریعا متوجه خطاهایشان شوند .
* Jerkins ، یک ابزار cross-platform است و روی سیستم عامل های linux/MAC OSX/ Windows قابلیت اجرا دارد .
* اینترفیس این نرم افزار user-friendly میباشد .
* کانفیگ کردن آن آسان میباشد.

## **در صورتی که از متدلوژی های چابک برای تولید نرم افزار خود استفاده میکنید ، موارد زیر را تکمیل کنید**

* + - * ***Sprint Map***
      * ***حداقل 10 تا Story ، در قالب فرمت همراه با تست پذیرش***
      * ***DoD***
      * ***DoR***
      * ***خروجی یکی از جلسات Sprint Review***

### **Sprint Map**

فعالیت هایی که درون هر Sprint باید انجام شوند :

* + - * + برنامه ریزی
        + پیاده سازی
        + Review

**تعیین ظرفیت تیم(velocity) :**

User story ها بر اساس اولویت و تلاش تخمینی(نفر/ساعت) دسته بندی شده اند :

* + - * **US01 : High - 2P/H**
      * **US02 : - 2P/H High**
      * **US03 : - 1P/H High**
      * **US04 :1P/H Mid -**
      * **US05 :1P/H Low -**
      * **US06 : Low - 1P/H**
      * **US07 : Low - 1P/H**
      * **US08 : High - 1P/H**
      * **US09 : High - 1P/H**
      * **US10 : Low - 2P/H**

لذا 3 تا Sprint 2 هفته ای در نظر میگیریم :

* + - * Sprint اول :

برنامه ریزی : User Story های 1،8و9 انجام میشود .

پیاده سازی : این مرحله انجام نشده است.

Review :

* + - * Sprint دوم :

برنامه ریزی : UserStory های 2،3،4و5 انجام میشود .

پیاده سازی : این مرحله انجام نشده است.

Review :

* + - * Sprint سوم :

برنامه ریزی : User Story های 6،7و10

پیاده سازی : این مرحله انجام نشده است.

Review :

توجه : برای انتخاب اینکه کدام user story در کدام sprint باشد ، 2 معیار اولویت و زمان(نفر/ساعت) در نظر گرفته شده است .

توجه : در بین User story های با اولویت کمتر هم اولویت بندی ریز تر و جزیی تری قایل شدیم .

### **User Stories**

|  |  |  |  |
| --- | --- | --- | --- |
| ID:US01 | | | |
| عنوان: مشاهده ی منو | | | |
| شرح: به عنوان مشتری میخواهم وقتی وارد حساب کاربری شدم، برای من نام تمام شعبات نمایش در آید و از میان آن ها حق انتخاب داشته باشم. سپس منوی شعبه ی انتخاب شده را برای من نمایش دهد. | | تاییدیه:  Scope – مشتری باید به حساب خود وارد شده باشد و منوی به نمایش در امده ، مربوط به شعبه ی انتخاب شده باشد. تمامی نام ها ، قیمت ها ، وضعیت و .... منو باید به درستی به نمایش در بیاید.  Not-Scope – درصورتی که کاربر به حساب مشتری وارد نشده باشد نباید اجازه و دسترسی این کار ها را داشته باشد. اطلاعات شعبات به نمایش در آمده نباید خارج از موارد وارد شده توسط مدیر در دیتابیس باشد و اطلاعات به درستی نمایش داده شود. همچنین در منوی شعبات اطلاعات هرغذا باید به درستی به نمایش در بیاید | |
| داستان های مربوط: وارد شدن به حساب کاربری(US08) ، خرید(US02) | | | |
| نویسنده: MAMOMASA | تاریخ: 27/06/99 | | اولویت: بالا |
| وضعیت: در دست تولید | زمان: 2 نفر/ ساعت | | سختی: متوسط |
| ابزار: | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID:US02 | | | |
| عنوان: خرید | | | |
| شرح: به عنوان مشتری که منو را مشاهده کرده است ، بتوانم یک تا چند غذا را بواسطه ی آیکن های + و – از سبد خرید خود اضافه و کم کنم. نهایتا با کلیک بر گزینه ی اتمام خرید ، به صفحه ی پرداخت منتقل شوم. پس از انجام پرداخت ، در صورتی که سفارش من بیرون بر و توسط پیک باشد ، مشخصات تماس پیک برای من نمایش داده شود و نهایتا بتوانم در نظرسنجی شرکت کنم. | | تاییدیه:  Scope – مشتری برای دسترسی به این سناریو باید سناریوی "مشاهده ی منو" را پشت سر گذاشته باشد. همچنین با فشردن چند باره ی + در کنار هر غذا همان غذا به همان تعداد به سبد خرید مشتری اضافه گردد و با هر بار فشردن – همان غذا به همان تعداد از سبد خرید مشتری کاسته شود. هزینه ی کسر شده از حساب مشتری درست باید به مقدار سفارش موجود در سبد کالای مشتری باشد و بیشتر و کمتر از آن نباشد. همچنین که پس از پرداخت شدن مبلغ توسط مشتری ، نظر سنجی برای رستوران به نمایش داده شود که مشتری بتوند یک گزینه ی آن را انتخاب نماید.  Not-Scope – تعداد غذاهای موجود در سبد خرید نمیتواند منفی باشد. هزینه ی قابل پرداخت برای مشتری نمیتواند منفی باشد. | |
| داستان های مربوط: مشاهده ی منو.(US01) | | | |
| نویسنده: MAMOMASA | تاریخ:27/06/99 | | اولویت: بالا |
| وضعیت: در دست تولید | زمان: 2 نفر/ساعت | | سختی: متوسط |
| ابزار: | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID:US03 | | | |
| عنوان: اعلام آمادگی سفارشات | | | |
| شرح: به عنوان یک آشپز پس از ورود به حساب ، با کلیک بر روی گزینه ی "سفارشات" ، تمامی سفارشات شعبه ی در حال خدمت ، برای من به نمایش در بیاید (نوع غذا و تعداد). با انتخاب سفارش مورد نظر و انتخاب گزینه ی "اصلاح وضعیت" بتوانم وضعیت سفارش را در حالت "آماده" تغییر دهم. | | تاییدیه:  Scope – آشپز باید به حساب کاربری خود وارد شده باشد که بتواند وارد این سناریو شده باشد. نوع و تعداد سفارشاتی که توسط مشتری ها پرداخت شده است باید قابل نمایش باشد نه سفارشاتی که پرداخت نشده است. پس از گزینه ی اصلاح وضعیت ، سفارش مورد نظر باید از صف یا پشته ی قابل مشاهده ی آشپز خارج گردد.  Not-scope – آشپز در صورتی که وارد حساب خود نشده باشد نمیتواند وارد این سناریو شده باشد. سفارشاتی که وضعیت آن ها به حالت آماده یا ... در آمده است نباید در این صف یا پشته قرار بگیرد و قابل مشاهده ی آشپز باشد. | |
| داستان های مربوط: خرید(US02) | | | |
| نویسنده: MAMOMASA | تاریخ:27/06/99 | | اولویت: بالا |
| وضعیت: در دست تولید | زمان: 1 نفر/ساعت | | سختی: پایین |
| ابزار | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID: US04 | | | |
| عنوان: ارسال با پیک | | | |
| شرح: به عنوان یک پیک ، پس از ورود به حساب باید بتوانم با انتخاب گزینه ی "مشاهده ی سفارشات" ، سفارشاتی که آماده ی تحویل هستند را مشاهده کنم و بتوانم یکی از آن ها را برای ارسال انتخاب نمایم. سپس به صفحه ی lock سفید رنگ هدایت میشوم که در آن مشخصات کامل سفارش و مشتری قابل مشاهده می باشد. | | تاییدیه:  Scope – پیک باید به حساب کاربری خود وارد شده باشد تا بتواند وارد این سناریو شده باشد. همچنین که سفارشاتی که توسط آشپز به حالت آماده در آمده است و همچنین مشتریِ مالک آن سفارش گزینه ی ارسال با پیک را زده است باید در این صف یا پشته برای پیک قابل مشاهده باشد. اطلاعات مشتری به همراه آدرس و شماره ی تلفن همراه مشتری با مشتری سفارش دهنده ی آن سفارش یکسان باشد.  Not-Scope – سفارشاتی که برای "ارسال با پیک" انتخاب نشده باشند و توسط آشپز تغییر وضعیت داده نشده باشند نباید در این صف یا پشته قرار گیرد. پیک مادامی که سفارش را تحویل نداده است نباید بتواند کار دیگری با اپلیکیشن انجام دهد. | |
| داستان های مربوط: | | | |
| نویسنده: MAMOMASA | تاریخ: 27/06/99 | | اولویت: متوسط |
| وضعیت: در دست تولید | زمان: 1نفر/ساعت | | سختی: متوسط |
| ابزار | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID: US05 | | | |
| عنوان: مدیریت کارمندان | | | |
| شرح: به عنوان یک مدیر ، پس از ورود به حساب باید بتوانم با انتخاب "مدیریت کارمندان" ، حساب های آن ها را مشاهده کنم و در صورت نیاز بر روی گزینه ی اصلاح کلیک کرده تا بتوانم اطلاعات کارمند انتخاب شده را ویرایش کنم. همچنین با انتخاب گزینه ی "افزودن" بتوانم یک کارمند را به سیستم اضافه نمایم. | | تاییدیه:  Scope – مدیر باید وارد حساب کاربری خود شده باشد تا بتواند وارد این سناریو شده باشد. تمامی کارمندان همراه با مشخصات کامل آن ها باید برای مدیر قابل نمایش باشد. همچنین با وارد کردن کارمند جدید باید لیست کارمندان در دیتابیس و بار دیگری که مدیر وارد این صفحه میشود اضافه شده باشد. همچنین برای کارمندان اصلاح شده نیز این حکم برقرار است.  Not-Scope – درصورتی که کارمند افزوده شده اطلاعات یکسانی با کارمندان ثبت شده داشته باشد ، باید پیغام خطایی نمایش داده شود مبنی بر آنکه کارمند وارد شده قبل تر در سیستم وارد شده است. همچنین پس از ویرایش کردن یک کارمند باید اطلاعات ویرایش شده با دیگر کارمندان یکسان نباشد. | |
| داستان های مربوط: | | | |
| نویسنده: MAMOMASA | تاریخ: 27/06/99 | | اولویت: پایین |
| وضعیت: در دست تولید | زمان:1نفر/ساعت | | سختی: متوسط |
| ابزار | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID: US06 | | | |
| عنوان: مشاهده ی عملکرد شرکت | | | |
| شرح: به عنوان یک مدیر یا حسابدار ، باید بتوانم پس از ورود به حساب با انتخاب گزینه ی "مشاهده ی عملکرد" ، عملکرد شرکت را در این چند وقت اخیر بر حسب تفکیک شعبات و در قالب نمودار ها و جداول دسته بندی شده مشاهده کنم. | | تاییدیه:  Scope – فقط مدیر یا حسابدار پس از ورود به حساب کاربری خود میتوانند وارد این سناریو گردند. همچنین که تمامی تراکنش های رخ داده در این چند وقت اخیر باید به نمایش در آورد.  Not-Scope – اطلاعات شعبات نباید در هم ادغام شده باشد و یا اطلاعات نشان داده شده مربوط به شعبه ی مورد نظر نباشد. | |
| داستان های مربوط: | | | |
| نویسنده: MAMOMASA | تاریخ: 27/06/99 | | اولویت: پایین |
| وضعیت: در دست تولید | زمان:1 نفر/ساعت | | سختی: پایین |
| ابزار | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID: US07 | | | |
| عنوان: اصلاح حقوق کارمندان | | | |
| شرح: به عنوان یک حسابدار ، باید بتوانم پس از ورود به حساب ، با انتخاب گزینه ی "اصلاح حقوق کارمندان" ، حساب کارمندان را همراه با حقوق دریافتی شان ، مشاهده کنم و با انتخاب هر کدام از آن ها بتوانم حقوق آن ها را اصلاح نمایم. | | تاییدیه:  Scope – حسابدار باید ابتدا به حساب خود وارد شده باشد. همچنین که پس از ویرایش حقوق کارمند مورد نظر باید اطلاعات به درستی به دیتابیس رفته و در صورت ورود مجدد به این بخش اطلاعات ویرایش شده نمایش داده شود.  Not-scope – حقوق کارمند مورد نظر نمیتواند مقدار مفنی به خود بگیرد. همچنین تغییر در حقوق یک کارمند نباید تاثیری بر حقوق دیگر کامندان شرکت بگذارد و تنها باید حقوق فرد مورد نظر تغییر کرده باشد. | |
| داستان های مربوط: | | | |
| نویسنده: MAMOMASA | تاریخ: 27/06/99 | | اولویت: پایین |
| وضعیت: در دست تولید | زمان: 1 نفر/ساعت | | سختی: پایین |
| ابزار | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID: US08 | | | |
| عنوان: ورود | | | |
| شرح: به عنوان یک کاربر اپلیکیشن ، باید بتوانم هر بار که وارد اپلیکیشن میشوم هنگام ورود به حساب کاربری ، با پر کردن فیلدهای یوزرنیم و پسورود خود و انتخاب گزینه ی "ورود" ، وارد حساب کاربری شوم. | | تاییدیه:  Scope – اطلاعات وارد شده توسط کاربر باید با اطلاعات موجود در دیتابیس یکسان باشد.  Not-scope – درصورتی که اطلاعات مورد نظر با دیتابیس مغایرت داشت ، باید پیغام مناسبی نمایش داده شود. | |
| داستان های مربوط: | | | |
| نویسنده: MAMOMASA | تاریخ: 27/06/99 | | اولویت: بالا |
| وضعیت: در دست تولید | زمان:1 نفر/ساعت | | سختی: متوسط |
| ابزار | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID: US09 | | | |
| عنوان: ثبت نام | | | |
| شرح: به عنوان یک کاربر برای ایجاد حساب مشتری ، بعد از ورود به اپلیکیشن و انتخاب گزینه ی "ثبت نام" ، وارد صفحه ای بشوم که از من ، اطلاعات کاربری ام را بخواهد و با وارد کردن فیلد های مربوطه و فشردن گزینه ی "تکمیل ثبت نام" ، فرآیند ثبت نام من کامل گردد. و بعد به صفحه login هدایت شوم . | | تاییدیه:  Scope – فیلد ها باید قابلیت ویرایش مجدد را داشته باشند و با زدن گزینه تکمیل ثبت نام ، اطلاعات کاربر باید وارد سیستم (DB)شده باشد و بتواند بار دیگر در سیستم ورود کند  (save token in back-end)  . Not-scope – درصورتی که اطلاعات وارد شده برای ثبت نام کاربر با کاربران ثبت شده در گذشته یکسان باشد ، نباید اجازه ی تکمیل ثبت نام به مشتری داده شود و باید پیغام مناسبی برای کاربر به نمایش در آید: "شما قبلا ثبت نام کرده اید" | |
| داستان های مربوط: | | | |
| نویسنده: MAMOMASA | تاریخ: 27/06/99 | | اولویت: بالا |
| وضعیت: در دست تولید | زمان:1 نفر/ساعت | | سختی: متوسط |
| ابزار | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID: US10 | | | |
| عنوان: بازیابی رمز عبور | | | |
| شرح: به عنوان یک کاربر مشتری ، با ورود به اپلیکیشن و انتخاب گزینه ی "فراموشی رمز عبور" به صفحه ای وارد شوم که درون آن شماره ی همراه من را بخواهد ، با وارد کردن شماره ی همراه و انتخاب گزینه ی "تایید شماره ی همراه" ، یک کد امنیتی برای تلفن همراه من ارسال گردد که باید در صفحه ی بعدی آن را وارد کنم. در صورت صحت کد امنیتی وارد شده ، به صفحه ی مربوطه برای ایجاد رمز عبور جدید هدایت شوم. رمز عبور خود را وارد کنم و گزینه ی ذخیره را انتخاب کنم. | | تاییدیه:  Scope – شماره ی همراه باید در سیستم با شماره ی همراه یکی از کاربران موجود در سیستم یکسان باشد تا پیامک برای شخص مورد نظر ارسال گردد. همچنین این اطلاعات پیامک باید در حدود 1 دقیقه برای کاربر فرستاده شود. پس از نوشتن رمز عبور جدید و انتخاب "ذخیره" باید اطلاعات جدید جایگزین اطلاعات قبلی در سیستم گردد.  Not-scope – درصورتی که شماره ی همراه وارد شده با شماره ی همراه هیچ یک از کاربران موجود در سیتسم مطابقت نداشت باید پیغام مناسبی برای کاربر به نمایش در آید. همچنین اگر کد امنیتی ارسال شده برای کاربر با کد امنیتی وارد شده از سوی او مطابقت نداشت باید پیغام خطای مناسبی نمایش داده شود. | |
| داستان های مربوط: US08 | | | |
| نویسنده: MAMOMASA | تاریخ: 27/06/99 | | اولویت: پایین |
| وضعیت: در دست تولید | زمان: 2 نفر/ساعت | | سختی: بالا |
| ابزار | | | |

### **DoD**

******

* + - * sprint اول :

تست کردن اتوماتیک

توافق بین اعضا مبنی بر تکمیل شدن code در این sprint

مالک محصول اجازه ادامه کار دهد

بررسی Response Time ( کارکرد بالاتر از 1 ثانیه مجاز نیست)

نیازمند Sprint دوم

integrate شدن کد با master branch

* + - * Sprint دوم :

تست کردن اتوماتیک

توافق بین اعضا مبنی بر تکمیل شدن code در این sprint

مالک محصول اجازه ادامه کار دهد

بررسی Response Time

[[1]](#footnote-1)Cyclomatic Complexity از حد انتظار کمتر باشد

integrate شدن کد با master branch

* + - * Sprint سوم :

تست کردن اتوماتیک

توافق بین اعضا مبنی بر تکمیل شدن code در این sprint

مالک محصول اجازه ادامه کار دهد

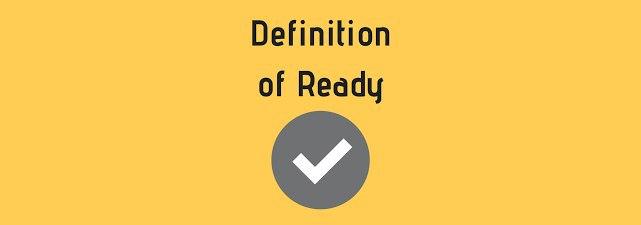
بررسی Response Time

Refactor کردن کدها

integrate شدن کد با master branch

deploy کردن کد

### **DoR**



معنای DoR (Definition of Ready) :

قبل از اینکه Sprint شروع شود ، بهتر است که یک check-list تهیه شود و در آن مشخص شود که کدام یک از کار ها قرار هست که روی PBI(Product backlog item) انجام شود . البته این جزو راهنمای Scrum نیست ولی توصیه شده است که انجام شود .

وظیفه تکمیل این لیست به عهده Product owner است .

* + - * Sprint اول :

صحبت راجع به Story های این Sprint بین تیم توسعه و مالک محصول با هدف رفع ابهام از مفهوم Story به جهت تکمیل و انجام PBI

بررسی ارزش محصول از نظر business

تخمین تلاش مورد نیاز در طی Sprint

مشخص کردن وابسنگی ها به گونه ای که وابستگی های خارجی جلوی کار را نگیرد

* + - * Sprint دوم :

تخمین تلاش مورد نیاز در طی Sprint

Story به اندازه کافی شکسته شود تا در داخل یک sprint جا شود

مشخص کردن Acceptance Criteria (معیار پذیرش)

مشخص کردن اینکه تیم به اندازه کافی توانایی دارد تا آن Sprint را انجام دهد

* + - * Sprint سوم :

تخمین تلاش مورد نیاز در طی Sprint

بررسی ارزش محصول از نظر business

مشخص کردن Acceptance Criteria (معیار پذیرش)

### **خروجی یکی از جلسات Sprint Review**

تیم ما فرصت پیاده سازی را نداشته است لذا جلسه Sprint Review تشکیل نشد.

**Technical Review**

* + - * شماره : 3
      * مسیول جلسه : مهدی نیک نژاد
      * تاریخ جلسه : 01/05/1399
      * نظرات و جمع بندی ها :

این جلسه به مدت 80 دقیقه طول کشید ، در مورد چند ابهام و چند نظر جدید صحبت شد .

* بیان شد که آیا ذکر "جزییات" در مورد قابلیت های محصول لازم هست یا خیر . که در نتیجه تصمیم بر این شد که در این بخش زیاد به جزییات پرداخته نشود .
* کمی ابهام در مورد نقش پیک وجود داشت در مورد 2 وظیفه این نقش . یکی تحویل سفارش مشتریان و دیگری تهیه مواد اولیه رستوران . در این مورد بحث شد که آیا بهتر نیست که این نقش را به 2 مورد تقسیم کنیم یکی پیک تحویل غذا و دیگری پیک تهیه مواد اولیه . اما نتیجه بر آن شد که پروژه را تا این حد خرد نکنیم و صرفا یک مدل پیک داشته باشیم . با توجه به جلسه قبلی ای که داشتیم بررسی شد که پیک باید با انباردار هم در ارتباط باشد. و یک نکته کوچک هم در مورد نام انبار دار که supplier انتخاب شده است عنوان شد که بهتر است نامش storekeeper باشد .
* همانطور که در جلسه قبل بیان شد ، پرداخت و ویرایش حقوق کارمندان (بعد از اعمال ویرایش نهایی حسابدار) به عهده مدیر و فقط *ویرایش* حقوق کارمندان به عهده حسابدار است .
* در ضمن در مورد "تخفیف" روی غذا ها هم که در قابلیت ها عنوان شده بود بحث شد که حذف گردد .
* در ضمن یکی از دوستان در مورد "گزینه بیرون بر" در اپلیکیشن بحث داشت و کلا با داشتن این feature مخالف بود و استدلالش هم این بود که اگر کاربری قرار هست که از این اپلیکیشن استفاده کند احتمالا در داخل رستوران حضور ندارد و لذا باید طبیعتا از پیک برای دریافت غذا استفاده کند ؛ اما برایش توضیح داده شد که پروژه ای که داریم انجام میدهیم چیزی شبیه به اسنپ فود نیست بلکه فرضا به این صورت هست که کاربر با این اپلیکیشن ، سفارش خود را ثبت میکند و بعد اگر گزینه بیرون بر را تیک بزند که پیک برایش غذت رت می آورد و اگر هم که این گزینه را تیک نزده باشد ، باید خودش به رستوران مراجعه کند و سفارشش را بگیرد – وقتی که وضعیت سفارش تکمیل شده بود یعنی "غذا حاضر است" شده بود -
  + - * + پس نتیجه گیری دوستان در مورد review قبل به این صورت شد در مورد قالب آشپز :

آشپز به سیستم اعلام میکند که غذا حاضر است و حالا سیستم باید بر اساس آن تیک که کاربر زده است تصمیم بگیرد که عملکرد مناسب داشته باشد ؛ یعنی اگر تیک گزینه بیرون بر زده شده بود ، سیستم به کاربر پیام دهد که غذا در حال ارسال است و به پیک هم پیام دهد تا غذا از آشپز دریافت و به کاربر تحویل داده شود و اگر که این گزینه زده نشده باشد باید به مشتری پیام دهد که غذا حاضر است و آماده تحویل میباشد .

* بحث دیگری که پیش آمد این بود که آیا حسابدار هر شعبه توانایی و اجازه مشاهده دخل و خرج سایر شعبات را دارد یا خیر که به نتیجه رسیدیم که این اجازه داده شود برای بحث رقابت بین شعبات با هم دیگر و مشاهده پیشرفت و پسرفت یکدیگر .
* بحث دیگری که ابهام داشت در مورد قابلیت تایید حساب کاربری کارکنان بود که اجازه اش به حسابدار هم داده شود اما با تصمیم جمع این امکان فقط در انحصار مدیر میباشد .
* در بخش ویرایش حقوق کارکنان ( هم وظیفه مدیر و هم وظیفه حسابدار) این بحث پیش آمد که مبلغ وارد شده میتواند به عنوان حکم جربمه کارمند منفی وارد شود و اینکه " نباید مبلغ وارد شده منفی باشد" غلط میباشد اما مسیول این قسمت بیان کرد که در نظرش این طور بوده که یک حقوق ثابت برای یک کارمند در نظر نمیگیریم بلکه هر ماه هنگام واریز حقوق کارمندان ، میزان حقوق هر کارمند آپدیت میشود و طبیعتا نمیتواند مقدار منفی باشد و جمع هم نظر را پذیرفت .
* در ضمن بحثی هم پیش آمد که شاید خود دیتابیس طوری نوشته شود که یک record با اطلاعات کاملا یکسان را نپذیرد اما ما باید آنرا handle کنیم و از وارد کردن اطلاعات یکسان جلوگیری کنیم . (Eception Handling)
* بیان شد که میتوانیم برای بحث احراز هویت از سیستم پیامکی مرسوم (مثل آپارات ، مکتبخونه و ... ) استفاده کنیم که کاربر فقط شماره تلفن را وارد میکند و بعد پیامکی برایش می آید و آنرا وارد میکند در اپلیکیشن وبعد وارد سایت میشود . و اطلاعات قبلی برای ورود دیگر نیاز نیست مثل نام کاربری و رمز عبور . در واقع داریم از یک رمز عبوریکبار مصرف استفاده میکنیم. که با تصمیم جمع نهایی شد که اگر بنا بود پروژه نسخه beta ای داشته باشد در آنجا این مورد را بحث کنیم.
* چندین سوال هم پیش آمد که باید در جلسه بعد مطرح میشد راجع به use case diagram .
  + - مورد sign up و log in قرار هست که تغیر کند .
    - علت اینکه هم user داریم و هم customer در ابتدا مورد داشت اما بعدا متوجه شدیم که هر کارمندی یک user است و از ارث بری استفاده شده است.
    - آیا ارور ها یعنی error message ها برای sign up و log in فرق داشته که جا کشیده شده ؟ چون میشد فقط یک use case باشد برای آن .
    - راجع به cash flow(جریان پرداخت) توضیح دهید لطفا.و همچنین جهت cash flow به branches under control نباید برعکس باشد؟
    - در بالا هم گفتیم که حسابدار اینقدر دسترسی دارد که دخل و خرج سایر شعبات را میتواند مشاهده کند .چرا؟ از نطر بیزینس که این غلطه !
    - رابطه بین orders و edit status of orders نباید include یا extend باشه ؟
    - چرا edit wages (ویرایش حقوق کارمندان)به Restaurant وصل شده است ؟ مگر نباید به system database وصل شود؟ و اصلا system db , Restaurant , account db چی هستند و چه فرقی دارند؟
    - بین employees و employee wages نباید رابطه extend باشد؟ و همچنین علت 2 تا پشت سرهم extend چی هست ؟ fire/hire --------- employees -------- branches
* موضوع خرید از نزدیکترین شعبه رستوران توسط کاربر هم مطرح شد که به نسخه Beta سپرده شد .

**Technical Review**

* + - * شماره : 4
      * مسیول جلسه : سعید معروف
      * تاریخ جلسه : 02/05/1399
      * نظرات و جمع بندی ها :

اینبار به مسئله ی Use case ها برگشتیم و از آن به بعد را بررسی کردیم که در ادامه بصورت مفصل آورده شده است.

* در ابتدا به این مسئله پرداختیم که آیا Database را بصورت global در نظر بگیریم و costumer و دیگران به یک دیتابیس مشترک شوند یا خیر. با بحثی که شد ، به دلیل اینکه Costumer نیازی نیست به اطلاعات دیتابیس اصلی رستوران دسترسی داشته باشد و اینکه security و نفوذ به سیستم اصلی را از این جهت بگیریم ، تصمیم بر آن شد که دو دیتابیس اصلی داشته باشیم. یکی Restaurant و یکی هم system database. به Restaurant که داده های اصلی در آن وجود دارد مشتری نمیتواند به هیچ عنوان دسترسی پیدا کند.
* در قسمت بعدی چند رابطه در Use case ها بهتر بود اصلاح شود که اصلاح شد. بدین شرح که میان see order و Edit آن ها توسط آشپز بهتر بود که رابطه ی Extend باشد. چراکه آشپز لزوما وقتی سفارشات را میبیند تغییر وضعیت به "آماده شد" برای آن ها درج نمیکند. یا میتواند برای آمار گیری کلی بخواهد order ها را مشاهده کند. پس لزوما هر see order به edit آن ها ختم نمیشود.(لزومی ندارد)
* سپس در Use case ، رابطه ی میان Sing up و create employee account یک رابطه ی inheritance بود که بسار زیاد در این قسمت صحبت کردیم. نهایتا بر آن شدیم که این رابطه را از میان آن ها برداریم. در اصل صحبت حول این موضوع بود که چون employee ها صرفا یک سری فیلد های بیشتری نسبت به Sign up مشتری دارند بهتر است که از آن ارث بری داشته باشیم یا خیر. ولی از طرفی گفتیم که ممکن است بخواهیم بعد ها و در ورژن های بعدی نرم افزار قابلیت احراز هویت(authentication) را اضافه کنیم که وارد sing up میشود. یعنی هنگام sing up به شماره ی همراه وارد شده توسط کاربر یک پیامک ارسال کنیم تا از ربات نبودن و صحت اطلاعات شماره ی تلفن همراه وی باخبر شویم. پس چون ممکن است sing up در ورژن های بعدی فابلیت هایی را اضافه به خود داشته باشد ، بهتر است از create employee account مستقل باشد و رابطه ای میان آن دو نباشد.
* در قسمت Actor ها نیز یک Actor به خود اضافه کرد به این شرح که Actor ای با نام Account که پدر تمام Actor های دیگر است و از ان ارث بری دارند پدید آمد که در اصل کار خاصی نمیکند و Act انجام نمیدهد.

در انتها نیز مباحثی نیز در قسمت طراحی استایل های معماری کج فهمی داشت که تکمیل شد.

**Technical Review**

* + - * شماره : 5
      * مسیول جلسه :
      * تاریخ جلسه : 10/05/1399
      * نظرات و جمع بندی ها :

در این جلسه که جلسه ی آخر هست و پروژه نسبتا کامل شده و آماده ی ارسال می باشد ؛ آخرین مطالب و نظرات را مطرح کردیم و خلاصه ای از این مطالب در ادامه آمده است.

* + - * + ابتدا به Activity Diagram مجدد رجوع شد و درمورد آن صحبت شد که بدلیل اینکه State ها و مراحل دقیقا با Sequence یکسان نیستند و جزئیاتی در این دو متفاوت است. مشکلی که یکی از دوستان مطرح کرد این بود که افعال نسبت به یک شخص نوشته شده است و با این کار در sequence همه ی آن مراحل زیر یک ستون Actor قرار خواهند گرفت. لذا بهتر است که افعال را تغییر دهیم و بطور مثال بجای "مشاهده ی منوی سفارش ها" ، بنویسیم "نمایش منوی سفارش ها" که Actor آن سیستم و UI خواهند بود. ولی از آن سو بحث شد که اگر این اتفاق بیفتد ، افعال مجهول مانند خواهند بود و فاعل آن ها مشخص نخواهند بود و دچار گنگی و ابهام خواهد شد. فلذا تصمیم نهایی بر آن شد که آن را بدون تغییر رها سازیم. در Sequence اما تغییرات اعمال شد چراکه Actor ها در آنجا مشخص است و فاعلی مجهول نخواهد ماند.
        + درمورد interface در نمودار sequence نیز صحبت شد و درمورد کارآیی آن بحث شد. گفته بودیم که شاید بهتر باشد UI فقط مربوط به interface باشد و قسمت logical را هندل نکند و از آن مجزا باشد. با بررسی هایی که در فضای اینترنت و مباحثه و نظر های دوستان انجام شد ، نهایتا تصمیم بر آن شد که برای کاهش پیچیدگی و کاهش بار سرور اصلی ، یک سری از افعال و اعمال Logical را بر UI محول نماییم که جزئی بود.
        + سپس درمورد component diagram صحبت کردیم که در ابتدا دارای دو کامپوننت نسبتا شبیه به هم یعنی order service و add order بود. اولی برای اعمالی بود که بین خود کارمندان شرکت یا رستوران انجام میشد برای اینکه یک سفارش به دست مشتری برسد و پروسه اتمام شود و دومی برای آن بود که مشتری یک سفارش را به شرکت یا رستوران بفرستند و از آن بخواهد. با صحبت هایی که میان دوستان صورت گرفت و نظر جمع تصمیم بر آن شد که بدلیل ارتباط زیاد و غایت یکسانی که این دو کامپوننت در نظر دارند ، آن دو کامپوننت به یک کامپوننت به نام Order تبدیل شود که تمام پروسه ی ذکر شده از سفارش مشتری تا آماده شدن کامل سفارش را برعهده داشته باشد.

1. Number of pathes in code segment [↑](#footnote-ref-1)