



دانشگاه آزاد اسلامی واحد تهران غرب

گروه کامپیوتر

گزارش و مستندات پروژه

سیستم مدیریت رستوران

استاد پروژه: سرکار خانم سلیمانلود

دانشجو: عباس اللهیاری (۸۷۰۷۱۹۳۵۷)

کاردانی کامپیوتر پیوسته

نیمسال اول ۸۹-۹۰

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

## فهرست مطالب

۱۳.....	شرح سیستم
۱۳.....	مشخصات فنی
۱۳.....	سیستم مورد نیاز برای اجرای سیستم
۱۳.....	خلاصه امکانات نرم افزار
۱۴.....	موجودیت ها
۱۵.....	موجودیت و جدول «مشتري»
۱۶.....	موجودیت و جدول «دسته بندی»
۱۶.....	موجودیت و جدول «غذا»
۱۷.....	موجودیت و جدول «کارمند»
۱۸.....	موجودیت و جدول «سفارش»
۱۹.....	جدول Foods_Orders
۲۰.....	اینترفیس IBaseModel
۲۳.....	کلاس BaseModel
۲۴.....	کلاس Category
۲۵.....	کلاس Food

۲۶.....	Employee	کلاس
۲۷.....	Customer	کلاس
۲۸.....	Order	کلاس
۳۰.....		پیاده‌سازی
۳۰.....		پیاده‌سازی کلاس‌های کمکی
۳۰.....	Common	کلاس
۳۰.....	SHA1	متد
۳۰.....	InitializeDatabase	متد
۳۱.....		کدهای SQL جداول
۳۴.....	HandelSqlCeException	متد
۳۶.....	Initialize	متد
۳۷.....	ChangeLanguage	متد
۳۷.....	ToggleRightToLeft	متد
۳۸.....	ShowDeletePrompt	متد
۳۹.....	ToggleControlsDisableEnable	متد
۳۹.....	InitializeErrorHandles	متد
۴۰.....	InitializeList	متد
۴۱.....	IsActiveTab	متد
۴۱.....	ShowErrorMessage	متد
۴۲.....	AddDatagridViewColumn	متد
۴۲.....	CommonDataGridCellBeginEdit	متد
۴۳.....	CommonDataGridSubmitEdit	متد
۴۳.....	CommonDiscardChanges	متد

۴۴.....	متد PopulateLanguageMenu
۴۶.....	متد CommonDataGridCellEndEdit
۴۹.....	پیاده‌سازی IBaseModel
۵۰.....	پیاده‌سازی کلاس BaseModel
۵۲.....	پیاده‌سازی رویداد ها و متد های مربوطه
۵۴.....	پیاده‌سازی متد سازنده
۵۴.....	پیاده‌سازی متد getList
۵۵.....	پیاده‌سازی متد ReadAll
۵۵.....	پیاده‌سازی متد Read
۵۶.....	پیاده‌سازی متد Save
۵۷.....	پیاده‌سازی متد Save با ورودی DataRow
۵۸.....	پیاده‌سازی متد Delete
۵۸.....	پیاده‌سازی متد Edit
۵۹.....	پیاده‌سازی متد Reload
۶۰.....	پیاده‌سازی متد Clone
۶۰.....	پیاده‌سازی کلاس Category
۶۰.....	پیاده‌سازی متد سازنده
۶۱.....	پیاده‌سازی متد Read
۶۲.....	پیاده‌سازی متد ReadFoods
۶۲.....	پیاده‌سازی متد GetHashCode
۶۳.....	پیاده‌سازی متد Equals با ورودی Category
۶۴.....	پیاده‌سازی متد Equals با ورودی object
۶۴.....	پیاده‌سازی متد Validate با ورودی Category
۶۴.....	پیاده‌سازی متد Validate با ورودی object
۶۵.....	پیاده‌سازی متد Fill با ورودی Category و DataRow
۶۵.....	پیاده‌سازی متد Fill با ورودی DataRow

۶۵.....	پایاده سازی پراپرتی CategoryName
۶۶.....	پایاده سازی پراپرتی dataTable
۶۷.....	پایاده سازی کلاس Food
۶۷.....	پایاده سازی متد سازنده
۶۸.....	پایاده سازی متد Read
۶۸.....	پایاده سازی متد ReadByCategory
۶۹.....	پایاده سازی متد Search
۷۰.....	پایاده سازی متدهای IncreaseQuantity و DecreaseQuantity
۷۱.....	پایاده سازی دیگر اعضا
۷۱.....	پایاده سازی کلاس Employee
۷۱.....	پایاده سازی متد سازنده
۷۲.....	پایاده سازی پراپرتی FullName
۷۲.....	پایاده سازی متد Authenticate
۷۳.....	پایاده سازی متد Search
۷۳.....	پایاده سازی اعضای دیگر
۷۴.....	پایاده سازی کلاس Customer
۷۴.....	پایاده سازی متد سازنده
۷۵.....	پایاده سازی متد Search
۷۵.....	پایاده سازی متد ReadOrders
۷۶.....	پایاده سازی متد DeleteOrders
۷۷.....	پایاده سازی متد FilterOrders
۷۷.....	پایاده سازی متد ReadOrdersByType
۷۷.....	پایاده سازی متد ReadOrdersByStatus
۷۷.....	پایاده سازی متد getCustomersId
۷۸.....	پایاده سازی پراپرتی CustomerId
۷۸.....	پایاده سازی اعضای دیگر

۷۹.....	Order	پیاده سازی کلاس
۷۹.....		پیاده سازی متد سازنده
۸۰.....	FirstOrder	پیاده سازی پراپرتی
۸۱.....	LastOrder	پیاده سازی پراپرتی
۸۱.....	RawPrice	پیاده سازی پراپرتی
۸۱.....	DiscountedPrice	پیاده سازی پراپرتی
۸۲.....	TotalPrice	پیاده سازی پراپرتی
۸۲.....	InitializeFoodsTable	پیاده سازی متد
۸۳.....	ReadFoods	پیاده سازی متد
۸۴.....	SaveFoods	پیاده سازی متد
۸۵.....	SaveFood	پیاده سازی متد
۸۵.....	DeleteFoods	پیاده سازی متد
۸۵.....	DeleteFood	پیاده سازی متد
۸۵.....		پیاده سازی متد های آمار گیری
۸۷.....		پیاده سازی اعضای دیگر
۸۸.....		پیاده سازی فرم ها
۸۸.....	frmLogin	فرم
۸۸.....	InitializeLanguage	متد
۸۹.....	frmLogin_Load	رویداد
۸۹.....	frmLogin_FormClosed	رویداد
۸۹.....	frmLogin	متد
۹۰.....	btnLogin_Click	رویداد
۹۲.....	frmManagement	فرم
۹۲.....	frmManagement	متد
۹۳.....	InitializeFoodsDataGrid	متد

۹۳.....	متد HideSuperAdmin
۹۴.....	متد InitializeEmployeeRole
۹۴.....	رویداد comboCategory_SelectedIndexChanged
۹۵.....	متد RebindFoodsDataGrid
۹۶.....	عمل حذف
۹۶.....	متد Search
۹۶.....	متد DiscardChanges
۹۶.....	رویداد dgFood_UserAddedRow
۹۷.....	متد SaveNewRows
۹۸.....	متد SaveModifiedRows
۹۸.....	رویداد dgFood_CellBeginEdit
۹۸.....	رویداد dgFood_CellEndEdit
۹۹.....	آشنایی محیط نرم افزار و نحوه کار با آن
۹۹.....	صفحه ورود
۹۹.....	پنل مدیریت
۱۰۰.....	امکانات پنل مدیریت
۱۰۱.....	منوها
۱۰۲.....	تولبار اصلی
۱۰۳.....	دسته بندی و جستجو
۱۰۳.....	تب «غذا»
۱۰۳.....	مدیریت دسته بندی ها
۱۰۳.....	دسته بندی جدید
۱۰۴.....	حذف دسته بندی
۱۰۴.....	ویرایش دسته بندی



- ۱۰۴..... اضافه کردن غذا
- ۱۰۵..... حذف غذا
- ۱۰۶..... تغییر دسته بندی
- ۱۰۶..... دسته بندی و جستجو
- ۱۰۷..... تب «کارمند»
- ۱۰۷..... اضافه کردن «کارمند»
- ۱۰۷..... جستجو
- ۱۰۷..... آمار
- ۱۱۰..... پنل اصلی یا سفارش
- ۱۱۱..... تولبار اصلی
- ۱۱۱..... مدیریت مشتری ها
- ۱۱۱..... اضافه کردن «مشتری»
- ۱۱۱..... حذف و ویرایش مشتری
- ۱۱۲..... مدیریت سفارش ها
- ۱۱۲..... ثبت سفارش
- ۱۱۴..... مشاهده سفارش
- ۱۱۵..... فیلتر کردن سفارش ها
- ۱۱۶..... حذف سفارش
- ۱۱۶..... ویرایش سفارش
- ۱۱۷..... گزارش گیری
- ۱۱۸..... ترجمه نرم افزار

## فهرست اشکال

- شکل ۱: دیاگرام EER سیستم مدیریت رستوران ..... ۱۴
- شکل ۲: کلاس دیاگرام سیستم مدیریت رستوران ..... ۱۵
- شکل ۳: کلاس دیاگرام IBaseModel ..... ۲۰
- شکل ۴: ModelState Enum ..... ۲۰
- شکل ۵: کلاس دیاگرام IBaseModel نسخه جنریک ..... ۲۱
- شکل ۶: کلاس دیاگرام BaseModel ..... ۲۲
- شکل ۷: کلاس دیاگرام Category ..... ۲۴
- شکل ۸: کلاس دیاگرام Food ..... ۲۵
- شکل ۹: کلاس دیاگرام Employee ..... ۲۶
- شکل ۱۰: Role Enum ..... ۲۶
- شکل ۱۱: کلاس دیاگرام Customer ..... ۲۷
- شکل ۱۲: کلاس دیاگرام Order ..... ۲۸
- شکل ۱۳: صفحه ورود ..... ۹۹
- شکل ۱۴: پنل مدیریت ..... ۱۰۰
- شکل ۱۵: منوی فایل ..... ۱۰۱
- شکل ۱۶: منوی زبان ..... ۱۰۱

- شکل ۱۷: مدیریت دسته بندی ..... ۱۰۳
- شکل ۱۸: پیغام مبنی بر پر بودن دسته بندی ..... ۱۰۴
- شکل ۱۹: غذای جدید ..... ۱۰۴
- شکل ۲۰: ویرایش غذا ..... ۱۰۵
- شکل ۲۱: غذای سفارش داده شده ..... ۱۰۵
- شکل ۲۲: تغییر دسته بندی ..... ۱۰۶
- شکل ۲۳: نمایش بر حسب دسته بندی ..... ۱۰۶
- شکل ۲۴: جستجو ..... ۱۰۶
- شکل ۲۵: کارمند جدید ..... ۱۰۷
- شکل ۲۶: نمودار فروش سالانه ..... ۱۰۸
- شکل ۲۷: فروش ماهانه ..... ۱۰۸
- شکل ۲۸: فروش ساعتی ..... ۱۰۹
- شکل ۲۹: فروش روزانه ..... ۱۰۹
- شکل ۳۰: پنل اصلی ..... ۱۱۰
- شکل ۳۱: مشتری جدید ..... ۱۱۱
- شکل ۳۲: سفارش جدید ..... ۱۱۲
- شکل ۳۳: گزارش سفارش ..... ۱۱۳
- شکل ۳۴: مشاهده سفارش ..... ۱۱۴

- شکل ۳۵: سفارش «بدون فیلتر» ..... ۱۱۵
- شکل ۳۶: سفارش های «بیرون» ..... ۱۱۵
- شکل ۳۷: سفارش های «معلق» ..... ۱۱۵
- شکل ۳۸: سفارش های «ارسال شده» به «بیرون» ..... ۱۱۶
- شکل ۳۹: اضافه کردن غذای جدید به سفارش ..... ۱۱۷
- شکل ۴۰: تغییر دیگر مشخصات سفارش ..... ۱۱۷
- شکل ۴۱: فایل های زبان ..... ۱۱۸
- شکل ۴۲: فایل زبان در Notepad ..... ۱۱۸
- شکل ۴۳: زبان جدید ..... ۱۱۸
- شکل ۴۴: صفحه ورود به سوئدی ..... ۱۱۸

### شرح سیستم

سیستم مدیریت رستوران به صاحبان رستوران این امکان را می‌دهد تا اعمالی مانند ثبت سفارش، ثبت مشتری و اداره رستوران را مکانیزه کنند در نتیجه باعث صرفه جویی در وقت و هزینه می‌شود.

### مشخصات فنی

این سیستم به صورت شی گرا پیاده‌سازی شده است که روند توسعه سیستم را آسان‌تر می‌کند. این سیستم با استفاده از زبان C# پیاده‌سازی شده است که زبانی بسیار قوی است و قابلیت برنامه نویسی شی گرا را دارا می‌باشد و پیاده‌سازی پروژه را آسان‌تر می‌نماید. همچنین قابلیت اجرا در سیستم عامل‌های مختلف از خانواده ویندوز و لینوکس (با استفاده از Mono) را دارا می‌باشد. این سیستم برای ذخیره اطلاعات از پایگاه داده SQL Server Compact Edition استفاده می‌کند که یک پایگاه داده رابطه‌ای و مستقل، بدون نیاز نصب و پیکربندی سرور است که برای چنین پروژه‌ای ایده آل است.

### سیستم مورد نیاز برای اجرای سیستم

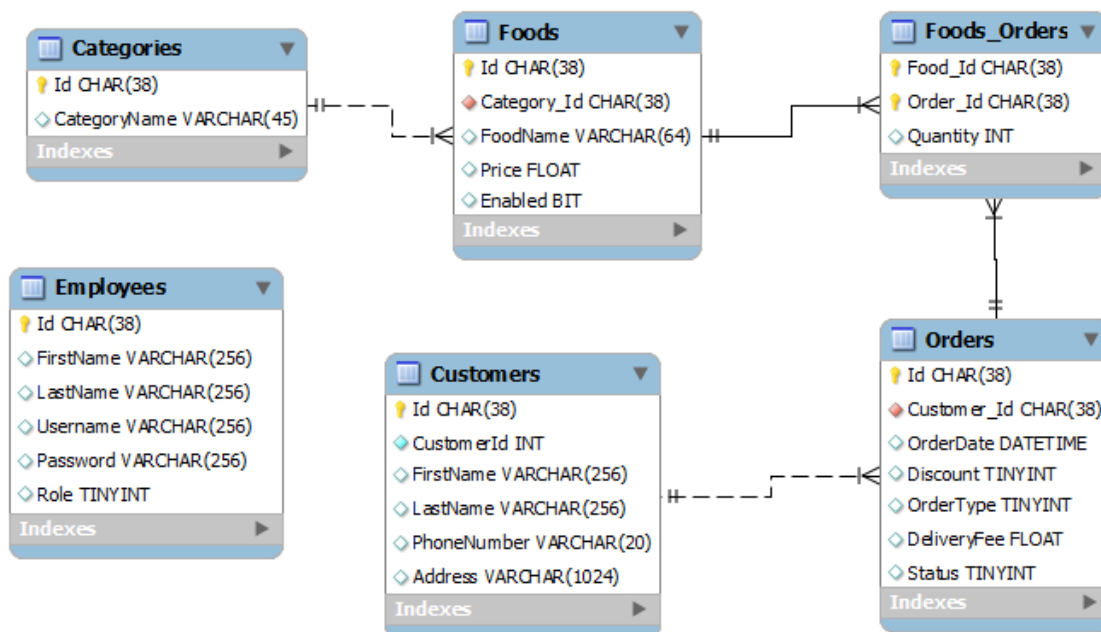
- Windows XP یا بالا تر
- .Net Framework 4.0
- SQL Server Compact Edition 3.5 SP2

### خلاصه امکانات نرم‌افزار

- تعریف، ویرایش، حذف، جستجو مشتری
- تعریف، ویرایش، جستجو حذف کارمند
- صدور نام کاربری و رمز عبور برای کارمندان جهت کار با سیستم
- تعیین، تغییر سطوح دسترسی برای کارمندان

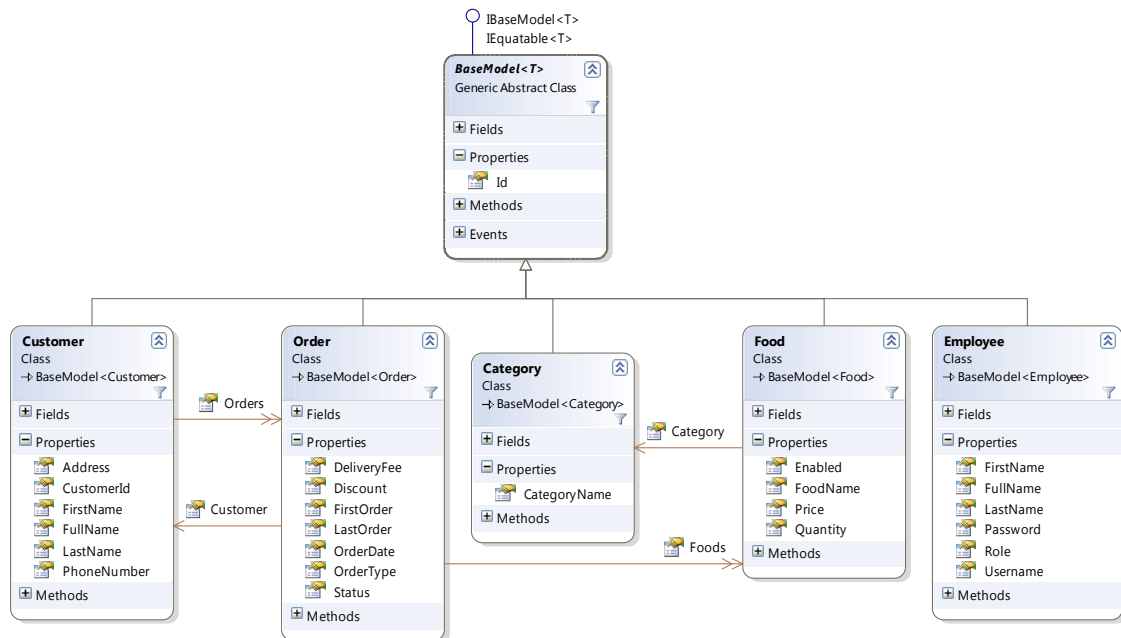
- تعریف، ویرایش، جستجو حذف غذا
- تعریف، ویرایش، حذف، تغییر دسته بندی برای غذا
- تعریف، ویرایش، حذف، فیلتر کردن سفارش
- امکان آمار گیری فروش سالانه، ماهانه، و روزانه
- امکان صدور فاکتور
- امکان ترجمه نرم افزار

### موجودیت ها



شکل ۱: دیاگرام EER سیستم مدیریت رستوران

در این سیستم ۵ موجودیت وجود دارد.



شکل ۲: کلاس دیاگرام سیستم مدیریت رستوران

### موجودیت و جدول «مشتري»

موجودیت «مشتري» یا «Customer» نمایانگر یک مشتري در این سیستم می‌باشد و یک مشتري کسی است که به درخواست او سفارش در سیستم به نام او ثبت می‌شود. همچنین اطلاعات هر مشتري در جدول «Customers» ثبت می‌شود و دارای فیلدهای زیر می‌باشد.

- **Id**: مقدار منحصر که برای انجام اعمال مختلف به موجودیت نسبت داده می‌شود.

- **CustomerId**: شماره اشتراک مشتري

- **FirstName**: نام مشتري

- **LastName**: نام خانوادگی مشتري

- **PhoneNumber**: شماره تلفن مشتري

- **Address**: آدرسی مشتري

در سیستم این موجودیت با کلاس «Customer» مشخص می‌شود که دارای فیلد هایی مشابه و همانم در جدول می باشد.

### موجودیت و جدول «دسته بندی»

موجودیت «دسته بندی» یا «Category» دسته بندی غذا ها در سیستم را مشخص می‌کند و لازم است که هر غذا حتماً به یک دسته بندی معتبر و تعریف شده تعلق داشته باشد؛ اطلاعات دسته بندی ها در جدول «Categories» ذخیره می شوند. و دارای فیلد های زیر است.

- **Id**: مقدار منحصر که برای انجام اعمال مختلف به موجودیت نسبت داده می‌شود.

- **CategoryName**: نام دسته بندی

در سیستم این موجودیت با کلاس «Category» مشخص می‌شود که دارای فیلد هایی مشابه و همانم در جدول می باشد.

### موجودیت و جدول «غذا»

موجودیت «غذا» یا «Food» نماینگر یک غذا در سیستم می‌باشند و یکی از موجودیت های اصلی سیستم محسوب می‌شود؛ همانطور که در قبل بیان شهد هر «غذا» لازم است که متعلق به یک «دسته بندی» باشد. اطلاعاتی این موجودیت در جدول «Foods» ذخیره می شود. همچنین در ثبت سفارش ها «Id» این موجودیت در جدول «Foods\_Orders» ذخیره می شود.

صفات این موجودیت به شرح زیر است

- **Id**: مقدار منحصر که برای انجام اعمال مختلف به موجودیت نسبت داده می‌شود.

- **Category\_Id**: فیلدکلید جدول «Categories» که برای مشخص کردن دسته بندی «غذا» به کار می رود. این فیلد در پیاده‌سازی به نام «Category» به عنوان یکی از فیلد های کلاس آورده می‌شود که به یک «دسته بندی» اشاره می کند.

- **FoodName**: نام غذا



- **Price:** قیمت غذا

- **Enabled:** مقدار این مشخص می‌کنید که آیا «غذا» فعال است یا خیر غذاهای فعال در هنگام ثبت سفارش آورده می‌شوند. ولی غذا های غیرفعال مخفی می‌شوند.

در سیستم این موجودیت با کلاس «Food» مشخص می‌شود که دارای فیلد هایی مشابه و همنام در جدول می باشد. به استثنای «Category\_Id» که با نام «Category» آورده می‌شود و از نوع «Category» است.

### موجودیت و جدول «کارمند»

موجودیت «کارمند» یا «Employee» نمایانگر یک «کارمند» در سیستم است؛ برای اینکه کاربر بتواند با سیستم کار کند حتماً باید جزو یکی از دسته بندی های کارمندان تعریف شده باشد تا بتواند وارد سیستم شده و با آن کار کند؛ کارمندان به ۳ دسته تقسیم می‌شوند.

- **«مدیرکل» یا «SuperAdmin»:** اولین کارمند سیستم می‌باشد و فقط یک کامند «مدیرکل» می‌تواند در سیستم وجود داشته باشد و اطلاعاتش قابل تغییر نمی‌باشد. مدیرکل به تمام قسمت‌های سیستم دسترسی دارد.

- **«مدیر» یا «Manager»:** مانند «مدیر کل» است با این تفاوت که این نوع کارمند قابل تعریف توسط کاربر می‌باشد، مدیران هم به پنل مدیریت و هم پنل اصلی که مخصوص ثبت سفارش‌ها است دسترسی دارند.

- **«صندوقدار» یا «Cashier»:** کارمندان صندوقدار فقط به پنل اصلی دسترسی دارند.

اطلاعات کارمندان در جدولی به نام «Employees» ذخیره می‌شود که فیلد های آن به شرح زیر است.

- **Id:** مقدار منحصر که برای انجام اعمال مختلف به موجودیت نسبت داده می‌شود.

- **FirstName:** نام «کارمند»

- **LastName:** نام خانوادگی «کارمند»

- **Username:** نام کاربری
  - **Password:** رمز عبور
  - **Role:** سطح دسترسی کارمند را تعیین می‌کند که با توجه به نوع کارمند فرق می‌کند؛  
«مدیرکل» عدد ۳ است، «مدیر» عدد ۲، و «صندوقدار» عدد ۳.
- در سیستم این موجودیت با کلاس «Employee» مشخص می‌شود و فیلدهای مشابه با فیلدهای نام برده شده دارد.

### موجودیت و جدول «سفارش»

- موجودیت «سفارش» یا «Order» نمایانگر یک سفارش در سیستم می‌باشد. سفارش‌ها یکی از اجزای اصلی سیستم هستند و یکی از پیچیده‌ترین موجودیت‌های سیستم محسوب می‌شود. هم سفارش شامل چندین «غذا» به تعدادی معین می‌باشد.
- اطلاعات این موجودیت در جدول «Order» ذخیره می‌شوند، سفارش‌ها به دو نوع تقسیم می‌شوند.
- «بیرون»: سفارشات است که به بیرون ارسال می‌شوند. و با شماره «۰» ذخیره می‌شود.
  - «سالن»: سفارشات است که درون سالن سفارش داده می‌شوند و با شماره «۱» ذخیره می‌شوند.
- هر سفارش ۴ نوع وضعیت مختلف دارد.
- «نامشخص»: وضعیت سفارش هنوز مشخص نیست و با شماره «» ذخیره می‌شود.
  - «معلق»: وضعیت سفارش معلق است و با شماره «۱» ذخیره می‌شود.
  - «ارسال شده»: سفارش‌هایی که ارسال شده‌اند و با شماره «۲» ذخیره می‌شوند.
  - «تحويل داده شده»: سفارش‌های تحويل داده شده و با شماره «۳» مشخص می‌شوند.
- سفارش دارای فیلدهای زیر است.
- **Id:** مقدار منحصر که برای انجام اعمال مختلف به موجودیت نسبت داده می‌شود.

- **Customer\_Id**: فیلد کلید موجودیت «مشتري»

- **OrderDate**: تاريخ ثبت سفارش

- **OrderType**: نوع سفارش

- **Status**: وضعیت سفارش

- **DeliveryFee**: هزینه پیک

- **Discount**: تخفیف

این موجودیت در سیستم با کلاس «Order» مشخص می‌شود تمام فیلدها نامی مشابه با نام آنها در جدول دارند به استثناء «Customer\_Id» که بجای آن از نام «Customer» استفاده می‌شود که خود از همان نوع «Customer» است.

جدول Foods\_Orders

این جدول برای ارتباط میان دو موجودیت «غذا» و «سفارش» استفاده می‌می‌شود. و دارای فیلدهای زیر است.

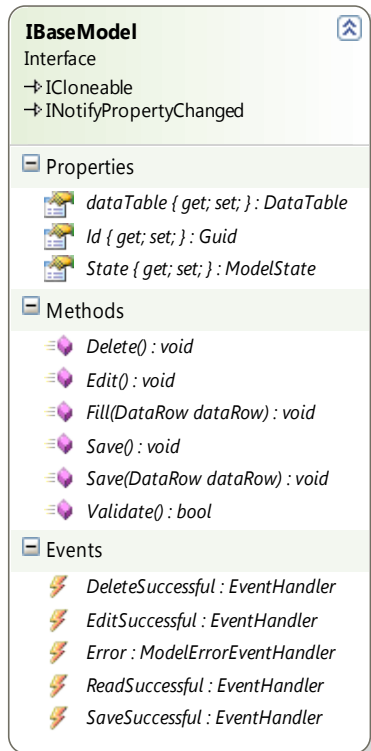
- **Food\_Id**: مقدار کلید موجودیت «غذا»

- **Order\_Id**: مقدار کلید موجودیت «سفارش»

- **Quantity**: تعداد غذا

این موجودیت هیچ کلاسی در سیستم ندارد.

## اینترفیس IBaseModel



این اینترفیس قالبی کلی برای تمام کلاس‌های در سیستم مشخص می‌کند. به این معنا است که تمام کلاس‌های موجود در سیستم متد هایی با همین نام و کارکردی مشابه دارا می‌باشند.

این کلاس دو نسخه دارد یک نسخه غیر جنریک و جنریک؛ که نسخه جنریک آن نسخه معمولی آن را به ارث می‌برد.

## پراپرتی های نسخه معمولی

- پراپرتی **dataTable**: به DataTable هر یک از

کلاس‌های فرزند اشاره می‌کند.

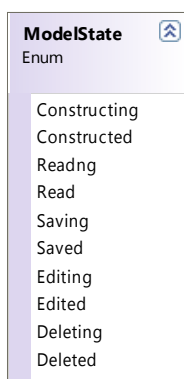
- پراپرتی **Id**: مقدار کلید هر موجودیت

- پراپرتی **State**: وضعیت هر موجودیت که شرح زیر

است.

شکل ۳: کلاس دیاگرام

IBaseModel



شکل ۴:

ModelState  
Enum

۱. **Constructing**: در حال ساخته شدن است.

۲. **Constructed**: ساخته شده ( در متد سازنده)

۳. **Reading**: در خواندن

۴.

۵. **Read**: خوانده شده ( بعد از موفقیت آمیز بودن عمل خواندن)

۶. **Saving**: در حال ذخیره سازی

۷. **Saved**: ذخیره شده ( بعد از موفقیت آمیز بودن عمل ذخیره)

۸. **Editing**: در حال ویرایش

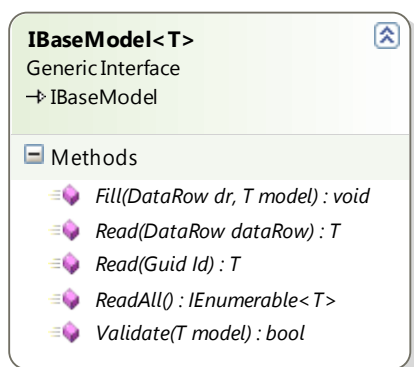
۹. **Edited:** ویرایش شده ( بعد از موفقیت آمیز بودن عمل ویرایش)

۱۰. **Deleting:** در حال حذف

۱۱. **Deleted:** حذف شده ( بعد از موفقیت آمیز بودن عمل حذف)

### متدهای نسخه ساده

- **متد Delete:** موجودیت جاری را پاک می کند.
- **متد Edit:** موجودیت را ویرایش می کند.
- **متد Fill:** کار این متد این است که ورودی `DataRow` می گیرد و مقادیر فیلدها را درون آن می ریزد.
- **متد Save:** موجودیت را ذخیره می کند.
- **متد Save با ورودی DataRow:** این متد ابتدا `DataRow` ورودی را می خواند و پس آن را ذخیره می کند.



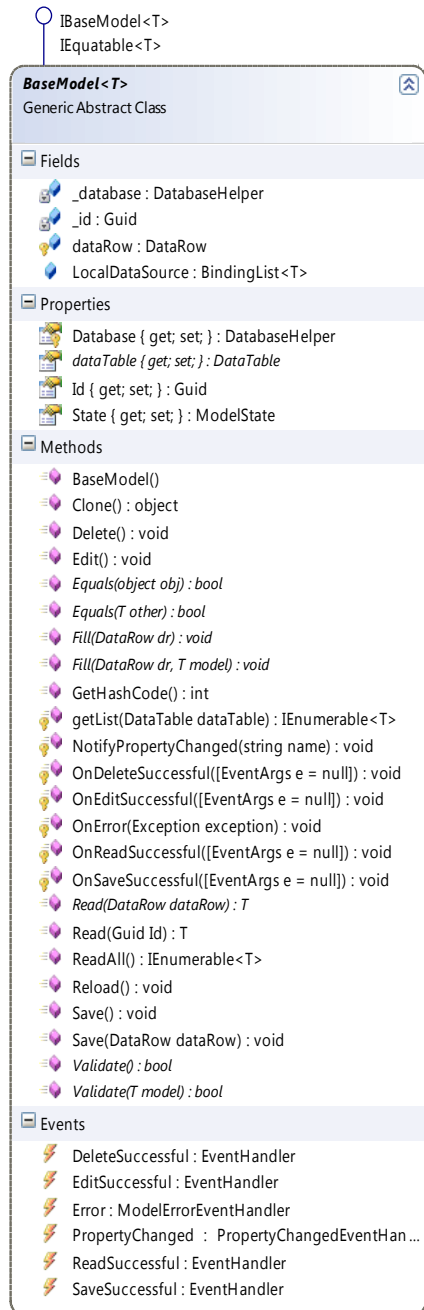
- **متد Validate:** مقادیر فیلدها را برای صحت آنها بررسی می کند.

### متدهای نسخه جنریک

- **متد Fill:** کارکرد آن مانند متد `Fill` در نسخه معمولی است با این تفاوت که دو ورودی می گیرد و اطلاعات ورودی دوم را در `DataRow` می ریزد.

شکل ۵: کلاس دیاگرام `IBaseModel` نسخه جنریک

- **متد Read با ورودی Guid:** این متد موجودیتی که فیلد کلید آن برابر با مقدار ورودی اش باشد را بر می گرداند.



- **متد Read با ورودی DataRow:** این متد یک DataRow را می خواند و مقادیر آن را به فیلد ها نسبت می دهد.
- **متد ReadAll:** تمام مقادیر موجود در جدول برای آن موجودیت خاص را می خواند.
- **متد Validate:** از صحت اطلاعات موجود در ورودی داده شده اطمینان حاصل می کند.

## رویداد ها

- **DeleteSuccessful:** زمانی رخ می دهد که عمل حذف با موفقیت انجام شود.
- **ReadSuccessful:** زمانی رخ می دهد که عمل خواندن با موفقیت انجام شود.
- **EditSuccessful:** زمانی رخ می دهد که عمل ویرایش با موفقیت انجام شود.
- **SaveSuccessful:** زمانی رخ می دهد که عمل ذخیره سازی با موفقیت انجام شود.
- **Error:** زمانی رخ می دهد که خطایی در برنامه رخ دهد.

شکل ۶: کلاس دیاگرام BaseModel

## کلاس BaseModel

این کلاس والد تمام کلاس‌ها در سیستم است و اینترفیس IBaseModel را پیاده‌سازی می‌کند. اعضای این کلاس به شرح زیر است که فقط اعضای جدید شرح داده می‌شوند.

### پراپرتی‌ها

- **Database:** یک نمونه از کلاس کمکی DatabaseHelper می‌باشد که کار ارتباط به پایگاه داده را انجام می‌دهد.

### فیلدها

- **dataRow:** یک DataRow که مربوط به نمونه جاری از کلاس فرزند هست را نگه می‌دارد.
- **LocalDatasource:** یک لیست حاوی تمام نمونه‌ها از کلاس فرزند.

### متدها

- **Clone:** از اعضای اینترفیس ICloneable است و کار آن کپی گرفتن از نمونه جاری است.
- **Equals با ورودی object:** مساوی بودن دو نمونه را بررسی می‌کند.
- **Equals با ورودی T:** مساوی بودن دو نمونه را بررسی می‌کند و از اعضای اینترفیس IEquatable جنریک می‌باشد.
- **GetHashCode:** هشکد نمونه جاری را تولید می‌کند.
- **getList:** ورودی DataTable را تبدیل به یک IEnumerable جنریک می‌کند و بر می‌گرداند.
- **NotifyPropertyChanged:** برای پاسخ‌گویی به رویداد PropertyChanged استفاده می‌شود.
- **OnDeleteSuccessful:** برای پاسخ‌گویی به رویداد DeleteSuccessful استفاده می‌شود.
- **OnEditSuccessful:** برای پاسخ‌گویی به رویداد EdiSuccessful استفاده می‌شود.

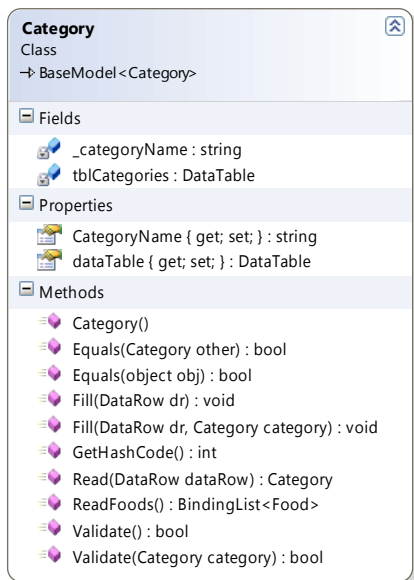
- **OnReadSuccessful**: برای پاسخ‌گویی به رویداد ReadSuccessful استفاده می‌شود.
- **OnSaveSuccessful**: برای پاسخ‌گویی به رویداد SaveSuccessful استفاده می‌شود.
- **OnError**: برای پاسخ‌گویی به رویداد OnError استفاده می‌شود.

### رویداد ها

- **PropertyChanged**: این رویداد مربوط به اینترفیس InotifyPropertyChanged است؛ زمانی رخ می‌دهد که مقدار یک پراپرتی عوض شود.

### کلاس Category

همانطور که قبلاً بیان شد این کلاس مربوط به موجودیت «دسته بندی» می‌باشد.



### فیلدها

- **tblCategories**: یک DataTable مخصوص کلاس Category می‌باشد، که داده‌های استخراج شده از پایگاه داده در آن ذخیره می‌شود.

### متدها

- **ReadFoods**: تمام غذا های موجود در این دسته بندی را می‌خواند و برمی‌گرداند.

شکل ۷: کلاس دیاگرام Category



## کلاس Food

این کلاس همانطور که بیان شده نمایانگر یک «غذا» در سیستم می باشد.

### فیلدها

- **tblFoods**: یک DataTable

مخصوص کلاس Food می باشد، که داده های استخراج شده از پایگاه داده در آن ذخیره می شود.

### پراپرتی ها

- **Category**: از نوع کلاس Category می باشد و دسته بندی غذا را مشخص می کند.
- **Quantity**: تعداد غذا در سفارش

### متدها

- **DecreaseQuantity**: کاهش

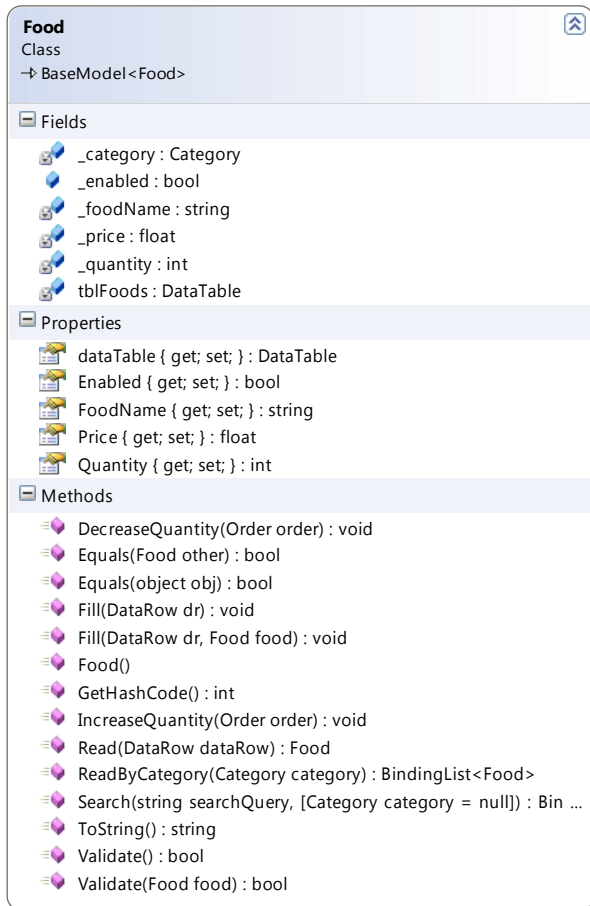
تعداد غذا

- **IncreaseQuantity**: افزایش تعداد غذا

- **ReadByCategory**: خواندن بر حسب «دسته بندی»

- **Search**: جستجو، اگر ورودی دوم نیز داده شود بر حسب «دسته بندی» جستجو را انجام می دهد در غیر این صورت جستجو بر روی تمام «غذا» ها انجام می شود.

- **ToString**: نام غذا را بر می گرداند



شکل ۸: کلاس دیاگرام Food

## کلاس Employee

این کلاس همانطور که بیان شده نمایانگر یک «کارمند» در سیستم می باشد.

### فیلدها

- **tblEmployees**: یک DataTable مخصوص

کلاس Employee می باشد، که داده های استخراج شده از پایگاه داده در آن ذخیره می شود.

### پراپرتی ها

- **Role**: سطح دسترسی کارمند را مشخص می کند که از نوع Role می باشد که یک Enum است.

۱. **Unauthorized**: کاربر غیرمجاز است و

مقدارش ۱- است.

۲. **Cashier**: کاربر «صندوق دار» است و

مقدارش ۱ است.

۳. **Manger**: کاربر «مدیر» است و مقدارش ۲ است.

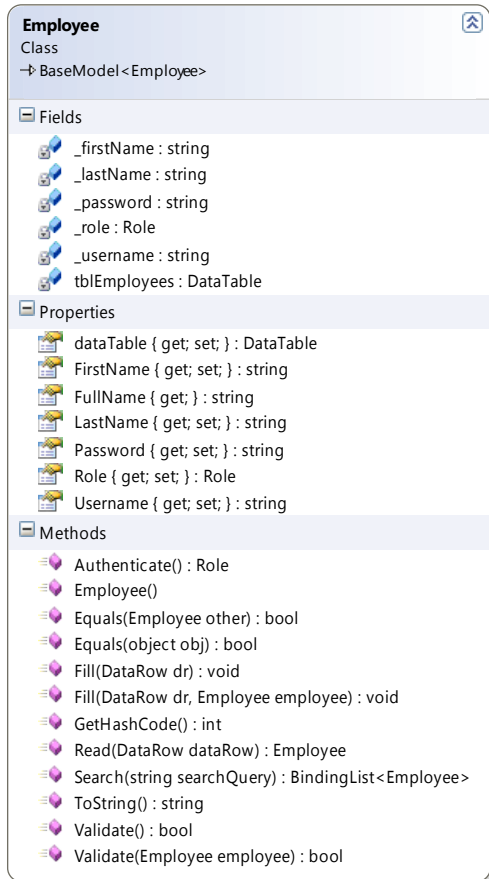
۴. **Default**: کاربر پیشفرض است و مقدارش ۰ است؛ کاربر پیشفرض

همان Cashier است.

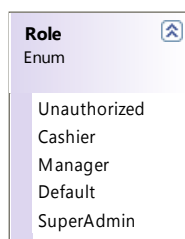
۵. **SuperAdmin**: کاربر «مدیرکل» است و مقدارش ۳

### متدها

- **Authenticate**: معتبر بودن «نام کاربری» و «رمز عبور» کارمند را بررسی می کند.



شکل ۹: کلاس دیاگرام Employee



شکل ۱۰: Role

Enum

- **Search:** در بین کارمندان جستجو می کند.

### کلاس Customer

این کلاس همانطور که بیان شده نمایانگر یک «مشتري» در سیستم می باشد.

### فیلدها

- **tblCustomers:** یک DataTable مخصوص کلاس Customer می باشد، که داده های استخراج شده از پایگاه داده در آن ذخیره می شود.

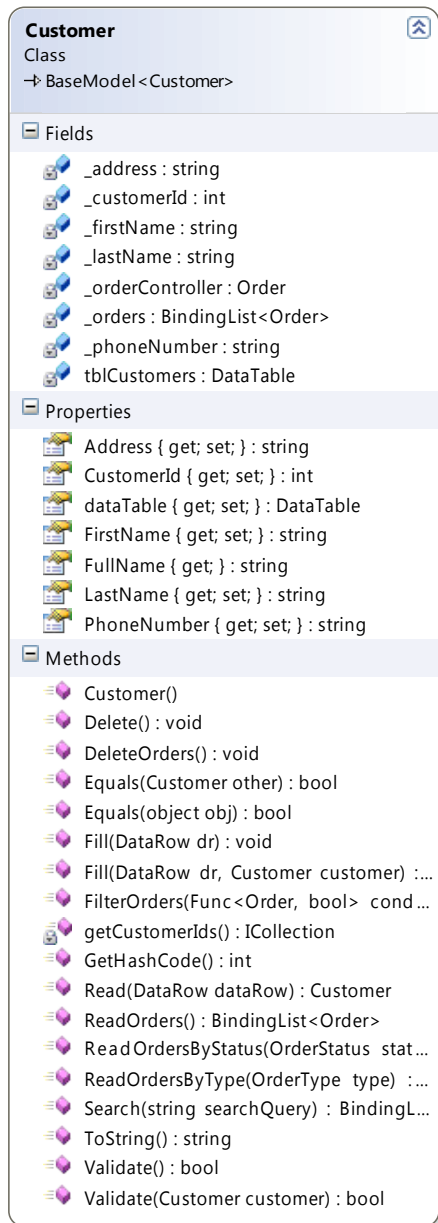
- **\_orderController:** یک نمونه از کلاس Order برای کار با سفارشات درون کلاس Customer.

### پراپرتی ها

- **FullName:** نام کامل را بر می گرداند.

### متدها

- **DeleteOrders:** تمام سفارش های مشتری جاری را پاک می کند.
- **FilterOrders:** خواندن سفارش های مشتری جاری با توجه به شرط داده شده.
- **getCustomerId:** لیستی از تمام شماره اشتراک ها بر می گرداند.
- **ReadOrders:** خواندن تمام سفارش های مربوط به این مشتری



شکل ۱۱: کلاس دیاگرام Customer

• **ReadOrdersByType**: سفارش ها را بر حسب نوعشان می خواند.

• **ReadOrdersByStatus**:

سفارش ها را بر حسب وضعیتشان می خواند.

• **Search**: در بین مشتریان

جستجو می کند.

• **ToString**: مقدار »

FullName را بر می گرداند.

### کلاس Order

این کلاس نمایانگر یک «سفارش» در سیستم می باشد؛ از آنجایی که هر سفارش دارای چندین «غذا» می باشد این کلاس مستقیماً با جدول

«Foods\_Orders» هم کار می کند. در

اینجا اعضا را با توجه به اینکه با چه

جدولی کار می کنند دسته بندی می کنیم.

### فیلدها

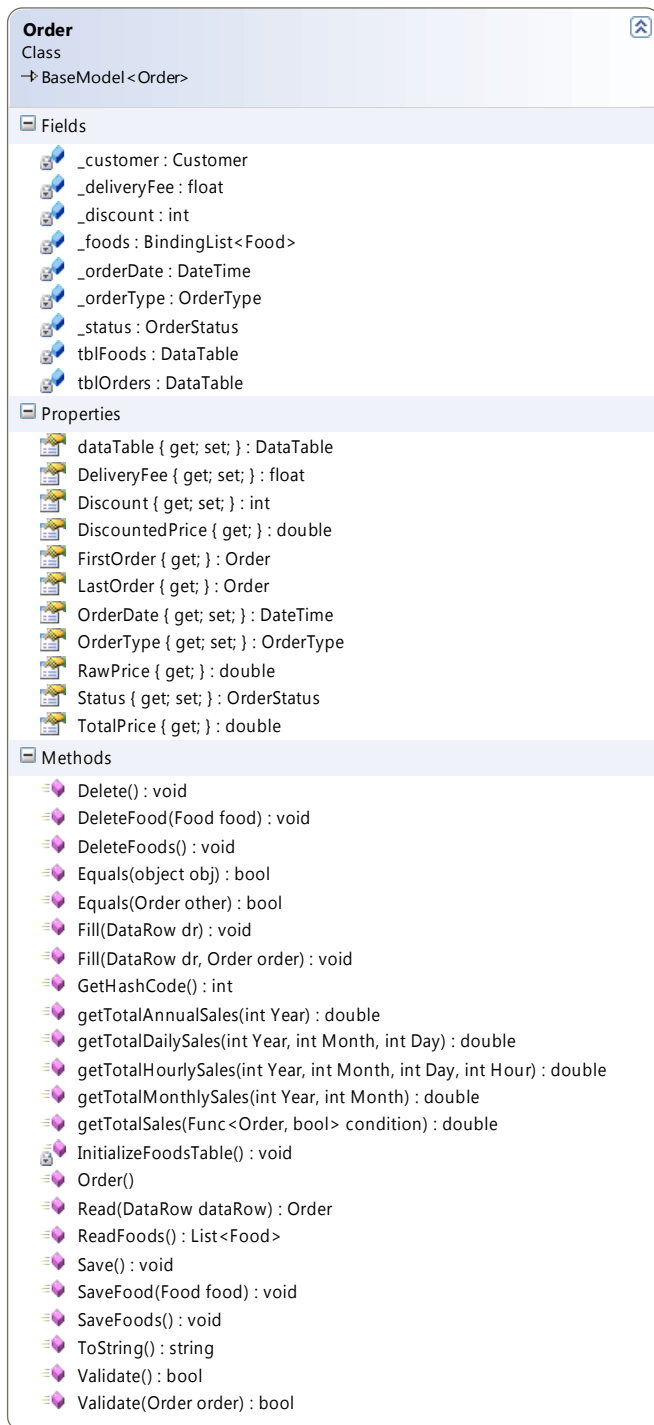
• **tblOrder**: یک DataTable

مخصوص کلاس Order

می باشد، که داده های استخراج

شده از پایگاه داده در آن

ذخیره می شود.



شکل ۱۲: کلاس دیاگرام Order

- **tblFoods**: یک DataTable که داده‌های آن متشکل از «غذا» های موجود در جدول «Foods\_Orders» می باشد. همچنین دارای تمام فیلد های جدول ذکر شده نیز هست.

#### پراپرتی ها

- **DiscountedPrice**: میزان تخفیف
- **FirstOrder**: اولین سفارش
- **LastOrder**: آخرین سفارش
- **RawPrice**: قیمت کل بدون تخفیف
- **TotalPrice**: قیمت کل با اعمال تخفیف

#### متد های مربوط به آمارگیری

- **getTotalSales**: این متد با توجه به ورودی جمع فروش را به ما می دهد.
- **getAnnualSales**: این متد فروش سالانه را محاسبه می کند.
- **getMonthlySales**: این متد فروش ماهیانه را در یک سال مشخص محاسبه می کند.
- **getDailySales**: این متد فروش روزانه را در یک ماه و سال مشخص محاسبه می کند.
- **getHourlySales**: این متد فروش ساعتی در یک روز مشخص را محاسبه می کند.

#### متد های مربوط به جدول Foods\_Orders

- **InitializeFoodsTable**: این متد DataTable مربوط به جدول Foods\_Orders را آماده می کند.
- **DeleteFoods**: حذف غذا های سفارش داده شده.
- **SaveFoods**: ذخیره غذا های سفارش داده شده.
- **ReadFoods**: خواندن تمام غذا های سفارش داده شده.

- **SaveFood**: اضافه کردن یک غذا به سفارش.

- **DeleteFood**: حذف یک غذا از سفارش.

## پیاده‌سازی

این سیستم به طور شی گرا پیاده‌سازی می‌شود. برای این کار ابتدا یک اینترفیس به نام «IBaseModel» ایجاد می‌کنیم؛ این اینترفیس قالبی کلی برای تمام کلاس‌های در سیستم مشخص می‌کند. این اینترفیس توسط کلاس «BaseModel» پیاده‌سازی می‌شود که پدر تمام کلاس‌ها در سیستم است و اعمالی مانند «حذف»، «اضافه»، و «ذخیره» را انجام می‌دهد و از کد‌های اضافه می‌کاهد. در بخش پیاده‌سازی، پیاده‌سازی اجزای مهم سیستم بیان می‌شود. همچنین در این سیستم فقط برای اعمال اصلی از کد SQL استفاده شده و برای باقی تهیه گزارش‌ها از عبارات LINQ استفاده می‌شود که عبارات LINQ شباهت زیادی به کد‌های SQL دارند.

## پیاده‌سازی کلاس‌های کمکی

در قسمت پیاده‌سازی کلاس‌های کمکی فقط پیاده‌سازی اعضای از آن کلاس‌های گفته می‌شود که در طول سیستم استفاده شده‌اند.

## کلاس Common

### متد SHA1

این متد رشته ورودی را توسط الگوریتم «SHA1» هش می‌کند.

```
return  
Convert.ToBase64String(MD5.Create().ComputeHash(Encoding.UTF8.GetBytes(str)));
```

### متد InitializeDatabase

این متد پایگاه داده و جدول‌های مورد نیاز را می‌سازد.

ابتدا مقدار «ConnectionString» را از فایل تنظیمات نرم‌افزار می‌خوانیم.

```

        ConnectionString =
ConfigurationManager.ConnectionStrings["RestaurantDB"].ConnectionString;

        if (!File.Exists("RestaurantDB.sdf"))
        {

```

در صورتی که پایگاه داده ساخته نشده بود آن را می سازیم.

```

        SqlCeEngine sqlCeEngine = new SqlCeEngine(ConnectionString);
        sqlCeEngine.CreateDatabase();

        Dictionary<string, string> Tables = new Dictionary<string, string>();

```

به دیکشنری «Tables» کدهای SQL برای ساخت هر جدول را اضافه می کنیم و در حلقه زیر آن را اجرا می کنیم.

```

        Database.Connect();

        foreach (KeyValuePair<string, string> table in Tables)
        {
            if (!Database.TableExists(table.Key))
                Database.ExecuteNonQuery(table.Value, false);
        }

        Database.Disconnect();
    }

```

کدهای SQL جداول

جدول «Categories»

```

CREATE TABLE Categories (Id NCHAR(38) NOT NULL DEFAULT newid(),
                           CategoryName NVARCHAR(45) NOT NULL ,
                           PRIMARY KEY (Id) );

```

**جدول «Foods»**

```
CREATE TABLE Foods (Id NCHAR(38) NOT NULL DEFAULT newid(),  
  
    Category_Id NCHAR(38) NOT NULL ,  
  
    FoodName NVARCHAR(64) NOT NULL ,  
  
    Price FLOAT NULL DEFAULT 0 ,  
  
    Enabled BIT DEFAULT 1,  
  
    PRIMARY KEY (Id) ,  
  
    CONSTRAINT fk_Foods_Categories1  
        FOREIGN KEY (Category_Id )  
        REFERENCES Categories (Id )  
  
    ON DELETE NO ACTION  
  
    ON UPDATE NO ACTION);
```

**جدول «Customers»**

```
CREATE TABLE Customers (Id NCHAR(38) NOT NULL DEFAULT newid(),  
  
    CustomerId INT NOT NULL ,  
  
    FirstName NVARCHAR(256) NOT NULL ,  
  
    LastName NVARCHAR(256) NOT NULL ,  
  
    PhoneNumber NVARCHAR(20) NULL ,  
  
    Address NVARCHAR(1024) NULL ,  
  
    PRIMARY KEY (Id),  
  
    CONSTRAINT UQ_Customers_CustomerId UNIQUE (CustomerId));
```

**جدول «Orders»**

```
CREATE TABLE Orders (Id NCHAR(38) NOT NULL DEFAULT newid(),  
  
    Customer_Id NCHAR(38) NOT NULL ,
```



```

        OrderDate DATETIME NULL ,
        Discount TINYINT NULL ,
        OrderType TINYINT NOT NULL ,
        DeliveryFee FLOAT NOT NULL DEFAULT 0,
        Status TINYINT NULL ,
        PRIMARY KEY (Id) ,
        CONSTRAINT fk_Orders_Customers1
            FOREIGN KEY (Customer_Id )
            REFERENCES Customers (Id )
            ON DELETE NO ACTION
            ON UPDATE NO ACTION);

```

### جدول «Foods\_Orders»

```

CREATE TABLE Foods_Orders (Food_Id NCHAR(38) NOT NULL ,
        Order_Id NCHAR(38) NOT NULL ,
        Quantity INT NULL ,
        PRIMARY KEY (Food_Id, Order_Id) ,
        CONSTRAINT fk_Foods_Orders_Foods
            FOREIGN KEY (Food_Id )
            REFERENCES Foods (Id )
            ON DELETE NO ACTION
            ON UPDATE NO ACTION,
        CONSTRAINT fk_Foods_Orders_Orders1
            FOREIGN KEY (Order_Id )
            REFERENCES Orders (Id )
            ON DELETE NO ACTION
            ON UPDATE NO ACTION);

```

## جدول «Employees»

```
CREATE TABLE Employees (Id NCHAR(38) NOT NULL DEFAULT newid(),
    FirstName NVARCHAR(256) NOT NULL ,
    LastName NVARCHAR(256) NOT NULL ,
    Username NVARCHAR(256) NOT NULL ,
    Password NVARCHAR(256) NOT NULL ,
    Role TINYINT NOT NULL ,
    PRIMARY KEY (Id) ,
    CONSTRAINT UQ_Employees_Username UNIQUE (Username));
```

## متد HandelSqlCeException

این متد به Exception های نوع SqlCeException رسیدگی می کند.

```
public static void HandleSqlCeException(SqlCeException ex){
    string FieldName = null;
    switch (ex.NativeError){
```

با استفاده از کد زیر نام فیلدی که آن خطا در آن رخ داد را می گیریم.

```
case 25005: //null value
    FieldName = ex.Errors[0].ErrorParameters[0].ToString();
```

سپس پیغامی مبنی بر خالی بودن این فیلد نمایش می دهیم.

```
MessageBox.Show(string.Format(Common.Words["FieldIsEmpty"],
Common.Words[FieldName]), Common.Words["EmptyField_Title"],
MessageBoxButtons.OK, MessageBoxIcon.Error);

break;

case 25016:
```

در اینجا برای دست آوردن نام فیلد از Regular Expression زیر استفاده می کنیم.

```

        FieldName = Regex.Match(ex.Message, string.Format(@"(?
<=Constraint name = UQ_{0}_)(\w+)",
ex.Errors[0].ErrorParameters[0].ToString())).Value;

```

پیغامی مبنی بر تکرار بودن مقدار این فیلد نمایش می دهیم.

```

        MessageBox.Show(string.Format(Common.Words["DuplicateValue"]
, Common.Words[FieldName]), Common.Words["DuplicateValue_Title"],
MessageBoxButtons.OK, MessageBoxIcon.Error);

        break;

```

در اینجا پیغامی مبنی بر تعلق «غذا» به سفارش چاپ می کنیم.

```

        case 25025: MessageBox.Show(Common.Words["FoodBelongsToOrder"],
Common.Words["Error"], MessageBoxButtons.OK, MessageBoxIcon.Error);

        break;

```

در نهایت اگر خطای رخ داده هیچ کدام از کدهای بالا نبود پیغامی مبنی بر نامشخص بودن خطا با شماره خطا نمایش می دهیم

```

        default:
        MessageBox.Show(string.Format(Common.Words["UnknownError"],
ex.NativeError.ToString()), Common.Words["UnknownError_Title"],
MessageBoxButtons.OK, MessageBoxIcon.Error); break;

    }

}

```

## متد Initialize

این متد زمانی که سیستم اجرا می‌شود اجرا می‌شود.

```
public static void Initialize(){  
    InitializeDatabase();  
    Common.Words = new Dictionary<string, string>();
```

در این قسمت فایل زبان را می‌خوانیم.

```
if (configFileParser == null){  
  
if (ConfigurationManager.AppSettings.AllKeys.Contains("DefaultLanguage")){  
    CurrentLanguage = new  
CultureInfo(ConfigurationManager.AppSettings["DefaultLanguage"]);  
    configFileParser = new  
ConfigFileParser(string.Format(@"Languages\{0}.ini", CurrentLanguage.Name));  
    }  
  
}
```

در این قسمت مقادیر موجود در فایل زبان را درون یک دیکشنری به نام «Words» می‌ریزیم.

```
Words = configFileParser.getDictionary();  
}
```

**متد ChangeLanguage**

این متد یک فایل زبان جدید را می‌خواند و دیکشنری «Words» را دوباره پر می‌کند.

```
public static void ChangeLanguage(string languageName) {
    configFileParser = new ConfigFileParser(string.Format(@"Languages\
{0}.ini", languageName));

    Words = configFileParser.getDictionary();
}
```

**متد ToggleRightToLeft**

```
public static void ToggleRightToLeft(Control control, Action leftToRight = null,
Action rightToLeft = null){
    if (Common.Words["RightToLeft"] == "1"){
```

در صورتی که مقدار «RightToLeft» در زبان «۱» باشد. رابط گرافیکی را راست به چپ می‌کند.

این کار با استفاده از پراپرتی «RightToLeft» کنترل ورودی انجام می‌شود.

```
control.RightToLeft = RightToLeft.Yes;
```

در صورت لزوم می‌توان یک تابع بدون هیچ گونه خروجی و ورودی را به ورودی «rightToLeft» و «leftToRight» بدهیم تا بتوانیم کد دلخواه را در این قسمت‌ها اجرا کنیم.

```
if (rightToLeft != null)
    rightToLeft();
}
else if (Common.Words["RightToLeft"] == "0")
{
```

در صورتی که مقدار «RightToLeft» در زبان «۱» باشد. رابط گرافیکی را چپ به راست می‌کند.

```
control.RightToLeft = RightToLeft.No;

if (leftToRight != null)
    leftToRight();
```

```
    }
}
```

### متد ShowDeletePrompt

پیغامی برای تأیید عمل حذف نمایش می‌دهد و در صورت تأیید عمل حذف را انجام می‌دهد.

```
public static void ShowDeletePrompt(DataGridView dataGrid, Action DeleteCanceled
= null){
    foreach (DataGridViewRow row in (dataGrid as DataGridView).SelectedRows){
        if (row.Index != dataGrid.NewRowIndex && row.Index != -1){
            DialogResult dialogResult =
                MessageBox.Show(string.Format(Common.Words["DeletePrompt"], (row.DataBoundItem
                as IBaseModel).ToString()), Common.Words["DeletePrompt_Title"],
                MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
            if (dialogResult == DialogResult.Yes){
                try{
```

در اینجا نمونه موجود در سطر DataGridView را به IBaseModel تبدیل کرده و متد Delete آن را فراخوانی می‌کند.

```
                (row.DataBoundItem as IBaseModel).Delete();
            }
            catch (SqlCeException ex){
```

اگر خطایی رخ دهد آن را به متد «HandleSqlCeException» می‌دهد.

```
                HandleSqlCeException(ex);
            }
        } else {
            if (DeleteCanceled != null)
```

کد دلخواه در صورت لغو شدن عمل حذف را اجرا می‌کند.

```
                DeleteCanceled();
            continue;
```

```

    }
}
}
}

```

### متد ToggleControlsDisableEnable

با توجه به اینکه سطر انتخاب چه سطر است کدی را اجرا می کند که برای فعال و غیرفعال کردن کنترل ها کاربرد دارد.

```

public static void ToggleControlsDisableEnable(DataGridView dataGrid,
Action enable, Action disable){
    foreach (DataGridViewRow row in dataGrid.SelectedRows)
        if (row.Index != dataGrid.NewRowIndex && row.Index != -1)
            enable();
        else
            disable();
}

```

### متد InitializeErrorHandles

به رویداد «Error» در کلاس ها رسیدگی می کند.

```

public static void InitailzeErrorHandlers(ref IBindingList list) {
    foreach (IBaseModel model in list) {
        model.Error += new ModelErrorEventHandler(delegate(object
sender, ModelErrorEventArgs e){

```

پیغام مربوط به آن Exception را نمایش می دهد.

```

Common.ShowErrorMessage(e.Exception);
});

```

```

    }
}

```

### متد InitializeList

لیست های داده شده را مقدار دهی می کند.

```

public static void InitializeList<T>(ref BindingList<T> list, ref T
Controller) where T : IBaseModel<T>, new(){

    if (list == null)

        list = new BindingList<T>();

    if (Controller == null)

        Controller = new T();
}

```

لیست های داده شده را مقدار دهی می کند.

```

Controller.Error += new ModelErrorEventHandler(delegate(object
sender, ModelEventArgs e){

    Common.ShowErrorMessage(e.Exception);

});

foreach (T model in Controller.ReadAll()){

```

به رویداد «Error» کلاس رسیدگی می کند و سپس آن ره به لیست اضافه می کند.

```

model.Error += new ModelErrorEventHandler(delegate(object
sender, ModelEventArgs e){

    Common.ShowErrorMessage(e.Exception);

});

list.Add(model);

}

}

```



### متد IsActiveTab

برسی می‌کند که آیا تب مورد نظر انتخاب شده است یا خیر

```
public static bool IsActiveTab(TabControl tab, TabPage tabPage){  
    return tab.SelectedTab.Equals(tabPage);  
}
```

### متد ShowErrorMessage

اگر متن خطا قبلاً تعریف شده بود به دنبال آن می‌گردد تا آنرا نشان بدهد.

```
public static void ShowErrorMessage(string errorMessage){  
    if (Common.Words.ContainsKey(errorMessage))  
        MessageBox.Show(Common.Words[errorMessage],  
Common.Words["Invalid Input"], MessageBoxButtons.OK, MessageBoxIcon.Error);  
    else
```

در غیر این صورت متن اصلی را نمایش می‌دهد.

```
        MessageBox.Show(errorMessage, Common.Words["Invalid Input"],  
MessageBoxButtons.OK, MessageBoxIcon.Error);  
}
```

این متد یک نسخه با ورودی «Exception» دارد که پیغام آن را گرفته و به ورودی همین متد می‌دهد.

```
public static void ShowErrorMessage(Exception ex) {  
    ShowErrorMessage(ex.Message);  
}
```

### متد AddDataGridViewColumn

این متد به DataGridView داده شده ستون مورد نظر را اضافه می کند.

```
public static void AddDataGridViewColumn(DataGridView dataGrid, string
fieldName)
{
    DataGridViewTextBoxColumn column = new DataGridViewTextBoxColumn();
    column.HeaderText = Common.Words[fieldName];
    column.Name = fieldName;
    column.DataPropertyName = fieldName;
    dataGrid.Columns.Add(column);
}
```

### متد CommonDataGridCellBeginEdit

این متد در زمان شروع به ویرایش یک سلول یک کپی از نمونه موجود در سطر در صفت «Tag» آن سطر ایجاد می کند.

```
public static void CommonDataGridCellBeginEdit(DataGridView dataGrid,
DataGridViewCellCancelEventArgs e){
    if (dataGrid.Rows[e.RowIndex].Tag == null &&
dataGrid.Rows[e.RowIndex].DataBoundItem != null){
        dataGrid.Rows[e.RowIndex].Tag =
(dataGrid.Rows[e.RowIndex].DataBoundItem as IBaseModel).Clone();
    }
}
```

**متد CommonDataGridSubmitEdit**

این متد تمام نمونه‌هایی که State آنها برابر با «Editing» را متد Edit آنها را فراخوانی می‌کند.

```
public static void CommonDataGridSubmitEdit(IEnumerable<IBaseModel> list){
    try{
        foreach (IBaseModel model in list.Where(
            model => model.State == ModelState.Editing)){
            model.Edit();
        }
    }
    catch (SqlCeException ex){
        Common.HandleSqlCeException(ex);
    }
}
```

**متد CommonDiscardChanges**

این متد تمام تغییرات را لغو می‌کند.

```
public static void CommonDiscardChanges<EntityType>(IList<EntityType> list)
where EntityType : IBaseModel
{
    }
```

ابتدا مقادیر موجود در لیست ورودی را درون یک لیست موقت می‌ریزیم.

```
List<EntityType> tempModels = new List<EntityType>();
foreach (EntityType model in list)
    tempModels.Add(model);
```

سپس مقدار State هر نمونه را بررسی می‌کنیم

آنهایی که مقدار State آنها برابر با «Editing»، «Constructed»، و یا «Constructing» می‌باشد را از لیست حذف می‌کنیم.

```
foreach (EntityType model in tempModels.Where(
(model) => model.State == ModelState.Constructed ||
    model.State == ModelState.Editing ||
    model.State == ModelState.Constructing)){
    list.Remove(model);
}
}
```

### متد PopulateLanguageMenu

این متد منوی «زبان» را با زبان‌های موجود در سیستم پر می‌کند.

```
public static void PopulateLanguageMenu(ToolStripMenuItem languageMenu,
Action InitializeLanguage){
    if (Languages == null)
        Languages = new Dictionary<string, CultureInfo>();
    foreach (string languageFile in Directory.GetFiles("Languages")){
```

در اینجا نام فایل زبان را می‌گیریم و نام «Culture» را از آن استخراج می‌کنیم. و به لیست اضافه می‌کنیم.

```
        string lang =
languageFile.Remove(languageFile.LastIndexOf('.')).Substring(languageFile.IndexOf(
f("\\") + 1);

        if (!Languages.ContainsKey(lang))
            Languages.Add(lang, new CultureInfo(lang));
```

در اینجا زبان‌ها را به منو اضافه می‌کنیم.

```
ToolStripMenuItem subMnuLanguage = new
ToolStripMenuItem(Languages[lang].NativeName);

subMnuLanguage.Tag = Languages[lang];

subMnuLanguage.BackColor = System.Drawing.Color.FromArgb(255,
192, 128);
```

در اینجا رویداد «Click» شده هر زیر منو را پیاده‌سازی می‌کنیم.

```
subMnuLanguage.Click += delegate(object sender, EventArgs e)
{
```

در اینجا متد «ChangeLanguage» را فراخوانی می‌کنیم. و همچنین زبان جاری را به زبان انتخاب شده تغییر می‌دهیم.

```
Common.ChangeLanguage(((sender as ToolStripMenuItem).Tag as CultureInfo).Name);
CurrentLanguage = (sender as ToolStripMenuItem).Tag as CultureInfo;
```

متد های «InitializeLanguage» در فرم‌های مختلف را صدا می‌زنیم. تا متن کنترل‌ها به زبان انتخاب شده تغییر کند.

```
MainForm.InitializeLanguage();

AdministrationForm.InitializeLanguage();

InitializeLanguage();
```

در نهایت تغییرات را درون فایل تنظیمات برنامه ذخیره می‌کنیم.

```
if (ConfigurationManager.AppSettings.AllKeys.Contains("DefaultLanguage")){

    Configuration configs =
ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

    configs.AppSettings.Settings["DefaultLanguage"].Value =
CurrentLanguage.Name;

    configs.Save(ConfigurationSaveMode.Modified);

    ConfigurationManager.RefreshSection("appSettings");

}
```

```
};

languageMenu.DropDownItems.Add(subMnuLanguage);

}

}
```

### متد CommonDataGridCellEndEdit

```
public static void CommonDataGridCellEndEdit<EntityType>(DataGridView
dataGrid, DataGridViewCellEventArgs e, Action<EntityType> editModel) where
EntityType : IBaseModel, new()

{

    if (e.RowIndex != -1 && e.RowIndex != dataGrid.NewRowIndex)

    {
```

ابتدا بررسی می‌کنیم که مقدار «Tag» خالی نباشد و همچنین State آن برابر با «Constructing» نباشد. که بیان گر این است که این سطر در حال ویرایش است.

```
if (dataGrid.Rows[e.RowIndex].Tag != null &&
(dataGrid.Rows[e.RowIndex].Tag as IBaseModel).State != ModelState.Constructing)
```

مقدار درون «DataBoundItem» را با «Tag» آن مقایسه می‌کنیم. اگر مساوی نبودند به این معنی است که این نمونه ویرایش شده است.

```
if (!(dataGrid.Rows[e.RowIndex].DataBoundItem as
IBaseModel).Equals(dataGrid.Rows[e.RowIndex].Tag as IBaseModel))

{
```

در صورت ویرایش شده State آن را به «Editing» تغییر می‌دهیم و رویداد «ModelsChanged» را فراخوانی می‌کنیم.

```
(dataGrid.Rows[e.RowIndex].DataBoundItem as
IBaseModel).State = ModelState.Editing;

if (ModelIsChanged != null)

    ModelIsChanged(dataGrid, e);

}
```

```
else
```

```
{
```

دی غیر این صورت وضعیت نمونه را به حالت قبلی بر می گردانیم.

```
(dataGrid.Rows[e.RowIndex].DataBoundItem as
IBaseModel).State = (dataGrid.Rows[e.RowIndex].Tag as IBaseModel).State;
```

```
if (ModelIsNotChanged != null)
```

```
ModelIsNotChanged(dataGrid, e);
```

```
}
```

```
//new
```

```
if (dataGrid.CurrentRow.Tag != null) {
```

زمانی این کد اجرا می شود که در سطری جدید قرار داشته باشیم. بررسی می کنیم که اگر State آن برابر با «Constructing» یا «Editing» بود وارد بدنه شرط بشود.

```
if ((dataGrid.CurrentRow.Tag as IBaseModel).State == ModelState.Constructing ||
(dataGrid.CurrentRow.DataBoundItem as IBaseModel).State == ModelState.Editing)
{
```

در اینجا یک نمونه از پارامتر جنریکمان می سازیم

```
EntityType newModel = new EntityType();
```

و اگر مقدار State آن برابر با «Constructing» بود مقدار «Tag» سطر جاری را به نمونه جدید نسبت می دهیم.

```
if (((EntityType)(dataGrid.CurrentRow.Tag)).State == ModelState.Constructing){
newModel = (EntityType)dataGrid.CurrentRow.Tag;
}
```

در صورتی که مقدار State نمونه موجود در «DataBoundItem» برابر با «Editing» بود نمونه جدید را برابر آن قرار می دهد.

```

else if ((dataGrid.CurrentRow.DataBoundItem as
IBaseModel).State == ModelState.Editing)
{
    newModel = (EntityType)dataGrid.CurrentRow.DataBoundItem;
}

```

در نهایت تابعی که از ورودی می گیرد را اجرا می کند. این تابع یک ورودی از نوع پارامتر جنریک می گیرد. که از همان نوع IBaseModel است.

```

editModel(newModel);
}
}
}
}

```



## پیاده‌سازی کلاس‌ها

### IBaseModel پیاده‌سازی

در اینترفیس‌ها فقط شمای کلی تعریف می‌شود و هیچ چیزی پیاده‌سازی نمی‌شود.

### نسخه معمولی

```
public interface IBaseModel : ICloneable, INotifyPropertyChanged{

    Guid Id { set; get; }

    ModelState State { set; get; }

    DataTable dataTable { set; get; }

    void Save();

    void Save(DataRow dataRow);

    void Edit();

    void Delete();

    bool Validate();

    void Fill(DataRow dataRow);

    event EventHandler SaveSuccessful;

    event EventHandler DeleteSuccessful;

    event EventHandler EditSuccessful;

    event EventHandler ReadSuccessful;

    event ModelErrorEventHandler Error;

}
```

### نسخه جنریک

پارامتر ورودی را از نوع «IBaseModel» در نظر می گیریم.

```
public interface IBaseModel<T> : IBaseModel where T : IBaseModel,
IBaseModel<T>, new()
{
    T Read(DataRow dataRow);
    T Read(Guid Id);

    bool Validate(T model);
    void Fill(DataRow dr, T model);
    IEnumerable<T> ReadAll();
}
```

### پیاده سازی کلاس BaseModel

این کلاس را به صورت «abstract» تعریف می کنیم. همانطور که در کد مشاهده می شود این کلاس اینترفیس «IBaseModel» و «IEquatable» را پیاده سازی می کند.

```
public abstract class BaseModel<T> : IBaseModel<T>, IEquatable<T> where T :
IBaseModel<T>, new()
{
}
}
```

ابتدا پراپرتی Id را به صورت زیر پیاده سازی می کنیم. تمام صفات در کلاس ها به شکلی مشابه پراپرتی Id تعریف می شوند.

```
private Guid _id;
public virtual Guid Id{
    set{
        _id = value;
    }
}
```

بعد از نسبت دادن مقدار دریافتی به فیلدمان (که به صورت Private) تعرف شده است.

متد «NotifyPropertyChanged» را صدا می زنیم. در ادامه این متد را پیاده سازی خواهیم کرد.

```
this.NotifyPropertyChanged("Id");  
}  
get{  
    return _id;  
}  
}
```

### اعضای های «abstract»

این پراپرتی ها و متد ها را «abstract» در نظر می گیریم از آنجا که پیاده سازی آن ها در هر کلاس متفاوت است.

```
public abstract T Read(DataRow dataRow);  
  
public abstract bool Validate(T model);  
public abstract bool Validate();  
  
public abstract void Fill(DataRow dr);  
public abstract void Fill(DataRow dr, T model);  
  
public abstract DataTable dataTable { set; get; }  
  
public abstract bool Equals(T other);  
public override abstract bool Equals(object obj);  
public override abstract int GetHashCode();
```

## پیاده‌سازی رویداد ها و متد های مربوطه

در تکه کد زیر کد مربوط به تعریف رویداد، ۳ رویداد با نوع های مختلف آورده شده است.

- **EventHandler**: هیچ پارامتر خاصی ندارد.
- **ModelErrorEventHandler**: این نوع که خود تعریف کرده‌ایم برای اعلام بروز خطا به کار می‌رود و حاوی فیلدی برای نگه داری Exception می باشد.
- **PropertyChangedEventHandler**: حاوی خصیصه ای با نام «PropertyName» می‌باشد که نام پراپرتی تغییر کرده را در خود نگه می دارد.

بقیه رویداد ها نیز به همین صورت تعریف می شوند.

```
public event EventHandler ReadSuccessful;

public event ModelErrorEventHandler Error;

public event PropertyChangedEventHandler PropertyChanged;
```

## متد NotifyPropertyChanged

این متد برای پاسخ‌گویی به این رویداد است و صورت زیر پیاده‌سازی می شود. در هر یک از پراپرتی ها هرگاه مقداری را به فیلدی نسبت می‌دهیم نام پراپرتی را به این متد می فرستیم.

```
protected void NotifyPropertyChanged(string name) {

    if (PropertyChanged != null)

        PropertyChanged(this, new PropertyChangedEventArgs(name));

}
```

## متد OnError

این متد یک ورودی از نوع «Exception» گیرد و از ارگمان های این رویداد یک نمونه ساخته و آن را توسط متد سازنده اش بهش نسبت می دهد.

```
protected void OnError(Exception exception){

    if (Error != null)
```

```
Error(this, new ModelEventArgs(exception));
}
```

کلاس ModelEventArgs به صورت زیر پیاده‌سازی می‌شود.

```
public class ModelEventArgs : EventArgs{
    public Exception Exception { get; set; }
    public ModelEventArgs(){ }
    public ModelEventArgs(Exception ex){
        this.Exception = ex;
    }
}
```

دلیگت این رویداد به این صورت تعریف می‌شود.

```
public delegate void ModelEventHandler(object sender, ModelEventArgs
e);
```

### متد OnReadSuccessful

این متد به صورت زیر پیاده‌سازی می‌شود. بقیه متد های مشابه برای پاسخگویی به رویداد ها نیز به همین صورت پیاده‌سازی می‌شوند.

```
protected void OnReadSuccessful(EventArgs e = null)
{
    if (e == null) e = EventArgs.Empty;
    if (ReadSuccessful != null)
        ReadSuccessful(this, e);
    this.State = ModelState.Read;
}
```

### پیاده‌سازی متد سازنده

در متد سازنده یک بار متد Initialize کلاس Database را صدا می‌زنیم تا نمونه Database آماده شود. و سپس مقدار «State» را برابر با «Constricted» قرار می‌دهیم. در بقیه کلاس‌ها روند کار به همین صورت است.

```
public BaseModel(){  
    Database.Initialize<SqlCeCommand, SqlCeConnection, SqlCeDataAdapter,  
    SqlCeCommandBuilder>(Common.ConnectionString);  
    State = ModelState.Constructed;  
}
```

### پیاده‌سازی متد getList

```
protected IEnumerable<T> getList(DataTable dataTable)  
{  
    IList<T> models = new List<T>();  
    foreach (DataRow dr in dataTable.Rows)  
    {
```

تمام سطرهای موجود در dataTable را می‌خوانیم و آن را به متد Read نمونه جدید از پارامتر جنریک می‌دهیم.

```
T model = new T();  
model.Read(dr);
```

در این خط بررسی می‌کنیم که اگر رویداد Error نمونه جاری پیاده‌سازی شده است آن را به رویداد Error نمونه‌های جدید نیز نسبت دهد. و در نهایت آن را به لیست اضافه می‌کنیم.

```
if (this.Error != null)  
    model.Error += this.Error;  
models.Add(model);  
}  
return models;
```

```
}
```

به عنوان مثال اگر نوع پارامتر جنریک را از نوع «Customer» بدهیم این متد برای ما یک لیست از «Customer» ها بر می گرداند.

### پیاده سازی متد ReadAll

```
public virtual IEnumerable<T> ReadAll()
{
```

در این قسمت از متد getDataTable نمونه Database استفاده می کنیم. کار این متد این است تا تمام مقادیر در جدولی که به عنوان ورودی داده شده است را می خواند و به عنوان یک DataTable بر می گرداند. ما از خروجی این متد استفاده می کنیم و آن را به dataTable نسبت می دهیم.

```
dataTable = Database.getDataTable(dataTable.TableName);
```

در این قسمت نیز dataTable را به متد getList می دهیم تا آن را تبدیل به لیست کند و آن را به لیست جنریک models نسبت می دهیم و سپس از آن برای مقدار دهی به LocalDataSource استفاده می کنیم.

درواقع با هربار فراخوانی این متد تمام مقادیر موجود در جدول مربوط به موجودیت مورد نظر خوانده شده و در حافظه بارگزاری می شود.

```
List<T> models = getList(dataTable).ToList();

LocalDataSource = new BindingList<T>(models);

return models;
}
```

### پیاده سازی متد Read

به طور کلی این متد یک ورودی از نوع Guid می گیرد و درون تمام مقادیر موجود به دنبال یک نمونه با Id مورد نظر می گردد. این متد را به صورت «virtual» تعریف می کنیم که بتوانیم در صورت لزوم پیاده سازی آن را تغییر دهیم.

```
public virtual T Read(Guid Id){
    T model;
```

ابتدا بررسی می‌کند که اگر مقدار LocalDataSoure خالی نباشد در آن جستجو به دنبال آن موجودیت را شروع می‌کند.

```
if (LocalDataSource != null)
    model = (from T m in LocalDataSource where m.Id == Id select
m).FirstOrDefault<T>();
else
```

در غیر این صورت یک بار متد ReadAll را صدا می‌زند و در لیستی که از خروجی آن بدست می‌آید به دنبال مقدار مورد نظر می‌گردد.

```
model = (from T m in ReadAll().OfType<T>() where m.Id.Equals(Id)
select m).FirstOrDefault<T>();
```

در نهایت با استفاده از متد «Copy» کلاس Iterator از مقادیر نمونه پیدا شده را به نمونه جاری نسبت می‌دهیم.

```
Iterator.Copy(this, model);
```

همانطور که در قبل گفته شد رویداد «ReadSuccessful» زمانی رخ می‌دهد که عمل خواندن با موفقیت انجام شود در اینجا این متد را صدا می‌زنیم تا در صورتی که این رویداد تعریف شده بود کد آن را اجرا کند. در نهایت نمونه بدست آورده شده را بر می‌گردانیم.

```
OnReadSuccessful();
return model;
}
```

## پیاده‌سازی متد Save

```
public virtual void Save(){
```

ابتدا یک سطر جدید ایجاد می‌کنیم. و آن را درون newRow می‌ریزیم.

```
DataRow newRow = dataTable.NewRow();
```



در این قسمت یک Guid جدید ساخته و مقدارش را به Id می دهیم.

```
this.Id = Guid.NewGuid();
```

در اینجا با فراخوانی متد Validate از صحت اطلاعات ورود اطمینان حاصل می کنیم. اگر Validate بدون ورودی صدا زده شود نمونه جاری را مورد بررسی قرار می دهد.

```
if (Validate()){
```

پس از اطلاع از صحت اطلاعات ورود آن ها را با استفاده از متد «Fill» در newRow می ریزیم. و در نهایت آن را به dataTable اضافه می کنیم.

```
Fill(newRow);
```

```
dataTable.Rows.Add(newRow);
```

برای اعمال تغییرات در پایگاه داده از تکه کدی مانند زیر استفاده می کنیم. کار متد Update کلاس DataAdapter این است که تغییرات را در پایگاه داده ذخیره کنید. لازم به ذکر است که کد های SQL و پارامتر های لازم در هر کلاس به طور جداگانه تعریف می شود.

```
Database.DataAdapter.Update(dataTable);
```

```
OnSaveSuccessful();
```

```
}
```

```
}
```

### پیاده سازی متد Save با ورودی DataRow

پیاده سازی این متد بسیار آسان است کافی است تا مقدار DataRow ورودی را بخوانیم و سپس متد Save معمولی را صدا بزنیم.

```
public void Save(DataRow dataRow)
```

```
{
```

```

        this.Read(dataRow);

        this.Save();

    }

```

## پیاده‌سازی متد Delete

```

public virtual void Delete(){

    if (this.Read(Id) != null)

    {

```

در اینجا متد Delete مربوط به dataRow را صدا می‌زنیم تا این سطر که متناظر با نمونه جاری است از DataTable حذف شود.

```

        this.dataRow.Delete();

```

در اینجا کد SQL لازم برای حذف را می‌نویسیم.

```

        Database.DataAdapter.DeleteCommand.CommandText =
string.Format("DELETE FROM {0} WHERE Id=@pId", dataTable.TableName);

        Database.InitializeParameters(Database.DataAdapter.DeleteCommand
.Parameters, "p", "Id");

        Database.DataAdapter.Update(dataTable);

        OnDeleteSuccessful();

    }

}

```

## پیاده‌سازی متد Edit

```

public virtual void Edit()

{

```

ابتدا یک نمونه جدید می‌سازیم و سپس مقادیر نمونه جاری را در نمونه جدید کپی می‌کنیم.

```
T modifiedModel = new T();
Iterator.Copy(modifiedModel, this);
```

حال برای صحت از وجود مقدار مورد نظر در پایگاه داده یکبار متد Read را صدا می‌زنیم. و اگر خروجی اش Null نبود کار را ادامه می‌دهیم.

```
if (this.Read(Id) != null)
{
```

در اینجا نمونه ویرایش شده را به ورودی متد Validate می‌دهیم تا از صحت داده‌های موجود در آن اطمینان حاصل شود سپس آن را به عنوان پارامتر ورودی دوم متد Fill می‌دهیم. این نسخه از متد Fill نمونه‌ای که در ورودی می‌گیرد را درون DataRow می‌ریزد.

```
if (Validate(modifiedModel))
{
    Fill(dataRow, modifiedModel);
}

Database.DataAdapter.Update(dataTable);
OnEditSuccessful();
}
}
```

### پیاده‌سازی متد Reload

مقادیر نمونه جاری را به حالت اول بر می‌گرداند. برای اینکار یک بار متد Read آن را صدا می‌زنیم.

```
public virtual void Reload()
{
    this.Read(this.Id);
}
```

## پیاده‌سازی متد Clone

برای پیاده‌سازی این کلاس که یکی از اعضای Icloneable می‌باشد کافی است تا متد CreateClone کلاس Iterator را اجرا کنیم و به عنوان ورودی نمونه جاری را بدهیم. این متد از مقدار ورودی یک کپی تهیه می‌کند و بر می‌گرداند.

```
public virtual object Clone()
{
    return Iterator.CreateClone(this);
}
```

## پیاده‌سازی کلاس Category

به طور کلی کلاس به این حالت تعریف می‌شود. همانطور که در تکه کد زیر مشاهده می‌شود از کلاس «BaseModel» با پارامتر جنریک از نوع همین کلاس به ارث می‌برد.

```
public class Category : BaseModel<Category>
{
}
```

## پیاده‌سازی متد سازنده

همانطور که قبلاً گفته شد کد های SQL برای اعمال مانند ذخیره سازی، و ویرایش در هر کلاس جداگانه نوشته می‌شود. برای اینکار از متد سازنده استفاده می‌کنیم.

```
public Category(){
    //Save
}
```

در اینجا کد SQL را به پراپرتی «CommandText» نمونه «InsertCommand» می‌دهیم. همچنین پارامتر های مورد نیاز نیز می‌سازیم.

در ادامه با فراخوانی متد «Update» که متعلق به «DataAdapter» است؛ دستور مورد نظر اجرا می شود.

```
Database.DataAdapter.InsertCommand.CommandText = @"INSERT INTO
Categories (Id,CategoryName) VALUES(@pId,@pCategoryName)";

Database.InitializeParameters(Database.DataAdapter.InsertCommand.Par
ameters, "p", "Id", "CategoryName");

//Edit
```

در اینجا کد SQL را به پراپرتی «CommandText» نمونه «UpdateCommand» می دهیم.

```
Database.DataAdapter.UpdateCommand.CommandText = @"UPDATE Categories
SET CategoryName=@pCategoryName WHERE Id=@pId";
```

```
Database.InitializeParameters(Database.DataAdapter.UpdateCommand.Par
ameters, "p", "Id", "CategoryName");

}
```

از آنجایی که برای عملی حذف کد یکسانی استفاده می شود؛ لازم نیست که برای هر کلاس آن کد را بنویسیم و عمل حذف را کلاس والد می سپاریم. برای عمل خواندن توسط «Id» این مساله نیز صادق است.

### پیاده سازی متد Read

در این متد مقادیر موجود در DataRow را خوانده و آن را به صفات مربوطه در نمونه جاری از «Category» نسبت می دهیم.

```
public override Category Read(DataRow dataRow)
{
```

همانطور که در کد مشخص است اندیس dataRow همانم به صفت مربوطه در کلاس می باشد. و پس از نسبت داده تک تک مقادیر به صفات مربوطه، dataRow را به dataRow درون نمونه نسبت می دهیم که از آن برای اعمال حذف و ویرایش استفاده می شود.

```
this.Id = Guid.Parse(dataRow["Id"].ToString());
```

```
this.CategoryName = dataRow["CategoryName"].ToString();

this.dataRow = dataRow;
```

در نهایت متد «OnReadSuccessful» را صدا می زنیم.

```
OnReadSuccessful();

return this;

}
```

در دیگر کلاس ها نیز این متد مشابه کلاس «Category» پیاده سازی می شود.

### پیاده سازی متد ReadFoods

در اینجا با استفاده از عبارت LINQ تمام غذاهایی که در دسته بندی جاری وجود دارد را می خوانیم. و آن را به لیست و سپس به BindingList جنریک با پارامتر «Food» تبدیل می کنیم. BindingList امکان تغییر اعضای آن توسط یک DataGridView را می دهد. برای همین موضوع معمولاً خروجی متد هایی مانند متد ReadFoods این نوع خروجی را دارا می باشند.

```
public BindingList<Food> ReadFoods()

{

    return new BindingList<Food>((from Food food in new Food().ReadAll()
where food.Category.Id == this.Id select food).ToList());

}
```

### پیاده سازی متد GetHashCode

به طور کلی برای پیاده سازی این متد خروجی متد GetHashCode صفات اصلی که همان هایی است که در جدول نیز وجود دارند را با هم XOR می کنیم.

از آنجایی که متد «Equals» را پیاده سازی می کنیم بهتر است این متد هم پیاده سازی کنیم.

```
public override int GetHashCode()

{
```

```
return Id.GetHashCode() ^ CategoryName.GetHashCode();
}
```

در کلاس‌های دیگر این متد به روشی مشابه پیاده‌سازی می‌شود.

### پیاده‌سازی متد Equals با ورودی Category

این متد در اصل یکی از اعضای اینترفیس «IEquatable» می‌باشد که عضو آن متدی مانند کد زیر است.

```
public override bool Equals(Category other){
```

ابتدا بررسی می‌کنیم که اگر مقدار نمونه ورودی برابر با «null» بود «false» را بر می‌گردانیم.

```
if (other == null) return false;
```

سپس نوع داده‌ای نمونه جاری با نمونه‌ای که از ورودی گرفته‌ایم را با استفاده از متد GetType بررسی می‌کنیم و اگر با هم برابر نبودند «false» بر می‌گردانیم.

```
if (other.GetType() != this.GetType()) return false;
return (
```

در نهایت تک تک صفات نمونه جاری را با نمونه ورودی مقایسه می‌کنیم اگر تمام صفات با هم برابر بودند مقدار «true» بازگشت داده می‌شود. برای مقادیر ارجاعی مانند مواردی که یک صفت از نوع کلاسی دیگر در کلاس‌مان داریم برای مقایسه از متد «Equals» استفاده می‌کنیم. برای دیگر کلاس‌ها نیز از روندی مشابه استفاده می‌شود.

```
Id.Equals(other.Id) &&
CategoryName == other.CategoryName
);
}
```

### پیاده‌سازی متد Equals با ورودی object

برای پیاده‌سازی این متد در تمام کلاس‌ها کافی است تا ورودی را به نوع همان کلاس تبدیل کرده و به متد Equals دیگر بدهیم.

```
public override bool Equals(object obj)
{
    return Equals(obj as Category);
}
```

### پیاده‌سازی متد Validate با ورودی Category

در پیاده‌سازی این متد صفات را با توجه به منطق سیستم بررسی می‌کنیم. سپس نتایج را با هم AND بیتی می‌کنیم و حاصل را بر می‌گردانیم.

```
public override bool Validate(Category category)
{
    bool result = true;
```

در اینجا بررسی می‌کنیم که Id ما برابر با یک Guid خالی نباشد یک Guid خالی تمام بیت‌های آن مقدار «۰» دارد .

```
result &= category.Id != Guid.Empty;
```

در اینجا نیز بررسی می‌کنیم که «نام دسته بندی» حتماً از حروف الفبا تشکیل بشود.

```
result &= Validator.IsAlphabetical(category.CategoryName);
return result;
}
```

### پیاده‌سازی متد Validate با ورودی object

برای پیاده‌سازی این متد در تمام کلاس‌ها کافی است تا نمونه جاری را به ورودی متد Validate دیگر بدهیم.

```
public override bool Validate(){
    return Validate(this);
}
```



```
}
```

### پیاده‌سازی متد Fill با ورودی Category و DataRow

این متد دقیقاً عکس متد Read با ورودی DataRow عمل می‌کند. پیاده‌سازی آن به این شکل است که تک تک صفات نمونه ورودی که ورودی دوم است را به اندیس مربوطه در dr نسبت می‌دهیم.

```
public override void Fill(DataRow dr, Category category){

    dr["Id"] = category.Id.ToString();

    dr["CategoryName"] = category.CategoryName;

}
```

### پیاده‌سازی متد Fill با ورودی DataRow

در تمام کلاس‌ها به این صورت پیاده‌سازی می‌شود. فقط کافی است تا به عنوان پارامتر ورودی دوم متد Fill دو ورودی نمونه جاری را بدهیم.

```
public override void Fill(DataRow dr) {

    Fill(dr, this);

}
```

### پیاده‌سازی پراپرتی CategoryName

```
public string CategoryName{

    get{
```

در قسمت get فقط مقدار فیلد «\_categoryName» را بر می‌گردانیم.

```
        return _categoryName;

    }

    set{
```

در قسمت set بررسی می‌کنیم که مقدار «value» از حروف الفبا تشکیل شده است یا خیر. اگر از حروف الفبا بود آن را به فیلد نسبت می‌دهیم و همچنین متد NotifyPropertyChanged را فراخوانی می‌کنیم و نام پراپرتی را به آن می‌دهیم.

```
if (Validator.IsAlphabetical(value)){
    _categoryName = value;
    this.NotifyPropertyChanged("CategoryName");
}
else{
```

اگر شرط برقرار نبود یک نمونه از FormatException با پیغام مناسب می‌سازیم و به ورودی متد OnError می‌دهیم.

```
OnError(new FormatException("Invalid Input(Alphabet)"));
}
}
}
```

بقیه پراپرتی‌ها هم مشابه این پراپرتی پیاده‌سازی می‌شوند. و فقط در نوع شرط تفاوت دارند.

### پیاده‌سازی پراپرتی dataTable

فیلد «tblCategories» خود به صورت static می‌باشد به این معنی است که فقط از طریق نام کلاس می‌توان به آن دسترسی داشت و حافظه آن بین تمام نمونه‌های این کلاس مشترک است. این نمونه علاوه بر اینکه به ما این امکان را می‌دهد در در نمونه نیز به این فیلد دسترسی داشته باشیم، به ما اجازه می‌دهد از آن در کلاس والد برای اعمالی همچون خواندن، حذف، و ویرایش از آن استفاده کنیم.

```
public override DataTable dataTable{
    get{
```

در قسمت get اگر DataTable مربوطه مقدار دهی نشده بود یک نمونه از آن ساخته و به آن نسبت می دهیم. و سپس آن را بر می گردانیم.

```
if (tblCategories == null)
    tblCategories = new DataTable("Categories");

return tblCategories;
}

set{
```

در قسمت set فقط مقداری که ورودی می گیریم را به DataTable نسبت می دهیم.

```
tblCategories = value;
}

}
```

در کلاس های دیگر نیز نحوه پیاده سازی مشابه همین کلاس می باشد.

## پیاده سازی کلاس Food

### پیاده سازی متد سازنده

مانند متد سازنده Category می باشد با این تفاوت که مقادیر CommandText در دو نمونه InsertCommand و UpdateCommand فرق دارند.

```
//Save

Database.DataAdapter.InsertCommand.CommandText = @"INSERT INTO Foods
(Id,FoodName,Price,Category_Id,Enabled)
VALUES(@pId,@pFoodName,@pPrice,@pCategory_Id,@pEnabled)";

Database.InitializeParameters(Database.DataAdapter.InsertCommand.Parameters, "p", "Id", "FoodName", "Price", "Category_Id", "Enabled");

//Edit

Database.DataAdapter.UpdateCommand.CommandText = @"UPDATE Foods SET
FoodName=@pFoodName,Price=@pPrice,Category_Id=@pCategory_Id,Enabled=@pEnabled
WHERE Id=@pId";
```

```
Database.InitializeParameters(Database.DataAdapter.UpdateCommand.Parameters, "p", "Id", "FoodName", "Price", "Category_Id", "Enabled");
```

### پیاده‌سازی متد Read

در پیاده‌سازی متد Read فقط در جاهایی که نسبت به پیاده‌سازی متد Read کلاس Category فرق دارد بیان می‌شود.

در اینجا مقدار مورد نظر را از DataRow گرفته و به متد SanitizeFloat کلاس Sanitizer می‌دهیم. و خروجی را به صفت «Price» نسبت می‌دهیم. همین کار را برای صفت «Enabled» انجام می‌دهیم.

```
this.Price = Sanitizer.SanitizeFloat(dataRow["Price"].ToString());
this.Enabled = Sanitizer.SanitizeBool(dataRow["Enabled"], true);
```

در اینجا هم مقدار «Category\_Id» را به متد Read نمونه جدید «Category» می‌دهیم. در کل صفاتی که از نوع کلاسی دیگر است به طور عمل می‌کنیم. مقدار فیلدی که در آن Id آن موجودیت وجود دارد را به متد Read همان موجودیت می‌دهیم تا برود و اطلاعات مربوط به آن Id را بخواند.

```
this.Category = new
Category().Read(Guid.Parse(dataRow["Category_Id"].ToString()));
```

### پیاده‌سازی متد ReadByCategory

```
public BindingList<Food> ReadByCategory(Category category){
```

ابتدا بررسی می‌کنیم که مقدار ورودی خالی نباشد.

```
if (category != null)
{
```

سپس بررسی می‌کنیم که مقدار «Id» آن خالی نباشد. اگر خالی بود تمام غذاها را بر می‌گردانیم.

```
if (category.Id != Guid.Empty)
{
```

با استفاده از این عبارت LINQ غذا هایی که مقدار دسته بندی آنها برابر با دسته بندی ورودی است بر گردانده می شود.

```
return new BindingList<Food>((from Food food in
LocalDataSource where food.Category.Id == category.Id select food).ToList());

}
```

و اگر مقدار «Id» خالی بود «LocalDataSource» را بر می گردانیم که یک لیست از تمام نمونه های کلاس جاری در آن است.

```
else return LocalDataSource;

}

else return null;

}
```

### پیاده سازی متد Search

```
public BindingList<Food> Search(string searchQuery,Category category=null){
```

اگر پارامتر ورودی دوم متد که دسته بندی را مشخص می کند داده شده بود در دسته بندی داده شده جستجوی می کند.

```
if (category != null && !category.Id.Equals(Guid.Empty)){

return new BindingList<Food>((from Food food in LocalDataSource

where food.FoodName != null &&
food.FoodName.ToLower().Contains(searchQuery.ToLower()) &&
food.Category.Equals(category)

select food).ToList());

}
```

در غیر این صورت در تمام دسته بندی ها به دنبالش می گردد.

```
else{

return new BindingList<Food>((from Food food in LocalDataSource

where food.FoodName != null &&
food.FoodName.ToLower().Contains(searchQuery.ToLower())
```

```

        select food).ToList());
    }
}

```

### پایاده‌سازی متدهای IncreaseQuantity و DecreaseQuantity

```
public void IncreaseQuantity(Order order){
```

ابتدا به پایگاه داده متصل می‌شویم. بعد چنین کد SQL می‌نویسیم و پارامترهای لازم را ایجاد می‌کنیم.

```

        Database.Connect();

        Database.DataAdapter.UpdateCommand.CommandText = "UPDATE
Foods_Orders SET Quantity=@pQuantity WHERE Food_Id=@pFood_Id AND
Order_Id=@pOrder_Id";

        Database.InitializeParameters(Database.DataAdapter.UpdateCommand.Par
ameters, "p", "Order_Id", "Food_Id", "Quantity");

```

حال به پارامترها مقدار دهی می‌کنیم.

```

        Database.DataAdapter.UpdateCommand.Parameters["pOrder_ID"].Value =
order.Id.ToString();

        Database.DataAdapter.UpdateCommand.Parameters["pFood_Id"].Value =
this.Id.ToString();

```

در اینجا به مقدار «Quantity» یکی اضافه کرده و آن را به پارامتر مربوطه می‌دهیم.

```

        Database.DataAdapter.UpdateCommand.Parameters["pQuantity"].Value = +
+Quantity;

        Database.DataAdapter.UpdateCommand.ExecuteNonQuery();

        Database.Disconnect();
    }
}

```

متد «DecreaseQuantity» هم به همین صورت پایاده‌سازی می‌شود با این تفاوت که بجای اضافه کردن مقدار «Quantity» از آن یکی کم می‌کنیم.

## پیاده‌سازی دیگر اعضا

در این قسمت پیاده‌سازی اعضای دیگر کلاس گفته می‌شود که پیاده‌سازی آن‌ها قبلاً شرح داده شده است و فقط در جاهایی که تفاوت وجود دارد شرح داده می‌شود.

## پراپرتی dataTable

مانند کلاس «Category» است با این تفاوت که tblFoods را بر می‌گرداند.

## متد Validate با ورودی Food

برسی می‌کند که «نام غذا» از حروف الفبا باشد همچنین این برسی در پراپرتی «FoodName» نیز انجام می‌شود.

برای برسی دسته بندی از متد Validate آن استفاده می‌کند.

```
result &= Validator.IsAlphabetical(food.FoodName);

result &= Category.Validate();

result &= Price != 0.0f;
```

## پیاده‌سازی کلاس Employee

### پیاده‌سازی متد سازنده

کد مربوط به ذخیره سازی.

```
Database.DataAdapter.InsertCommand.CommandText = @"INSERT INTO
Employees (Id,FirstName,LastName,Username,Password,Role)
VALUES(@pId,@pFirstName,@pLastName,@pUsername,@pPassword,@pRole)";

Database.InitializeParameters(Database.DataAdapter.InsertCommand.Parameters,
"p", "Id", "FirstName", "LastName", "Username", "Password", "Role");
```

کد مربوط به ویرایش.

```
Database.DataAdapter.UpdateCommand.CommandText = @"UPDATE Employees SET
FirstName=@pFirstName,LastName=@pLastName,Username=@pUsername,Password=@pPassword,Role=@pRole WHERE Id=@pId";

Database.InitializeParameters(Database.DataAdapter.UpdateCommand.Parameters,
"p", "Id", "FirstName", "LastName", "Username", "Password", "Role");
```

### پیاده‌سازی پراپرتی FullName

این پراپرتی بین «نام» و «نام خانوادگی» یک فاصله قرار داده و آن را بر می گرداند.

```
public string FullName{  
    get{  
        return string.Format("{0} {1}", FirstName, LastName);  
    }  
}
```

### پیاده‌سازی متد Authenticate

با استفاده از LINQ بررسی می‌کنیم که آیا نمونه‌ای با «نام کاربری» و «کلمه عبور» مورد نظر در سیستم وجود دارد یا خیر.

```
public Role Authenticate(){  
    var result = from Employee employee in ReadAll()
```

برای نام کاربری آن ره به حروف کوچک تبدیل می‌کنیم و سپس آن را با مقدار کوچک شده نام کاربری نمونه جاری مقایسه می‌کنیم.

```
where employee.Username.ToLower() == this.Username.ToLower() &&
```

کلمه عبور را نیز با مقدار هش شده کلمه عبور نمونه جاری مقایسه می‌کنیم. که در این سیستم از الگوریتم SHA1 برای هش کردن کلمه عبور استفاده می‌شود.

```
employee.Password == Common.SHA1Hash(this.Password)  
select employee;  
  
if (result.Count<Employee>() > 0) {  
    Iterator.Copy(this, result.First<Employee>());  
    return this.Role;  
}
```



```

else
    return Role.Unauthorized;
}

```

### پیاده‌سازی متد Search

با استفاده از LINQ هر نمونه‌ای که رشته ورودی شامل «نام کامل» آن شود بر می گردانیم.

```

public BindingList<Employee> Search(string searchQuery)
{
return new BindingList<Employee>((from Employee employee in LocalDataSource
                                where
employee.FullName.ToLower().Contains(searchQuery)
                                select
employee).ToList<Employee>());
}

```

### پیاده‌سازی اعضای دیگر

#### پراپرتی dataTable

مقدار tblEmployees بازگشت داده می شود.

#### پراپرتی Role

در قسمت set آن اگر مقدار ورودی «Deafault» باشد کارمند را از نوع «Cashier» در نظر می گیرد.

```

if (value == Logic.Role.Default)
    value = Logic.Role.Cashier;

```

### متد Validate با ورودی Employee

در این متد بررسی می‌شود که مقدار «FirstName» و «LastName» از حروف الفبا باشد و مقدار «Username» و «Password» هیچ گاه خالی نباشند. این بررسی ها در پراپرتی ها نیز انجام می شود.

### متد Fill دو ورودی

برای نسبت دادن مقدار «Password» این قسمت مشخص می‌کند که آیا این یک کارمند جدید است یا یک کارمند قبلی که دارد ویرایش می‌شود. از آنجایی که برای ذخیره سازی نسخه یک ورودی این متد صدا می‌شود اگر مقدار نمونه جاری برابر با مقدار نمونه ورودی باشد به این معنی است که این یک کارمند جدید است.

```
if (employee != this) {
```

در این مرحله بررسی می‌کنیم که آیا «Password» کارمند عوض شده است یا خیر اگر «Password» هر دو نمونه با هم برابر نباشند به این معنی است که «Password» عوض شده است، پس مقدار جدید را هش کرده و به dr نسبت می‌دهیم.

```
if (employee.Password != this.Password)
    dr["Password"] = Common.SHA1Hash(employee.Password);
else
```

اگر «Password» هر دو نمونه یکی باشد به این معنی است که عوض نشده است و فقط مقدار آن را به dr می‌دهیم و نیازی به هش کردن نیست.

```
dr["Password"] = employee.Password ;

}
```

اگر کارمند جدید باشد مقدار «Password» را هش کرده و به dr نسبت می‌دهیم.

```
else if (employee == this){
    dr["Password"] = Common.SHA1Hash(employee.Password);
}
```

## پیاده‌سازی کلاس Customer

### پیاده‌سازی متد سازنده

کدهای مربوط به ذخیره

```

Database.DataAdapter.InsertCommand.CommandText = @"INSERT INTO
Customers (Id,FirstName,LastName,Address,PhoneNumber,CustomerId)
VALUES(@pId,@pFirstName,@pLastName,@pAddress,@pPhoneNumber,@pCustomerId)";

Database.InitializeParameters(Database.DataAdapter.InsertCommand.Par
ameters, "p", "Id", "FirstName", "LastName", "Address", "PhoneNumber",
"CustomerId");

```

کد های مربوط به ویرایش

```

Database.DataAdapter.UpdateCommand.CommandText = @"UPDATE Customers
SET
FirstName=@pFirstName,LastName=@pLastName,Address=@pAddress,PhoneNumber=@pPhoneN
umber,CustomerId=@pCustomerId WHERE Id=@pId";

Database.InitializeParameters(Database.DataAdapter.UpdateCommand.Par
ameters, "p", "Id", "FirstName", "LastName", "Address", "PhoneNumber",
"CustomerId");

```

### پیاده‌سازی متد Search

با استفاده از عبارت LINQ در «PhoneNumber»، «FullName»، و «CustomerId» به دنبال رشته مورد نظر می‌گردیم و نتایج را بر می‌گردانیم.

```

public BindingList<Customer> Search(string searchQuery)
{
return new BindingList<Customer>((from Customer customer in LocalDataSource
where customer.FullName.ToLower().Contains(searchQuery.ToLower())||
customer.PhoneNumber.ToLower().Contains(searchQuery.ToLower())||
customer.CustomerId.ToString().Contains(searchQuery)
select customer).ToList());
}

```

### پیاده‌سازی متد ReadOrders

```

public BindingList<Order> ReadOrders()
{

```

ابتدا یک نمونه از «\_orderController» می‌سازیم.

```

if (_orderController == null)

```

```
_orderController = new Order();
```

در اینجا تمام سفارش‌ها را خوانده و آن‌ها یک توسط این مشتری ثبت شده است را به لیست اضافه می‌کند.

```
foreach (Order order in _orderController.ReadAll().Where(order =>
order.Customer.Id.Equals(this.Id)))
{
```

از آنجایی که در پیاده‌سازی «Order» فقط «Id» ذخیره می‌شود. نمونه مشتری جاری را به سفارش نسبت می‌دهیم.

```
order.Customer = this;

Orders.Add(order);

}

return Orders;

}
```

### پیاده‌سازی متد DeleteOrders

```
public void DeleteOrders(){

    foreach (Order order in Orders){
```

ابتدا غذا های سفارش داده شده توسط تمام سفارش های این مشتری را پاک می‌کنیم.

```
order.DeleteFoods();

}
```

ابتدا غذا های سفارش داده شده توسط تمام سفارش های این مشتری را پاک می‌کنیم.

```
Database.ExecuteNonQuery(String.Format("DELETE FROM Orders WHERE
Customer_Id='{0}'", Id.ToString()), true);

}
```

### پیاده‌سازی متد **FilterOrders**

این متد به عنوان ورودی یک تابع با ورودی «Order» و خروجی «bool» می‌گیرد و آن را به ورودی متد «Where» می‌دهد که عبارت ورودی در اصل یک شرط است و تمام سفارش‌هایی که با این شرط همخوانی داشته باشند برگردانده می‌شود. این متد در پیاده‌سازی سازی متد های دیگر نیز کاربرد دارد.

```
public BindingList<Order> FilterOrders(Func<Order, bool> condition)
{
    return new BindingList<Order>(Orders.Where(condition).ToList<Order>());
}
```

### پیاده‌سازی متد **ReadOrdersByType**

نوع سفارش را به عنوان ورودی می‌گیرد و توسط عبارت Lambda زیر مشخص می‌کند که هر سفارشی که از این نوع باشد را انتخاب کند.

```
public BindingList<Order> ReadOrdersByType(OrderType type){
    return FilterOrders(order => order.OrderType == type);
}
```

### پیاده‌سازی متد **ReadOrdersByStatus**

وضعیت سفارش را به عنوان ورودی می‌گیرد و توسط عبارت Lambda زیر مشخص می‌کند که هر سفارشی که از این وضعیت باشد را انتخاب کند.

```
public BindingList<Order> ReadOrdersByStatus(OrderStatus status){
    return FilterOrders(order => order.Status == status);
}
```

### پیاده‌سازی متد **getCustomersId**

تمام «CustomerId» های موجود را می‌خوانیم و پس از تبدیل آن به یک لیست آن را بر می‌گردانیم. این متد برای تولید مقدار «CustomrId» استفاده می‌شود.

```
ICollection getCustomerIds(){
```

```
return (from DataRow dr in tblCustomers.Rows select
(int)dr["CustomerId"]).ToList();

}
```

### پیاده‌سازی پراپرتی CustomerId

```
public int CustomerId{

    get{
```

در اینجا بررسی می‌کنیم و اگر مقدار فیلد «\_customerId» خالی بود مقدار آن را با خروجی متد «NextId» کلاس «UniqueID» پر کنیم. همانطور که در کد مشخص است خروجی متد «getCustomerIds» را به این متد داده ایم.

```
if (_customerId == 0)

    _customerId = UniqueID.NextId(getCustomerIds());

return _customerId;

}

set{
```

در اینجا بررسی می‌کنیم که شماره ورودی قبلاً وجود نداشته باشد در صورت وجود یک شماره جدید گرفته و بجای مقدار ورودی در فیلد «\_customerId» می‌ریزد.

```
if (!UniqueID.IdExists(value))

    _customerId = value;

else

    _customerId = UniqueID.NextId(getCustomerIds());

this.NotifyPropertyChanged("CustomerId");

}

}
```

### پیاده‌سازی اعضای دیگر

متد Read

در این متد بعد اتمام خواند DataRow این متد «ReadOrders» را صدا می‌زنیم تا سفارش های مربوط به این مشتری خوانده شود.

```
this.ReadOrders();
```

### متد Delete

در این متد ابتدا متد «DeleteOrders» صدا زده می‌شود سپس متد «Delete» کلاس والد صدا می‌زند تا مشتری را پاک کند.

```
public override void Delete()
{
    DeleteOrders();

    base.Delete();
}
```

## پیاده‌سازی کلاس Order

### پیاده‌سازی متد سازنده

کد مربوط به ذخیره سازی

```
Database.DataAdapter.InsertCommand.CommandText = @"INSERT INTO
Orders (Id, Customer_Id, DeliveryFee, OrderDate, OrderType, Discount, Status)
VALUES (@pId, @pCustomer_Id, @pDeliveryFee, @pOrderDate, @pOrderType, @pDiscount, @pSta
tus)";

Database.InitializeParameters(Database.DataAdapter.InsertCommand.Par
ameters, "p", "Id", "Discount", "DeliveryFee", "OrderDate", "OrderType",
"Customer_Id", "Status");
```

کد مربوط به ویرایش

```
Database.DataAdapter.UpdateCommand.CommandText = @"UPDATE Orders SET
Discount=@pDiscount, DeliveryFee=@pDeliveryFee, Status=@pStatus, OrderType=@pOrderT
ype, Customer_Id=@pCustomer_Id WHERE Id=@pId";

Database.InitializeParameters(Database.DataAdapter.UpdateCommand.Par
ameters, "p", "Id", "Discount", "DeliveryFee", "OrderType", "Customer_Id",
"Status");
```

```
}
```

## پیاده‌سازی پراپرتی FirstOrder

```
public Order FirstOrder{  
    get{  
        if (LocalDataSource != null && LocalDataSource.Count > 0)  
        {
```

به طور پیش فرض از متد «First» استفاده می‌کنیم و آن را به عنوان اولین «سفارش» در نظر می‌گیریم.

```
Order firstOrder = LocalDataSource.OfType<Order>().First<Order>();  
foreach (Order order in LocalDataSource)  
{
```

تاریخ آن را با تمام سفارش‌ها مقایسه می‌کنیم و اگر سفارشی پیدا شده که تاریخی زودتر از تاریخ «firstOrder» داشت مقدار آن را دروین این متغیر میریزیم.

```
if (order.OrderDate <= firstOrder.OrderDate)  
    firstOrder = order;  
}
```

در نهایت مقدار مورد نظر که اولین سفارش است را بر می‌گردانیم.

```
return firstOrder;  
}  
else return null;  
}  
}
```



### پیاده‌سازی پراپرتی LastOrder

دقیقاً عکس متد پراپرتی «FirstOrder» عمل می‌کنیم. کافی است تا کد درون حلقه را به کد زیر تغییر دهیم تا آخرین سفارش را بگیریم.

```
if (order.OrderDate >= lastOrder.OrderDate)

    lastOrder = order;
```

### پیاده‌سازی پراپرتی RawPrice

```
public double RawPrice{

    get{

        double rawPrice = 0.0;

        foreach (Food food in Foods)
```

قیمت اصلی هر غذا را در تعداد سفارش داده شده ضرب می‌کنیم. این کار را به ازای تمام غذا های سفارش داده شده انجام می‌دهیم و حاصل آن‌ها را با هم جمع می‌کنیم.

```
rawPrice += (food.Price * food.Quantity);
```

در آخر هزینه حمل و نقل را اضافه کرده و مقدار بدست آمده را بر می‌گردانیم.

```
rawPrice += DeliveryFee;

    return rawPrice;

}

}
```

### پیاده‌سازی پراپرتی DiscountedPrice

در قسمت get این پراپرتی کد زیر را قرار می‌دهیم. این کد به ما هزینه ای که تخفیف داده می‌شود را بر می‌گرداند.

```
return (RawPrice * ((float)Discount / 100));
```

### پیاده‌سازی پراپرتی TotalPrice

در قسمت get این پراپرتی کد زیر را می‌نویسیم. فقط کافی است تا مقدار «قیمت خام» را از «میزان قیمت تخفیف داده شده» کم می‌کنیم و حاصل را بر می‌گردانیم.

```
return RawPrice - DiscountedPrice;
```

### پیاده‌سازی متد InitializeFoodsTable

```
private static void InitializeFoodsTable(){  
    if (tblFoods == null)  
    {  
        tblFoods = new DataTable("Foods");
```

فیلد های موجود در جدول «Foods\_Orders» را اضافه می‌کنیم.

```
tblFoods.Columns.Add("Food_Id");  
tblFoods.Columns.Add("Order_Id");  
tblFoods.Columns.Add("Quantity");
```

فیلد های موجود در جدول «Foods» را اضافه می‌کنیم.

```
tblFoods.Columns.Add("Id");  
tblFoods.Columns.Add("FoodName");  
tblFoods.Columns.Add("Price");  
tblFoods.Columns.Add("Enabled");  
tblFoods.Columns.Add("Category_Id");
```

کلید اصلی برابر با دو فیلد «Order\_Id» و «Food\_Id» قرار می‌دهیم.

```
tblFoods.PrimaryKey = new DataColumn[]  
{ tblFoods.Columns["Food_Id"], tblFoods.Columns["Order_Id"] };  
  
}  
  
}
```

## پیاده‌سازی متد ReadFoods

```
public List<Food> ReadFoods(){  
  
    Foods.Clear();  
  
    InitializeFoodsTable();
```

ابتدا این کد SQL را می‌نویسیم و پارامترهای لازم را درست می‌کنیم.

```
Database.DataAdapter.SelectCommand.Parameters.Clear();  
  
Database.DataAdapter.SelectCommand.CommandText = @"SELECT  
    Foods.*,Foods_Orders.* FROM Foods INNER JOIN Foods_Orders ON  
        (Foods.Id=Foods_Orders.Food_Id) INNER JOIN Orders ON  
        (Orders.Id=Foods_Orders.Order_Id) WHERE Orders.Id=@pId";  
  
Database.DataAdapter.SelectCommand.Parameters.Add(Database.CreateParameter("pId", "Id", Id.ToString()));  
  
Database.DataAdapter.Fill(tblFoods);
```

پس از پر کردن tblFoods از متد «Select» آن استفاده می‌کنیم تا تمام سطرهایی که با «Id» سفارش جاری برابر است را بخوانیم. سپس در بین آن‌ها حلقه می‌زنیم.

```
foreach (DataRow dr in tblFoods.Select(string.Format("Order_Id='{0}'",  
Id.ToString())))  
  
    {  
  
        Food food = new Food();
```

در اینجا «غذا» های سفارش داده را می‌خوانیم همچنین صفت «Quantity» آن‌ها را مقدار دهی می‌کنیم.

```
food.Read(dr);  
  
food.Quantity = int.Parse(dr["Quantity"].ToString());  
  
Foods.Add(food);
```

```

    }

    return Foods.ToList<Food>();
}

```

### پیاده‌سازی متد SaveFoods

```

public void SaveFoods(){
    InitializeFoodsTable();
}

```

به ازای تمام «غذا» های موجود در لیست «Foods» یک DataRow به نام newRow ساخته و مانند کد زیر مقدار دهی می کنیم.

```

foreach (Food food in Foods){
    DataRow newRow = tblFoods.NewRow();
    newRow["Food_Id"] = food.Id.ToString();
    newRow["Quantity"] = food.Quantity.ToString();
    newRow["Order_Id"] = this.Id;
}

```

آنها به سطرهای tblFoods اضافه می کنیم.

```

tblFoods.Rows.Add(newRow);
}

```

از این کد SQL برای وارد کردن مقادیر در جدول «Foods\_Orders» استفاده می کنیم.

```

Database.DataAdapter.InsertCommand.CommandText = @"INSERT INTO Foods_Orders
(Food_Id,Order_Id,Quantity) VALUES(@pFood_Id,@pOrder_Id,@pQuantity)";

Database.InitializeParameters(Database.DataAdapter.InsertCommand.Parameters,
"p", "Food_Id", "Order_Id", "Quantity");

Database.DataAdapter.Update(tblFoods);
}

```

### پیاده‌سازی متد SaveFood

مانند متد «SaveFoods» عمل می‌کنیم با این تفاوت که «غذا» را از ورودی متد می‌گیریم و وارد جدول می‌کنیم.

### پیاده‌سازی متد DeleteFoods

برای این کار از کد SQL زیر استفاده می‌کنیم.

```
public void DeleteFoods(){
    Database.DataAdapter.DeleteCommand.CommandText = "DELETE FROM
    Foods_Orders WHERE Food_Id=@pFood_Id AND Order_Id=@pOrder_Id";

    Database.InitializeParameters(Database.DataAdapter.DeleteCommand.Parameters, "p", "Food_Id", "Order_Id");
}
```

بعد با استفاده از کد زیر تمام غذا هایی که سفارش داده شده را پیدا کرده و پاک می‌کنیم.

```
foreach (Food food in Foods)
{
    tblFoods.Rows.Find(new object[] { food.Id, Id }).Delete();

    Database.DataAdapter.Update(tblFoods);
}
```

### پیاده‌سازی متد DeleteFood

مانند متد «DeleteFoods» عمل می‌کنیم با این تفاوت که فقط غذایی که از ورودی متد گرفته شده است را پاک می‌کنیم.

```
tblFoods.Rows.Find(new object[] { food.Id, Id }).Delete();
```

### پیاده‌سازی متد های آمار گیری

#### متد getToatalSales

```
public double getTotalSales(Func<Order, bool> condition){
```

ابتدا همه سفارش ها را در صورت لزوم می‌خوانیم.

```
if (LocalDataSource == null)
{
    ReadAll();
}
```

```
double sales = 0.0;
```

بعد با توجه به تابعی که از ورودی می‌گیریم «سفارش»‌ها را انتخاب کرده و مقدار «قیمت کل» آن‌ها را با هم جمع می‌بندیم.

```
if (LocalDataSource.Count > 0){  
    foreach (Order order in LocalDataSource.OfType<Order>().Where(condition))  
    {  
        sales += order.TotalPrice;  
    }  
}
```

در نهایت این مقدار را بر می‌گردانیم.

```
return sales;
```

```
}
```

### متد `getTotalAnnualSales`

تمام سفارش‌هایی که در یک سال خاص به ثبت رسیده‌اند را بر می‌گرداند. مقدار سال را از ورودی می‌گیرید.

```
return getTotalSales(order => order.OrderDate.Year == Year);
```

### متد `getTotalMonthlySales`

تمام سفارش‌هایی که در یک سال و ماه خاص به ثبت رسیده‌اند را بر می‌گرداند. مقدار سال و ماه را از ورودی می‌گیرید.

```
return getTotalSales(order => order.OrderDate.Month == Month &&  
order.OrderDate.Year == Year);
```

### متد getTotalDailySales

تمام سفارش هایی که در یک سال و ماه و روز خاص به ثبت رسیده اند را بر می گرداند. مقدار سال و ماه و روز را از ورودی می گیرید.

```
return getTotalSales(order => order.OrderDate.Year == Year &&
order.OrderDate.Month == Month && order.OrderDate.Day == Day);
```

### متد getTotalHourlySales

تمام سفارش هایی که در یک سال و ماه و روز و ساعت خاص به ثبت رسیده اند را بر می گرداند. مقدار سال و ماه و روز و ساعت را از ورودی می گیرید.

```
return getTotalSales(order => order.OrderDate.Year == Year &&
order.OrderDate.Month == Month && order.OrderDate.Day == Day &&
order.OrderDate.Hour == Hour);
```

## پیاده سازی اعضای دیگر

### متد Read

در متد «Read» یک نمونه از «Customer» ساخته می شود و «Id» آن برابر با مقدار گرفته شده از dataRow قرار داده می شود.

```
this.Customer = new Customer(){
    Id = Guid.Parse(dataRow["Customer_Id"].ToString())
};
```

همچنین متد «ReadFoods» نیز در آخر صدا زده می شود.

```
ReadFoods();
```

### متد Delete

ابتدا متد «DeleteFoods» را صدا می زنیم تا غذا های سفارش داده شده از جدول «Foods\_Order» حذف شوند سپس متد «Delete» والد را صدا می زنیم.

```
public override void Delete(){
    DeleteFoods();
}
```

```
base.Delete();
```

```
}
```

### متد Save

ابتدا متد «Save» والد را صدا می‌زنیم تا سفارش ثبت شود و سپس متد «SaveFoods» را صدا می‌زنیم تا غذا های سفارش داده شده در جدول «Foods\_Orders» درج شوند.

```
public override void Save(){
```

```
    base.Save();
```

```
    SaveFoods();
```

```
}
```

### پیاده‌سازی فرم‌ها

#### فرم frmLogin

#### متد InitializeLanguage

در زیر تکه کدی از کدی که درون این متد نوشته می‌شود آورده شده است همانطور که قابل مشاهده است مقدار مقادیر را از دیکشنری «Words» می‌خوانیم و به صفت «Text» کنترل‌ها نسبت می‌دهیم.

```
mnuAbout.Text = Common.Words["About"];
```

```
btnLogin.Text = Common.Words["Login"];
```

```
this.Text = Common.Words["Login"];
```

در اینجا راست به چپ کنترل‌ها را تغییر می‌دهیم.

```
Common.ToggleRightToLeft(flwLayoutLogin);
```

```
Common.ToggleRightToLeft(mnuStripLogin);
```

در فرم‌های دیگر نیز متدی با این نام وجود دارد و روشی مشابه پیاده‌سازی می‌شود.



**رویداد frmLogin\_Load**

این متد که متعلق به رویداد «Load» می باشد. در آن منوی «زبان» را پر می کنیم.

و یکبار متد «InitializeLanguage» را صدا می زنیم.

```
Common.PopulateLanguageMenu(mnuLanguage, delegate()
{
    InitializeLanguage();
});
InitializeLanguage();
```

در فرمهای دیگر نیز متدی با این نام وجود دارد و روشی مشابه پیاده سازی می شود.

**رویداد frmLogin\_FormClosed**

در تمام فرمها این رویداد این گونه پیاده سازی می شود.

متد «Exit» از کلاس «Application» را صدا می زنیم.

```
private void frmLogin_FormClosed(object sender, FormClosedEventArgs e) {
    Application.Exit();
}
```

**متد frmLogin**

```
public frmLogin() {
    InitializeComponent();
```

در اینجا تعداد دفعاتی که کاربر سعی داشته است به سیستم وارد شود ثبت می شود و اگر بیش از ۳ بار تکرار شود از سیستم خارج می شود.

```
Attempts = 1;
```

متد «Initialize» کلاس «Common» را فراخوانی می کنیم. و یک نمونه از «Employee» ساخته و به «EmployeeController» نسبت می دهیم.

```
Common.Initialize();

EmployeeController = new Employee();
```

اگر در سیستم هیچ کارمندی تعریف نشده باشد به طور پیشفرض یک کارمند با نام کاربری و رمز عبور «admin» ایجاد می شود.

```
if (EmployeeController.ReadAll().OfType<Employee>().Count() == 0){
    Employee admin = new Employee(){
        Id = Guid.NewGuid(),
        Username = "admin",
        Password = "admin",
        FirstName = "admin",
        LastName = "admin",
        Role = Role.SuperAdmin
    };
    admin.Save();
}
}
```

#### رویداد btnLogin\_Click

```
private void btnLogin_Click(object sender, EventArgs e) {
    EmployeeController.Username = txtUsername.Text;
    EmployeeController.Password = txtPassword.Text;
    if (EmployeeController.Authenticate() != Role.Unauthorized){
```

اگر کاربر معتبر باشد یک پیغام خوش آمد گویی نمایش داده شده و با توجه به نوع آن به یکی از پنل های مدیریت یا پنل اصلی هدایت می شود.

```

        if(EmployeeController.Role!=Role.SuperAdmin)

            MessageBox.Show(string.Format(Common.Words["authorized_message"],EmployeeController.FullName));

        else

            MessageBox.Show(string.Format(Common.Words["authorized_message"],Common.Words["SuperAdmin"]));

        Common.CurrentUser = EmployeeController;

        if (EmployeeController.Role == Role.Cashier){

```

اگر کاربر از نوع «صندوقدار» باشد به پنل اصلی هدایت می شود.

```

        new frmMain().Show();

        this.Hide();

    }

    else if (EmployeeController.Role == Role.Manager ||
EmployeeController.Role == Role.SuperAdmin){

```

اگر کاربر از نوع «مدیر» یا «مدیر کل» باشد به پنل مدیریت هدایت می شود.

```

        new frmManagement().Show();

        this.Hide();

    }

}

else{

```

پیغامی مبنی بر نامعتبر بودن نام کاربری یا پسورد نمایش می دهد.

```

        MessageBox.Show(Common.Words["not_authorized_message"]);

        if (attempts == 3){

```

اگر ۳ مرتبه نام کاربری و یا رمز عبور اشتباه باشد از سیستم خارج می شود.

```

        Application.Exit();
    }
    attempts++;
}
}

```

### فرم frmManagement

#### متد frmManagement

متغیر های FoodController, EmployeeController, OrderController, CategoryController با نمونه‌های متناسب مقدار دهی می شوند.

```
FoodController = new Food();
```

ادامه نیز به همی صورت است.

همچنین لیست های مربوطه را نیز مانند تکه کد زیر پر می کنیم.

```
Common.InitializeList(ref foods, ref FoodController);
```

در خانه اول لیست «دسته بندی» ها یک مقداری به نام «همه» همه اضافه می کنیم. که Id آن خالی می باشد.

```

categories.Insert(0, new Category() { CategoryName =
Common.Words["All"], Id = Guid.Empty });

InitializeStatistics();

InitializeLanguage();

```

با استفاده از متد InitializeFoodsDataGrid این دیتاگرید ویو را آماده می کنم. به ازای تمام دیتاگریدویو ها در سیستم چنین متدی نیز وجود دارد.

```

InitializeFoodsDataGrid();

InitializeEmployeesDataGrid();

```

### متد InitializeFoodsDataGrid

```
private void InitializeFoodsDataGrid(){
```

ابتدا مقدار «DataSource» را برابر با «LocalDataSource» از نمونه FoodController قرار می دهیم.

```
dgFood.AutoGenerateColumns = false;

dgFood.DataSource = FoodController.LocalDataSource;
```

سپس ستون‌ها را با استفاده از متد «AddDataGridColumn» به DataGridView اضافه می کنیم.

```
Common.AddDataGridColumn(dgFood, "FoodName");

Common.AddDataGridColumn(dgFood, "Price");

DataGridViewCheckBoxColumn columnEnabled = new
DataGridViewCheckBoxColumn() { Name = "Enabled", DataPropertyName = "Enabled" };

columnEnabled.Width = 60;

dgFood.Columns.Add(columnEnabled);
```

در اینجا ComboBox مربوط به دسته بندی ها را نیز پر می کنیم.

```
comboCategory.DataSource = categories;

comboCategory.DisplayMember = "CategoryName";

}
```

این متد برای بقیه DataGridView ها به روشی مشابه پیاده سازی می شود.

### متد HideSuperAdmin

در این متد اولین کارمند که همان کابر «مدیر کل» است را از لیست پاک می کنیم.

## متد InitializeEmployeeRole

```
public void InitializeEmployeeRole(){
    if (dgEmployee.Columns.Contains("Role"))
    {
```

در اینجا لیستی از سطوح دسترسی کارمندان ایجاد می‌کنیم که درون آن را با کلاس ناشناس پر می‌کنیم؛ که شامل ۲ صفت می‌باشد.

- Value: مقدار

- Text: متن

```
List<object> roles = new List<object>() {
new { Value = Role.Default, Text = Common.Words["Default"] } ,
new { Value = Role.Manager, Text = Common.Words["Manager"] } ,
new { Value = Role.Cashier, Text = Common.Words["Cashier"] } };
```

بعد صفات مختلف این ستون را مقدار دهی می‌کنیم.

```
(dgEmployee.Columns["Role"] as DataGridViewComboBoxColumn).DataSource = roles;
(dgEmployee.Columns["Role"] as DataGridViewComboBoxColumn).DisplayMember = "Text";
(dgEmployee.Columns["Role"] as DataGridViewComboBoxColumn).ValueMember = "Value";
(dgEmployee.Columns["Role"] as DataGridViewComboBoxColumn).DataPropertyName = "Role";
(dgEmployee.Columns["Role"] as DataGridViewComboBoxColumn).HeaderText =
Common.Words["Role"];
    }
}
```

برای دیگر ستون‌ها از نوع ComboBox نیز به روشی مشابه پیاده‌سازی می‌شوند.

## رویداد comboCategory\_SelectedIndexChanged

```
private void comboCategory_SelectedIndexChanged(object sender, EventArgs e){
    txtSearch.Text = string.Empty;
```

اگر دسته بندی «همه» انتخاب شده باشد یک سری اعمال غیر فعال می شوند.

```
if ((comboBoxCategory.SelectedValue as Category).Id.Equals(Guid.Empty)){
    dgFood.AllowUserToAddRows = false;
    tsBtnAdd.Enabled = false;
    tsBtnSave.Enabled = false;
}
else{
    dgFood.AllowUserToAddRows = true;
    tsBtnAdd.Enabled = true;
    tsBtnSave.Enabled = true;
}
```

غذا های موجود در دسته بندی جاری را می خواند.

```
dgFood.DataSource =
FoodController.ReadByCategory(comboBoxCategory.SelectedValue as Category);
}
```

### متد RebindFoodsDataGrid

یک بار دیگر غذا ها را خوانده و به DataGridView نسبت می دهد.

```
foods = new BindingList<Food>(FoodController.ReadAll().OfType<Food>().ToList());
if (currentCategory){
```

در دسته بندی جاری

```
dgFood.DataSource =
FoodController.ReadByCategory((comboBoxCategory.SelectedValue as Category));
}
else{
```

تمام دسته بندی ها

```
dgFood.DataSource = FoodController.LocalDataSource;
```

```
}
```

برای DataGridView های دیگر نیز چنین متدی وجود دارد که با روشی مشابه پیاده‌سازی می‌شود.

### عمل حذف

متد ShowDeletePrompt را صدا می‌زنیم و به عنوان ورودی DataGridView مورد نظر را می‌دهیم. سپس یکبار دیگه مقادیر را بازخوانی می‌کنیم.

```
Common.ShowDeletePrompt(dgEmployee);  
  
RebindEmployeesDataGrid();
```

این متد در پیاده‌سازی رویداد های UserDeletingRow از DataGridView و رویداد Click دکمه «حذف» نوشته می‌شود.

برای UserDeletingRow این خط نیز اضافه می‌شود.

```
e.Cancel = true;
```

### متد Search

در این متد نسبت به محل قرار گیری متد Search مربوط به نمونه مورد نظر را صدا می‌زنیم.

### متد DiscardChanges

در این متد نسبت به محل قرار گیری متد DiscardChanges را بر روی لیست مربوطه اجرا می‌کند.

### رویداد dgFood\_UserAddedRow

```
private void dgFood_UserAddedRow(object sender, DataGridViewRowEventArgs e) {  
  
    try{
```

یک نمونه جدید می‌سازیم.

```
Food newFood = foods.AddNew();
```

مقدار State آن را به Constructing تغییر می‌دهیم.

```
newFood.State = ModelState.Constructing;
```



دسته بندی جاری را به آن نسبت می دهیم.

```
newFood.Category = comboCategory.SelectedValue as Category;  
newFood.Enabled = true;  
dgFood.CurrentRow.Cells["Enabled"].Value = true;
```

آن را درون صفت «Tag» سطر جاری میریزیم. و در نهایت آیکون «ذخیره» را عوض می کنیم.

```
dgFood.CurrentRow.Tag = newFood;  
tsBtnSave.Image = Properties.Resources.changes_pending;  
}  
catch (Exception ex){  
  
    MessageBox.Show(ex.Message);  
  
}  
}
```

### متد SaveNewRows

```
private void SaveNewRows(){  
    if (IsActiveTab(tabFood)){
```

در اینجا آن مقادیری که با شرط آورده شده در کد برابرند را ذخیره می کنیم.

```
foreach (Food food in foods.Where((food) => food.State ==  
ModelState.Constructed || food.State == ModelState.Constructing)){  
    food.Save();  
}  
RebindFoodsDataGrid(true);  
}  
}
```

### متد SaveModifiedRows

با توجه به تب انتخاب شده کدی مانند تکه کد زیر می نویسیم.

```
Common.CommonDataGridSubmitEdit(foods);

RebindFoodsDataGrid(true);
```

### رویداد dgFood\_CellBeginEdit

یه طور کلی برای شروع ویرایش سطر از این کد استفاده می کنیم.

```
Common.CommonDataGridCellBeginEdit(sender as DataGridView, e);
```

### رویداد dgFood\_CellEndEdit

```
Common.ModelIsChanged += new EventHandler(Common_ModelIsChanged);

Common.ModelIsNotChanged += new EventHandler(Common_ModelIsNotChanged);
```

یک دلیگت با ورودی «Food» ایجاد می کنیم.

```
Common.CommonDataGridCellEndEdit(sender as DataGridView, e, delegate(Food food)

{
```

در اینجا صفات نمونه مورد نظر را مقدار دهی می کنیم.

```
if (!string.IsNullOrEmpty((string)dgFood.CurrentRow.Cells["FoodName"].Value))
food.FoodName = dgFood.CurrentRow.Cells["FoodName"].Value.ToString();

if (dgFood.CurrentRow.Cells["Price"].Value != null &&
Validator.IsFloat(dgFood.CurrentRow.Cells["Price"].Value.ToString()))

    food.Price = (float)dgFood.CurrentRow.Cells["Price"].Value;

    else Common.ShowErrorMessage("Invalid Input(Number)");

    food.Enabled = (bool)dgFood.CurrentRow.Cells["Enabled"].Value;

}

);
```

## آشنایی محیط نرم افزار و نحوه کار با آن

### صفحه ورود



در هربار اجرای برنامه این صفحه نمایش داده می شود که برای تأیید هویت کاربران است. اگر برای اولین بار برنامه را اجرا کنید به طور پیشفرض یک کاربر با نام کاربری و رمز عبور «admin» ساخته می شود.

شکل ۱۳: صفحه ورود

### پنل مدیریت

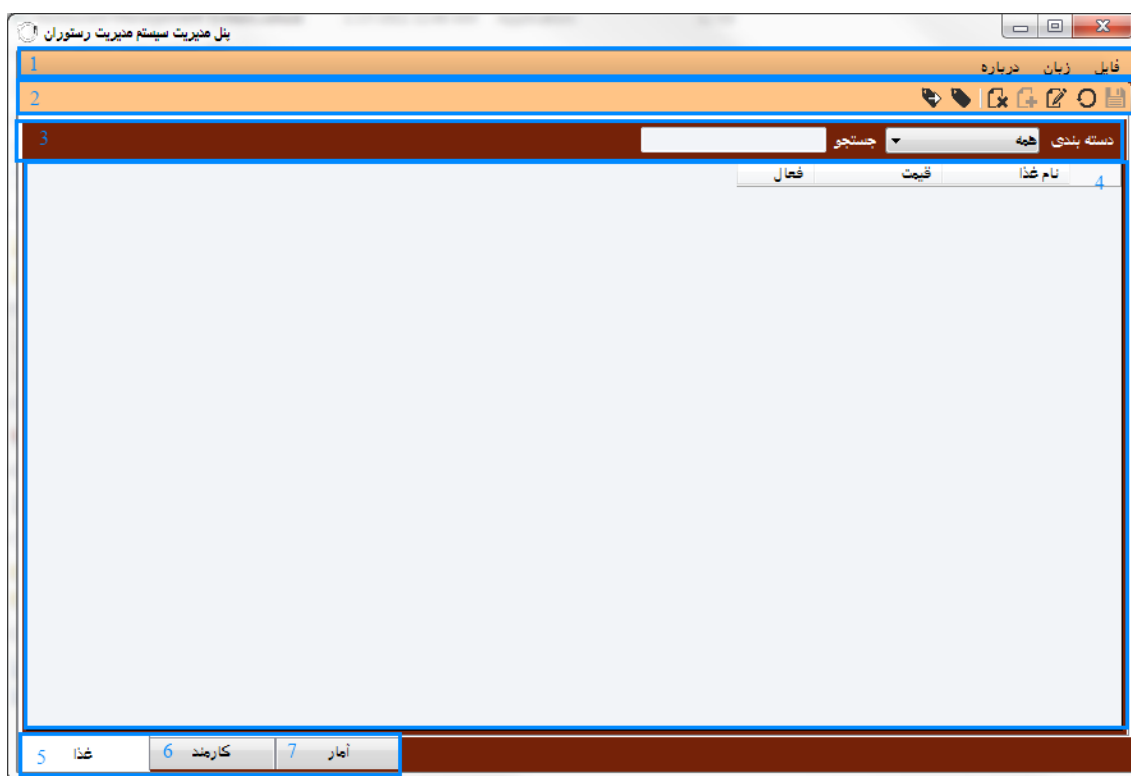
برای اولین بار به نام کاربری پیشفرض «admin» وارد سیستم بشویم به پنل مدیریت هدایت می شویم.

به طور کلی ۲ نوع کاربر در این سیستم وجود دارد.

- صندوقدار: این کاربر فقط به پنل مخصوص به سفارش ها دسترسی دارد.
- مدیر: این کاربر علاوه بر دسترسی به پنل سفارش ها به پنل مدیریت هم دسترسی دارد.

### امکانات پنل مدیریت

- امکان اضافه کردن، حذف، ویرایش، تغییر دسته بندی، و فعال و غیر فعال کردن غذا
- امکان اضافه کردن، حذف، ویرایش دسته بندی ها
- امکان اضافه کردن، حذف، ویرایش، تغییر سطح دسترسی کارکنان
- امکان آمارگیری



شکل ۱۴: پنل مدیریت

۱. **منوی اصلی:** این منو در تمام قسمت‌های نرم‌افزار وجود دارد و شامل سه منوی «فایل»، «زبان» و «درباره» می‌باشد.
۲. **تولبار اصلی:** این تولبار نیز در تمام قسمت‌های نرم‌افزار وجود دارد و برای کار با داده‌های نرم‌افزار از آن استفاده می‌شود.

### ۳. دسته بندی و جستجو

○ دسته بندی: برای نمایش غذا های موجود در یک دسته بندی خاص استفاده می شود.

○ جستجو: برای جستجو استفاده می شود.

۴. جدول مربوط به غذا

۵. تب غذا

۶. تب کارمند

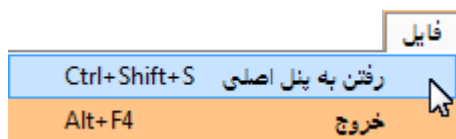
۷. تب آمار

### منوها

#### منوی اصلی

گزینه های این منو با توجه به قسمتی که کاربر در آن قرار دارد فرق می کند؛ در تمام صفحات گزینه «خروج» وجود دارد که با استفاده از آن کاربر از سیستم خارج می شود.

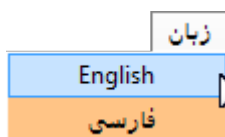
در پنل مدیریت هم گزینه ای با نام «رفتن به پنل اصلی» وجود دارد که همان پنل مربوط به سفارش ها است، با انتخاب این گزینه به آن پنل می رود. همچنین می توان برای این کار از کلید میانبر **Ctrl+Shift+S** نیز استفاده کرد.



شکل ۱۵: منوی فایل

در پنل اصلی هم گزینه ای مشابه با نام «رفتن به پنل مدیریت» وجود دارد که فقط برای کاربران «مدیر» قابل مشاهده می باشد.

#### منوی زبان



شکل ۱۶: منوی زبان

زبان

این منو زبان های موجود سیستم را نمایش می دهد که به طور پیش فرض دو زبان انگلیسی و فارسی در آن تعبیه شده است و از آنجایی که سیستم قابلیت

ترجمه شدن و اضافه کردن زبان‌های جدید را دارا می باشد؛ آن زبان‌ها در همین منو نمایش داده می شوند.

### منوی درباره

این منو کادری را نمایش می‌دهد که در آن اطلاعاتی در مورد سیستم نمایش داده می شود.

### تولبار اصلی



### دکمه های اصلی

این دکمه ها در تمام قسمت‌های سیستم وجود داشته و کاربردی یکسان دارند.

آیکون	کلید میانبر	توضیحات
	Ctrl+S	برای ذخیره غذا، کارمند، مشتری جدید استفاده می شود. و هنگامی که نیاز به ذخیره سازی باشد این آیکون به شکل  در می‌آید.
	F5	برای لغو تمام تغییرات استفاده می شود.
	Ctrl+E	برای ذخیره غذا، کارمند، یا مشتری ویرایش شده استفاده می شود. و هنگامی که لازم به ذخیره ویرایش ها باشد آیکون آن به شکل  در می‌آید.
	ندارد	مکان نما را به سطر جدید برای درج غذا، کارمند، یا مشتری جدید می برد.
	Delete	غذا ها، کارمند ها، و مشتری‌های انتخاب شده را حذف می کند.

## دکمه های خاص مربوط به تب غذا

این دکمه ها مخصوص تب غذا می باشد.

آیکون	کلید میانبر	توضیحات
	ندارد	برای مدیریت دسته بندی ها استفاده می شود. در آن می توان دسته بندی جدیدی را ایجاد کرد و یا دسته بندی را حذف یا ویرایش کرد.
	ندارد	برای تغییر دسته بندی غذا های انتخاب شده استفاده می شود.

## دسته بندی و جستجو

جستجو در تب غذا، امکان جستجو در بین غذا ها را می دهد.

جستجو در تب کارمند، امکان جستجو در بین کارمندان را می دهد.

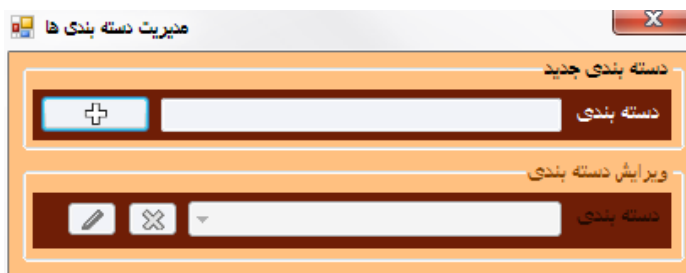
جستجو در تب مشتری (در پنل اصلی)، امکان جستجو در بین مشتریان را می دهد.

دسته بندی هم فقط مختص به غذا می باشد و این امکان را می دهد تا غذا ها را بر حسب دسته بندی مشاهده کنیم.


## تب «غذا»

### مدیریت دسته بندی ها

#### دسته بندی جدید

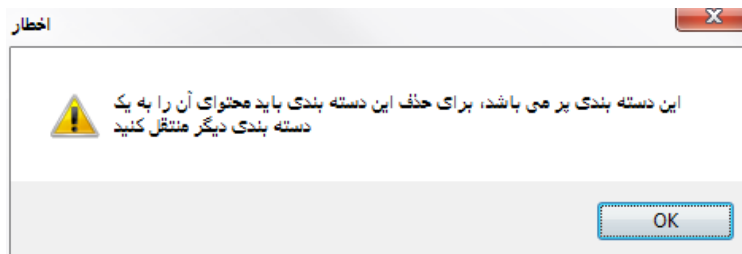



شکل ۱۷: مدیریت دسته بندی

برای اضافه کردن دسته بندی جدید کافی است تا نام دسته بندی را در کادر متن در قسمت «دسته بندی جدید» وارد کنیم و دکمه  را

انتخاب کنیم.

## حذف دسته بندی



برای این کار کافی است که پس از انتخاب دسته بندی مورد نظر بر روی دکمه  کلیک کنیم.


اگر دسته بندی خالی باشد

پیغامی برای تأیید عمل حذف

شکل ۱۸: پیغام مبنی بر پر بودن دسته بندی

نمایش داده می شود در غیر این صورت پیغامی مبنی بر پر بودن دسته بندی نمایش داده می شود و کاربر به کادر تغییر دسته بندی ارجاع داده می شود تا محتوای دسته بندی مورد نظر را به دسته بندی دیگری منتقل کند و دسته بندی مورد نظر را حذف کند؛ همچنین اگر فقط یک دسته بندی در سیستم وجود داشته باشد نمی توان آن را پاک کرد.

## ویرایش دسته بندی

برای ویرایش دسته بندی کافی است پس از انتخاب دسته بندی بر دکمه  کلیک کنید سپس پس از ویرایش دسته بندی دوباره این دکمه را کلیک کنید تا ویرایش اعمال شود.

## اضافه کردن غذا

نام غذا	قیمت	فعال
کیک شکلاتی	1500	<input checked="" type="checkbox"/>
ژله	700	<input checked="" type="checkbox"/>
کیک پنیر	2000	<input checked="" type="checkbox"/>
بستنی میوه ای	1000	<input checked="" type="checkbox"/>
بستنی شکلاتی	1000	<input checked="" type="checkbox"/>

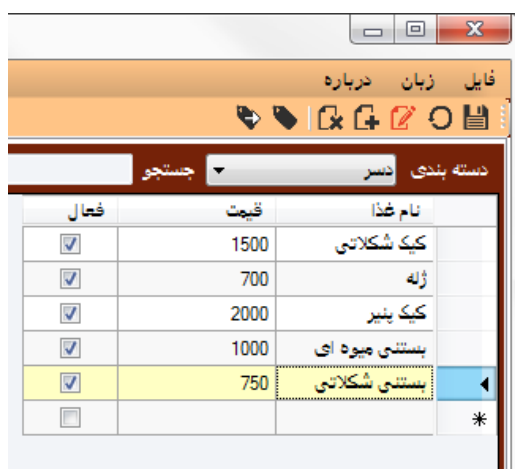
برای اضافه کردن غذا کافی است تا ابتدا یک دسته بندی ایجاد کنیم؛ پس از ایجاد دسته بندی و خروج از کادر «مدیریت دسته بندی ها» دسته بندی جدید به طور خودکار انتخاب می شود و می توانیم غذا های مورد نظر را به آن اضافه کنیم. برای این کار کافی است که در سطر جدید ایجاد شده مشخصات غذا که شامل «نام»، «قیمت» و «فعال» است را وارد کنیم.

شکل ۱۹: غذای جدید



گزینه فعال، تعیین می کند که آیا این غذا در منو نمایش داده شود یا خیر از آنجایی که سیستم اجازه نمی دهد غذاهایی که سفارش داده شده اند حذف شوند از این گزینه برای غیر فعال کردن آن ها استفاده می شود. همان طور که در شکل مشاهده می شود پس از وارد کردن مقادیر به سیستم آیکون «ذخیره» از رنگ مشکی به رنگ قرمز تغییر می کند.

### ویرایش غذا

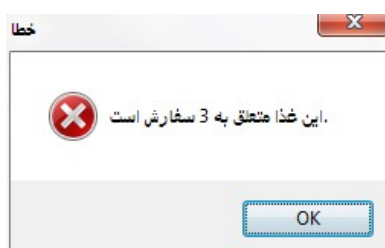


نام غذا	قیمت	فعال
کیک شکلاتی	1500	<input checked="" type="checkbox"/>
ژله	700	<input checked="" type="checkbox"/>
کیک پنیر	2000	<input checked="" type="checkbox"/>
بستنی میوه ای	1000	<input checked="" type="checkbox"/>
بستنی شکلاتی	750	<input checked="" type="checkbox"/>
		<input type="checkbox"/>


برای ویرایش غذا کافی است اطلاعات غذای مورد نظر را در جدول تغییر دهیم. همان طور که در شکل مشاهده می شود پس از تغییر قیمت «بستی شکلاتی» به «۷۵۰» آیکون مربوط به «ویرایش» به رنگ قرمز در آمد؛ پس از کلیک بر روی این کلید می توان تغییرات را ذخیره کرد.

شکل ۲۰: ویرایش غذا

### حذف غذا

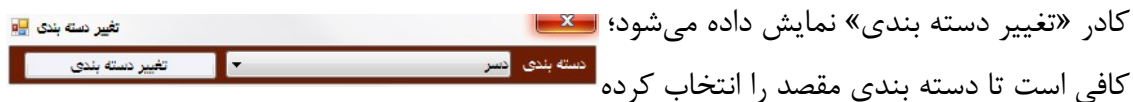


شکل ۲۱: غذای سفارش داده شده

برای اینکار کافی است تا غذای مورد نظر را انتخاب کنیم و بر روی دکمه  کلیک کنیم، پس از تأیید غذای مورد نظر حذف می شود. اگر این غذا قبلاً سفارش داده شده باشد پیغامی مانند این تصویر نمایش داده می شود. که می توان آن غذا را بجای حذف کردن غیرفعال کرد تا در منو نمایش داده شود.

## تغییر دسته بندی

برای تغییر دسته بندی کافی است غذا مورد نظر را انتخاب کرده و بر روی دکمه ➡ کلیک کنید. بعد



شکل ۲۲: تغییر دسته بندی

و بر روی دکمه «تغییر دسته بندی» کلیک کنید.

## دسته بندی و جستجو

## دسته بندی

دسته بندی	سبزی	جستجو
نام غذا	قیمت	فعال
قرمه سبزی	1500	<input checked="" type="checkbox"/>
خورش قیبه	1500	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

شکل ۲۳: نمایش بر حسب دسته بندی

برای نمایش غذاها بر حسب دسته بندی کافی است تا دسته بندی مورد نظر را انتخاب کنید که غذا های موجود در آن دسته بندی

نمایش داده شود. اگر دسته بندی «همه» انتخاب شود تمام غذا ها نمایش داده می شود.

## جستجو

برای جستجو کافی است تا در کادر مربوط به جستجو متن مورد نظر را تایپ کنیم تا نتایج متناسب با

دسته بندی	همه	جستجو
نام غذا	قیمت	فعال
کیک شکلاتی	1500	<input checked="" type="checkbox"/>
بستنی شکلاتی	750	<input checked="" type="checkbox"/>
خورش قیبه	1500	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

شکل ۲۴: جستجو

متن وارد شده نمایش داده شود. اگر

هنگام جستجو دسته بندی «همه»

انتخاب شده باشد در تمام دسته بندی ها

به دنبال آن غذا می گردد در غیر این

صورت فقط در دسته بندی انتخاب

شده دنبال آن غذا می گردند.

## تب «کارمند»

این تب مخصوص مدیریت کارمندان است. در این قسمت می‌توان کارمندی جدید اضافه کرد، کارمندان کنونی را ویرایش کرد، و یا آن‌ها را حذف کرد. همچنین می‌توان سطح دسترسی کارمندان را تغییر داد.

## اضافه کردن «کارمند»

برای این کار مانند تب «غذا» عمل می‌کنیم به این ترتیب که در سطر جدید مشخصات کارمند جدید را وارد می‌کنیم. و پس از اتمام کار، ذخیره می‌کنیم.

همان‌طور که قبلاً بیان شد دو

نوع کارمند در سیستم وجود

دارد «صندوقدار» و «مدیر» که

به طور پیشفرض کارمندان

«صندوقدار» در نظر گرفته می

شوند.

نام	نام خانوادگی	نام کاربری	رمز عبور	سطح دسترسی
عباس	اللهیاری	abbas	...fEqNCoo3Yq9h5	مدیر
رضا	احمدی پور	reza	...PU8r8H3BvjyDN	صندوق دار
علی	اکبری	ali	...fEqNCoo3Yq9h5	صندوق دار
صادق	احمدی	sadeq	123456	پیشفرض
*				

شکل ۲۵: کارمند جدید

ویرایش و حذف هم مانند تب

«غذا» می‌باشد.

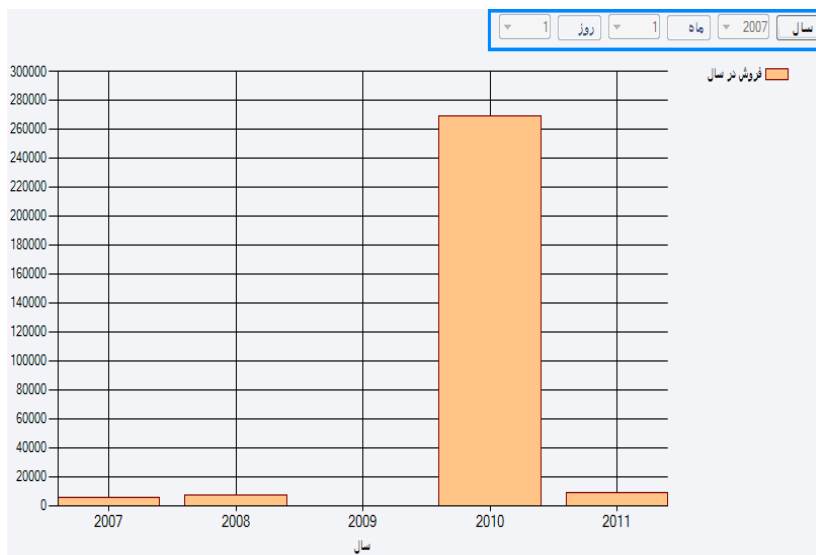
## جستجو

در این تب جستجو بر حسب «نام» یا «نام خانوادگی» کارمندان انجام می‌شود.

## آمار

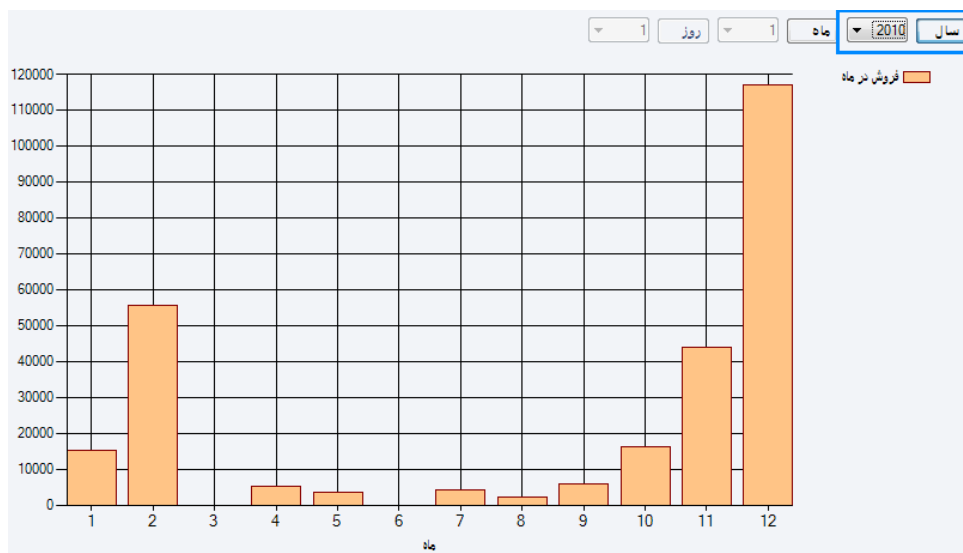
این قسمت نمودار فروش سالانه، ماهانه، روزانه، و ساعتی را نشان می‌دهد.

طرز استفاده از این امکان به این صورت است که با توجه به وضعیت گزینه های «سال»، «ماه»، و «روز» نمودار متناسب را نمایش می دهد.



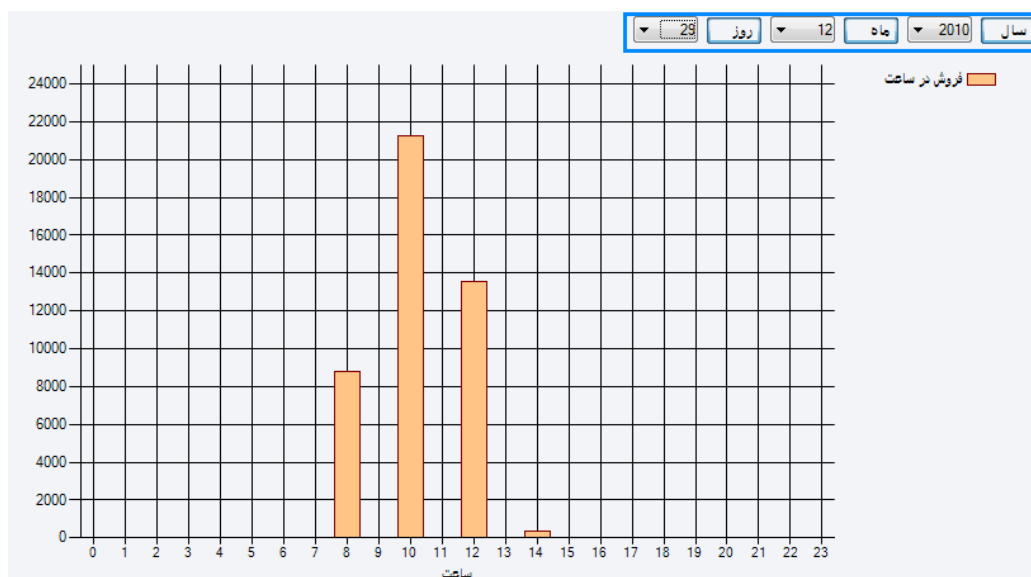
شکل ۲۶: نمودار فروش سالانه

اگر گزینه «سال» انتخاب شده باشد فروش ماهانه در سال انتخاب شده را نشان می دهد.



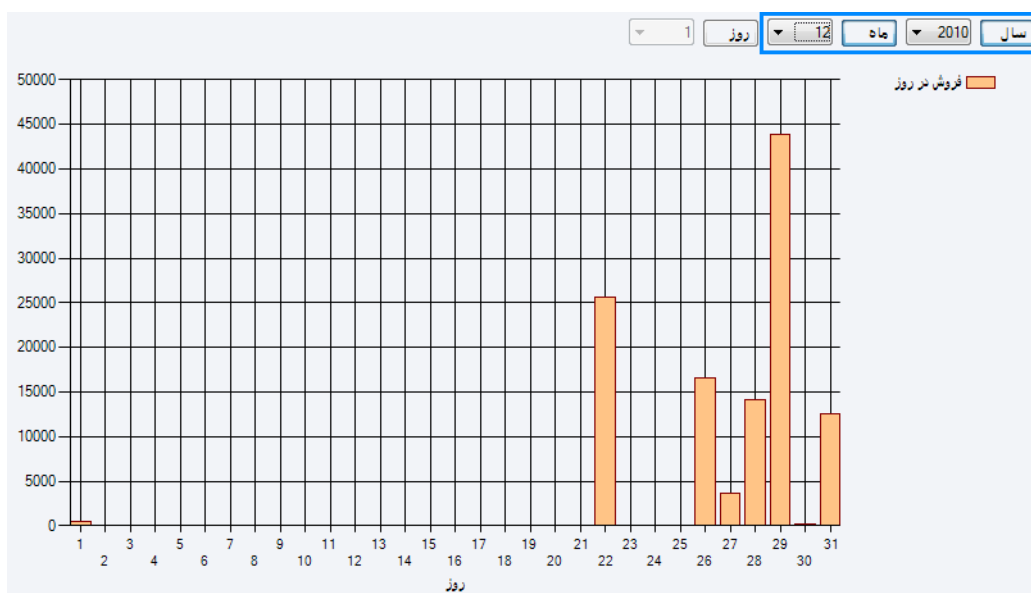
شکل ۲۷: فروش ماهانه

اگر گزینه «ماه» نیز انتخاب شود فروش روزانه در آن ماه را نشان می دهد.



شکل ۲۸: فروش ساعتی

و اگر گزینه «روز» انتخاب شده باشد فروش ساعتی در آن روز به نمایش در می آید.



شکل ۲۹: فروش روزانه

### پنل اصلی یا سفارش

این پنل برای کاربران «صندوقدار» به نمایش در می‌آید و به آنها امکان می‌دهد برای ثبت سفارش و تعریف مشتری می‌دهد؛ این پنل توسط کاربران «مدیر» نیز قابل دسترسی است.

### امکانات پنل اصلی

- اضافه، حذف، ویرایش مشتری
- ثبت، مشاهده، حذف، و ویرایش سفارش‌ها
- گزارش گیری



شماره اشتراک	نام	نام خانوادگی	شماره تلفن	آدرس
*				

شکل ۳۰: پنل اصلی

۱. **منوی اصلی:** مانند منوی اصلی در پنل مدیریت است با این تفاوت که اگر کاربر «مدیر» باشد در منوی «فایل» گزینه ای با نام «رفتن به پنل مدیریت» دیده می‌شود.
۲. **تولبار اصلی:** دو دکمه جدید دارد که برای ثبت سفارش، و گزارش گیری هستند.

۳. **نوار جستجو:** بر حسب «نام»، «نام خانوادگی»، «شماره اشتراک» و «شماره تلفن» در بین مشتریان جستجو می کند.

#### تولبار اصلی

آیکون	کلید میانبر	توضیحات
	ندارد	با کلیک بر روی این دکمه پنل را برا ثبت سفارش جدید آماده می کند.
	دابل کلیک بر روی سفارش	با کلیک بر روی این دکمه گزارشی از سفارش انتخاب شده نمایش می دهد که قابل چاپ نیز می باشد.

#### مدیریت مشتری ها

##### اضافه کردن «مشتری»


روند اضافه کردن مشتری نیز مانند اضافه کردن کارمند و غذا می باشد کافی است تا اطلاعات مشتری را وارد کنید و تغییرات را ذخیره کنید.

خود سیستم به طور خودکار شماره اشتراکی به مشتری نسبت می دهد که قابل تغییر نمی باشد.

جستجو				
شماره اشتراک	نام	نام خانوادگی	شماره تلفن	آدرس
1016	سارا	تهرانی	65453212	تهران - پلاک 3
21036	مجید	شایسته	22663344	تهران - شهرک غرب - پلاک 56
3082	احمد	عباسی	45654512	تهران - پاسداران - پلاک 23
229101				

شکل ۳۱: مشتری جدید

#### حذف و ویرایش مشتری


برای حذف مشتری کافی است تا دکمه  را کلیک کنید تا مشتری مورد نظر حذف شود. وقتی یک

مشتری حذف می شود تمام سفارش ها آن مشتری نیز حذف می شود.

ویرایش مشتری هم مانند ویرایش «غذا» و «کارمند» است.

## مدیریت سفارش ها

### ثبت سفارش

برای ثبت سفارش ابتدا باید مشتری مورد نظر را انتخاب کرده سپس بر روی دکمه  کلیک کنید، و شکل پنل برای ثبت سفارش تغییر می کند. در ابتدا اجزا را شرح می دهیم.





### شرح اجزا

شکل ۳۲: سفارش جدید


۱. کارکرد آن مانند دسته بندی و جستجو در تب «غذا» می باشد.
۲. لیست غذا های موجود را نمایش می دهد.
۳. نوار ابزار سفارش
۴. غذا های انتخاب شده
۵. نوع سفارش
۶. وضعیت
۷. هزینه پیک
۸. تخفیف به درصد
۹. ثبت یا لغو سفارش

توضیحات	کلید میانبر	آیکون
---------	-------------	-------



	ندارد	از لیست غذا ها، غذای انتخاب شده را به لیست اضافه می کند؛ همچنین اگر غذا قبلاً به لیست اضافه شده باشد کار افزایش «تعداد» غذا را انجام می دهد.
	Delete	غذای انتخاب شده را حذف می کند.
	ندارد	«تعداد» غذای سفارش داده شده را افزایش می دهد.
	ندارد	«تعداد» غذای سفارش داده شده را کاهش می دهد.

### نحوه ثبت سفارش جدید

بعد از تغییر شکل پنل می توان غذا ها را با کلیک راست بر روی نام آن ها و انتخاب «اضافه» از منو باز شده انتخاب کرد یا با استفاده از دکمه  این کار

را انجام بدهید.

مشتری		موجود شایسته
شماره اشتراک	21036	
آدرس	تهران - شهرک غرب - پلاک 56	
شماره تلفن	22663344	
تاریخ سفارش	1/28/2011 21:05:53 PM	
نوع سفارش	بیرون	

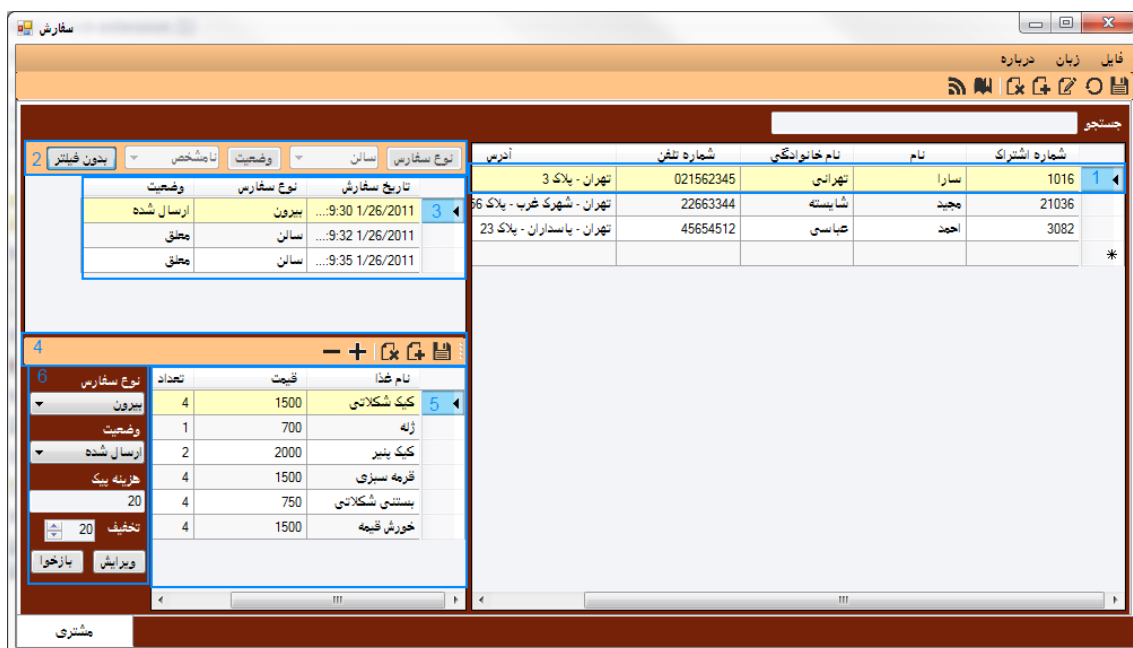
تعداد	قیمت	نام غذا
1	700	ژله
3	2000	کیک پنیر
4	1500	قرمه سبزی
1	1500	خورش قیمه
هزینه پیک		1,000.00
قیمت کل		15,200.00
تخفیف		760.00 (5%)
قیمت نهایی		14,440.00

سپس «نوع سفارش»، «وضعیت»، «هزینه پیک»، «تخفیف» را معین می کنیم و در نهایت بر روی دکمه «ثبت» کلیک می کنیم تا سفارش ثبت شود؛ پس از ثبت سفارش یک گزارش از سفارش نمایش داده می شود.

شکل ۳۳: گزارش سفارش

## مشاهده سفارش

برای مشاهده سفارش های ثبت شده کافی است تا بر روی مشتری مورد نظر کلیک کنید تا لیستی از سفارش های آن مشتری نمایش داده شود. همچنین با انتخاب هر سفارش می توان لیستی از غذا های سفارش داده شده را در پایین صفحه مشاهده کرد.



شکل ۳۴: مشاهده سفارش

۱. مشتری انتخاب شده
۲. فیلتر کردن سفارش های مشتری که به طور پیش فرض حالت «بدون فیلتر» فعال است.
۳. سفارش های مشتری
۴. نوار ابزار مربوط به ویرایش سفارش
۵. غذا های سفارش داده شده
۶. اطلاعات دیگر سفارش

## فیلتر کردن سفارش‌ها

نوع سفارش	سالن	وضعیت	نام شخص	بدون فیلتر
تاریخ سفارش	نوع سفارش	وضعیت		
....9:30 1/26/2011	بیرون	ارسال شده		
....9:32 1/26/2011	سالن	معلق		
....9:35 1/26/2011	سالن	معلق		
....8:21 1/28/2011	بیرون	معلق		

شکل ۳۵: سفارش «بدون فیلتر»

برای فیلتر کردن سفارش‌ها پس از انتخاب مشتری و مشاهده سفارش‌ها کافی است تا گزینه «بدون فیلتر» را غیرفعال کنید با این کار دو گزینه «نوع سفارش» و «وضعیت» فعال می‌شوند که

با فعال کردن هریک از آن‌ها سفارش بر حسب مقدار آن فیلتر می‌شود. همانطور که در شکل اول دیده می‌شود. گزینه «بدون فیلتر» انتخاب شده است در نتیجه تمام سفارش‌ها نمایش داده می‌شوند.

## فیلتر بر حسب «نوع سفارش»

نوع سفارش	بیرون	وضعیت	ارسال شده	بدون فیلتر
تاریخ سفارش	نوع سفارش	وضعیت		
....9:30 1/26/2011	بیرون	ارسال شده		
....8:21 1/28/2011	بیرون	معلق		

شکل ۳۶: سفارش‌های «بیرون»

برای نمایش سفارشاتی که فقط به بیرون ارسال شده است کافی است که ابتدا گزینه «بدون فیلتر» را غیر فعال کنیم سپس پس از انتخاب گزینه «نوع سفارش»، «بیرون» را انتخاب کنیم تا

فقط سفارش‌هایی که به بیرون فرستاده شده‌اند نمایش داده شوند.

## فیلتر بر حسب «وضعیت»

برای اینکه فقط سفارش‌های «معلق» را نمایش بدیم ابتدا باید گزینه «نوع سفارش» را غیر فعال

نوع سفارش	بیرون	وضعیت	معلق	بدون فیلتر
تاریخ سفارش	نوع سفارش	وضعیت		
....9:32 1/26/2011	سالن	معلق		
....9:35 1/26/2011	سالن	معلق		
....8:21 1/28/2011	بیرون	معلق		

شکل ۳۷: سفارش‌های «معلق»

کنیم و پس از فعال سازی گزینه «وضعیت»، «معلق» را انتخاب کنیم

تا فقط سفارش های معلق نمایش داده شوند، این کار برای دیگر سفارش ها با وضعیت های دیگر نیز قابل انجام است.

### فیلتر بر حسب «وضعیت» و «نوع سفارش»

همچنین می توان سفارش ها

تاریخ سفارش	نوع سفارش	وضعیت
1/26/2011 9:30...	بیرون	ارسال شده

هم بر حسب «نوع سفارش» و

هم «وضعیت» فیلتر کرد؛

برای این کار کافی است تا هر

دو گزینه را فعال کنیم و

مقدار مورد نظر را انتخاب

کنیم؛ همانطور که در شکل مشاهده می شود فقط سفارش هایی که «نوع سفارش» آن ها «بیرون» و «وضعیت» آن ها «ارسال شده» می باشد نمایش داده می شوند.

### حذف سفارش

برای اینکار پس از انتخاب سفارش مورد نظر کافی است تا کلید «Delete» را فشار بدهید و پس از تایید عمل حذف سفارش را حذف کنید.


### ویرایش سفارش

ویرایش کردن سفارش نسبت به بقیه کمی متفاوت تر است.

### تغییر تعداد غذا

برای اضافه، یا کم کردن «تعداد» غذا کافی است تا غذای مورد نظر را انتخاب کرده و از دکمه **+** برای افزایش تعداد یا دکمه **-** برای کاهش تعداد غذا استفاده کنید. تغییرات بلافاصله اعمال می شوند.

### حذف غذا از سفارش



همچنین برای یک غذا از لیست می‌توان از پس از انتخاب از دکمه  یا فشردن کلید «Delete» برای حذف غذا از لیست اقدام کرد. که برای این عمل هیچ تاییدی لازم نمی‌باشد و مانند حالت قبل بلافاصله تغییرات اعمال می‌شوند.

### اضافه کردن غذا به سفارش

دسته بندی همه		
نام غذا	قیمت	
کیک شکلاتی	1500	
ژله	700	
کیک پنیر	2000	
قرمه سبزی	1500	
بستنی شکلاتی	750	

نام غذا	قیمت	تعداد
ژله	700	3
کیک پنیر	2000	4
قرمه سبزی	1500	2
خورش قیمه	1500	1
کیک شکلاتی	1500	1

برای اضافه کردن غذای جدید به لیست سفارش‌ها کافی است یک بار دکمه  را کلیک کنید، با اولین کلیک لیست غذاها آورده می‌شود و کارکرد دکمه  به حالت قبل یعنی اضافه کردن غذا به لیست سفارش‌ها تغییر می‌کند. پس از انجام تغییرات با کلیک بر روی دکمه مربوط به ذخیره تغییرات را ذخیره می‌کنیم. همان‌طور که در شکل مشاهده می‌شود تعداد غذاها در سفارش «مجید شایسته» تغییر کرده است و «کیک شکلاتی» نیز به لیست اضافه شده است.

شکل ۳۹: اضافه کردن غذای جدید به سفارش

- +   			
نوع سفارش	تعداد	قیمت	نام غذا
بیرون	1	1500	کیک شکلاتی
وضعیت	3	700	ژله
تحویل داده شده	4	2000	کیک پنیر
هزینه بیک	2	1500	قرمه سبزی
1000	1	1500	خورش قیمه

تخفیف 7      

بازخوا  

### تغییر دیگر مشخصات سفارش

برای اینکار کافی است که مقادیری که در بخش مشخص شده در تصویر را تغییر دهیم و پس از تغییرات بر روی «ویرایش» کلیک کنید تا تغییرات اعمال شوند.

شکل ۴۰: تغییر دیگر مشخصات سفارش

### گزارش گیری

برای گزارش گیری کافی است تا بر روی سفارش مورد نظر دوبار کلیک کنیم تا گزارش سفارش انتخاب شده در پنجره ای جدید نمایش داده شود. که نمونه‌اش در بخش ثبت سفارش نمایش داده شده است.

## ترجمه نرم افزار

از آنجایی که قالب فایل های زبان به صورت متنی می باشد با استفاده از یک ویرایشگر متن ساده مانند «Notepad» برای ویرایش این فایل ها استفاده کرد.

Name	Date modified	Type	Size
en	1/28/2011 2:36 PM	Configuration sett...	3 KB
fa	1/28/2011 2:36 PM	Configuration sett...	4 KB
sv-SE	1/28/2011 11:59 PM	Configuration sett...	3 KB

برای اینکار ابتدا از فایل

زبان دلخواه کپی می

گرفته و نام آن را بر طبق

جدول کد های زبان

تغییر می دهیم.

سپس آن فایل را با یک ویرایشگر متن باز می کنیم. و

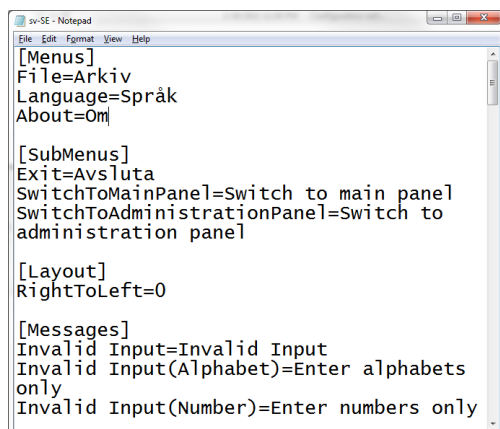
ترجمه متن های مورد نظر را در طرف راست «=» قرار

می دهیم. پس از اتمام کار فایل را ذخیره می کنیم.

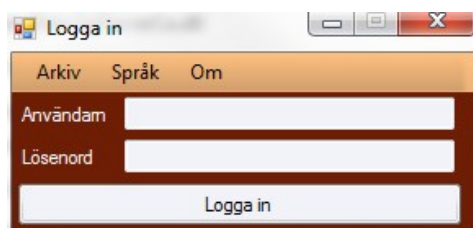
پس از اجرای نرم افزار زبان جدید که زبان سوئدی است

به منوی «زبان» اضافه می شود.

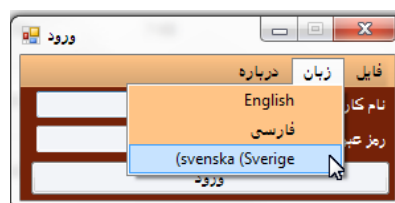
شکل ۴۱: فایل های زبان



شکل ۴۲: فایل زبان در Notepad



شکل ۴۴: صفحه ورود به سوئدی



شکل ۴۳: زبان جدید

پس از انتخاب زبان، همانطور که مشاهده می شود نرم افزار به زبان دلخواه ترجمه شده است.