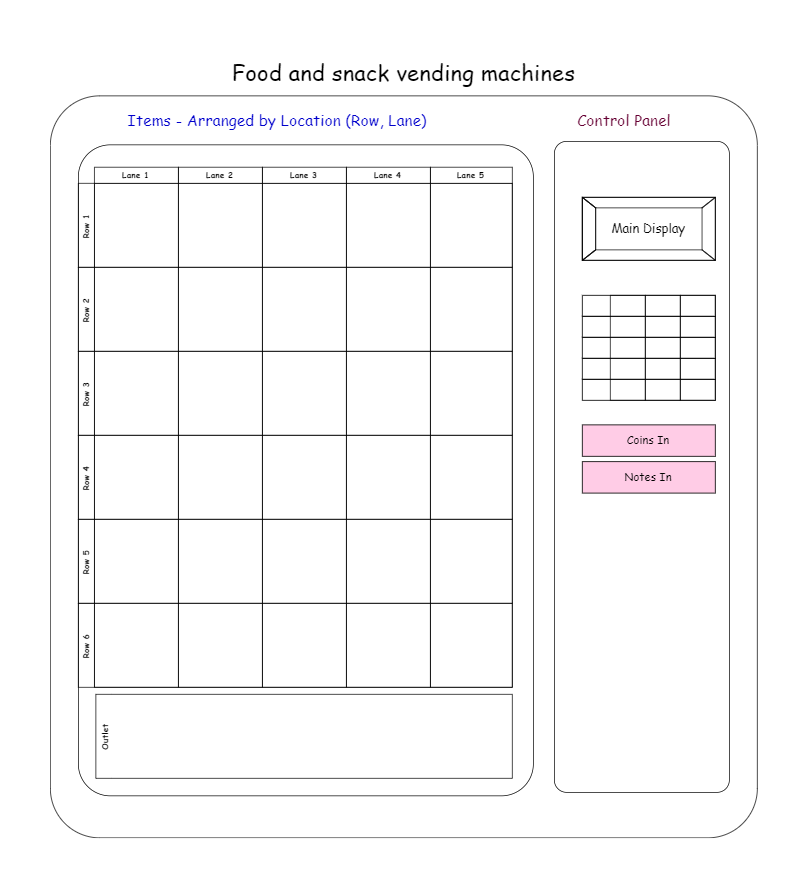
**Vending Machine Software**

**Inspired by DDD (Domain Driven Design)**





# **Vending Machine Subdomain**

1. **VM Specs:**
   1. VM has location property (to identify) along with ID
   2. VM has model, manufacturer properties
   3. VM has maintenance schedule (frequency, last maint date)
   4. VM has balances:
      1. Grand balance (non-reset amount)
      2. Customer session balance (transient balance)
      3. Inside Amount - > has details of coins/notes
   5. Operation start date
   6. End of life date
   7. VM inventory
      1. Is a map of each cell and item within it with Qty
   8. All VM has standard design (physical):
      1. Sell Items in 6 3 rows and 5 3 lanes (cells)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 |
|  |  | **Lane 1** | **Lane 2** | **Lane 3** | **Lane 4** | **Lane 5** |
| A | **Row A** | A1 | A2 | A3 | A4 | A5 |
| B | **Row B** | B1 | B2 | B3 | B4 | B5 |
| C | **Row C** | C1 | C2 | C3 | C4 | C5 |
| D | **Row D** | D1 | D2 | D3 | D4 | D5 |
| E | **Row E** | E1 | E2 | E3 | E4 | E5 |
| F | **Row F** | F1 | F2 | F3 | F4 | F5 |

* + 1. Each call has variable capacity based on sold item size (track Qty property)
    2. Accept coins of 1 Cent, 5 cent (Nickel), 10 cent (Dime), 25 cents (Quarter)
    3. Accept Notes of 1 $, 5 $, 10 $ & 20 $ dollar
    4. For future use (it may accept CC)!
    5. Sold items are food and snakes
    6. Each cell is allowed to contain single item type

1. **Sell item Specs:**
   1. Name
   2. Price
   3. Barcode (to link later with inventory system) / Product line
   4. Qty in each cell → part of VM inventory
   5. Type (food, snake) for future items expansion and stats
   6. Grand total (non reset counter)
   7. Grand sell for all times
2. **Use Case 1: Customer/User Purchase Process**
   1. Start new session
   2. Allowable Operations for customer:
      1. Purchase Item(s)/Cart ⇒ Goto 3.2
      2. Cancel customer session ⇒ Goto 3.4
   3. Insert coins and notes
      1. If (Acceptable)
         1. Add to session/transient balance
         2. Show total balance on front display
      2. Else
         1. Return it back to user
   4. Either insert more (Goto 3.1) or select item to purchase
      1. Enter Cell value (from touch screen or button matrix)
         1. If (cell value is correct)
            1. If Confirmed
            2. Added to item to cart (update Qty and total price)
            3. Validate against session balance
            4. If more than balance

Cancel last selection or ask for more coins/notes

* + - 1. Else
         1. Clear and ask to reselect
    1. Confirm Purchase Process
       1. Check balance is <= total cart amount
       2. if (! enough change to return after commit purchase)
       3. Cancel process and return all balance
       4. Always return highest possible notes or coins
    2. Else
       1. Commit purchase process
       2. Return change with highest possible notes and/or coins
       3. Reset cart line items
       4. Update items inventory in VM

Update each cell Qty

Update each item grand sell

Update each item grand total

* + - 1. Update VM balance
         1. Inside amount += cart\_amount
         2. Grand balance += cart\_amount
      2. Reset Display to state\_0
  1. Customer can select to cancel session at any time
     1. Return all balance back
     2. Reset Display to state\_0

1. **Use case 2: VM Operator:**
   1. Enter Login Credentials
   2. If (login is correct)
      1. VM state = OnHold (Inventory update)
   3. Allowable Operations:
      1. Takeout balance ⇒ Goto 4.4
      2. Refill (Update Inventory) ⇒ Goto 4.5
   4. Takeout Balance
      1. Inside Balance = 0
   5. Update Inventory
      1. Only VM inventory is updated
      2. For each cell -> confirm item -> confirm item price -> confirm Qty
2. **Use case 3: VM Maintenance:**
   1. Enter Login Credentials
   2. If (login is correct)
      1. VM state = OnHold (Maintenance)
   3. Allowable Operations:
      1. Start maint session/state
      2. Complete maint session/state
      3. Fix defect VM
   4. Start maint session
   5. Confirm completion
      1. Update last maint date
   6. If VM detected defect
      1. VM state = OnHold (Failure)
      2. Send notification to HQ/maint team
      3. Lock down mode
3. **Domain Models:**
   1. VendingMachine

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| ID | int | ToString() |
| Location | Location | GetLocation(stringLoc) |
| Model | string | UpdateLocation(stringLoc) |
| Manufacturer | string | GetNextMaintenanceDate() |
| MaintenanceSchedule | MaintenanceSchedule | UpdateMaintenanceLastDate() |
| GrandBalance | Money | GetGrandBalance() |
| CustomerBalance | Money | AddToGrandBalance(money) |
| InsideBalance | Balance | GetInsideBalance() |
| StartDate | Date | AddToInsideBalance(money) |
| EndOfLifeDate | Date | GetStartDate() |
| Cells | Cell List | GetEndOfLifeDate() |
| State | enum | UpdateCell(CellObject) |
|  |  | Cell GetCellDetails() |
|  |  | GetState() |
|  |  | ResetInsideBalance() |
|  |  | RefillItems(cellsCollection) |

* 1. Users

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| Id | int |  |
| UserName | string |  |
| UserPassword | string |  |
| UserPrivilege | enum |  |

1. **Value Objects:**
   1. **Location**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| Street | string |  |
| City | string |  |
| Landmark | string |  |

* 1. **MaintenanceSchedule**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| Frequency | Int (days) | Date GetNextMaintDate() |
| LastMaintDate | Date |  |

* 1. **Date**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| DateTime | DateTime | DateTimeSpan(Date) |
|  |  | +/-(Date1, Date2) |
|  |  | +/-(days) |
|  |  | +/-(Months) |
|  |  | +/-(Years) |

* 1. **Money**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| Amount | Decimal | ConvertToCurrency(rate) |
| Currency | string | ToString() |
|  |  | +/-(money) |
|  |  | String AmoneyField() |
|  |  | Money MoneyFactory(Decimal amount) |
|  |  | Money MoneyFactory(string moneyField) |

* 1. **Balance**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| CentCount | Int (1 cent) | Money CalculateAmount() |
| NickelCount | int (5 cent) |  |
| DimeCount | int (10 cent) |  |
| QuarterCount | int (25 cent) |  |
| DollarCount | Int (1 $) |  |
| FiveDollarCount | Int (5 $) |  |
| TenDollarCount | Int (10 $) |  |
| TwentyDollarCount | Int (20 $) |  |

* 1. **CartItem**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| Qty | Int |  |
| SellItem | SellItem |  |

* 1. **SellItem**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| Id | int | ToString() |
| Name | string | UpdateItemPrice() |
| Price | Money | UpdateItemBarcode() |
| Barcode | string | UpdateItemGrandTotal() |
| Type | Enum | UpdateItemGrandSell() |
| GrandTotal | int |  |
| GrandSellAmount | money |  |

* 1. **Cell**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| CellId | string |  |
| ~~RowNumber~~ | ~~Int (1-6)(A-F)~~ | ~~ToString() ⇒ D3, E5~~ |
| ~~LaneNumber~~ | ~~Int (1-5)~~ | ~~CreateFromLoc(string)~~ |
| SellItem | SellItem | UpdateItemQty(cellLoc) |
| SellItemQty | Int | UpdateItemPrice(cellLoc) |

* 1. **CustomerSession**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| CustomerBalance | Balance | Initialize() |
| CustomerCart | CartItems Collection | CancelSession() |
| Status | Enum | CommitPurchase() |
|  |  | CompleteSession() |

* 1. **~~Cart~~**

|  |  |  |
| --- | --- | --- |
| **~~Props~~** | **~~Primitive Type~~** | **~~Behaviors~~** |
| ~~CartItems~~ | ~~CartItems Collection~~ | ~~Money GetCartPrice()~~ |
|  |  | ~~Void AddLineItem(sellitem)~~ |
|  |  | ~~Void RemoveLineItem(sellitem)~~ |

1. **Domain Services**
   1. **InsertCoin()**
   2. **ValidateInsertedCoin()**
   3. **InsertNote()**
   4. **ValidateInsertedNote()**
   5. **CommitPurchase()**
   6. **ReturnChange(amountToReturn)**
   7. **CollectItems()**
   8. **DeclareDefectState()**
   9. **StarHealthDiagnosis()**
2. **Domain Events**
   1. **CoinInserted(CoinAmount)**
   2. **NoteInserted(NoteAmount)**
   3. **PurchaseCommitted()**
   4. **CustomerSessionCancelled()**
   5. **CustomerSessionCompleted()**
   6. **StateChangedToDefect()**
   7. **StateChangedToHold()**
   8. **StateChangedToMaintenance()**
3. **Aggregates**
   1. **VendingMachine**
   2. **Users**
4. **Repositories**

Repository is in-memory collection of models (objects) → Presentation/Data Wrapper

Used to wrap all CRUD (Create Read Update Delete) Operations

* 1. **VendingMachineRepository**

|  |  |
| --- | --- |
| **Method** |  |
| GetVendingMachineById(int Id) |  |
| SaveChanges() |  |
|  |  |
|  |  |

* 1. **UsersRepository**

|  |  |
| --- | --- |
| **Method** |  |
| ~~GetUserById(int Id)~~ | InsertUser(user) |
| ~~GetUserByUserName(userName)~~ | UpdateUser(user) |
| UserPrivilege GetUserPrivilege(userName, userPass) | DeleteUser(user) |
| SaveChanges() |  |
| InsertNewUser(Name, Pass, Priv) |  |
| User FindById(id) |  |
| List<User> GetAll() |  |

1. **Model Mappers**

Used to map between Domain models and database models

Domain (Source) model  Database (Destination) Model

Implemented in Presentation Layer

Operations:

Map from source to destination  MapForward

Map from destination to source  MapBackward

**namespace** VendingMachine**.**SharedKernel**.**Interfaces

**{**

**public** **interface** IModelMapper**<**SourceT**,** DestinationT**>**

**{**

#region Methods

DestinationT MapForward**(**SourceT domainModel**);**

SourceT MapBackward**(**DestinationT databaseModel**);**

#endregion Methods

**}**

**}**

1. **Solution Architecture**

Shared Kernel

Application

Domain

Presentation

UI

Master DataBase

Presentation Layer

DataBase

Models

Model

Mappers

Domain

Repositories

Application Layer

DTO

Models

Model

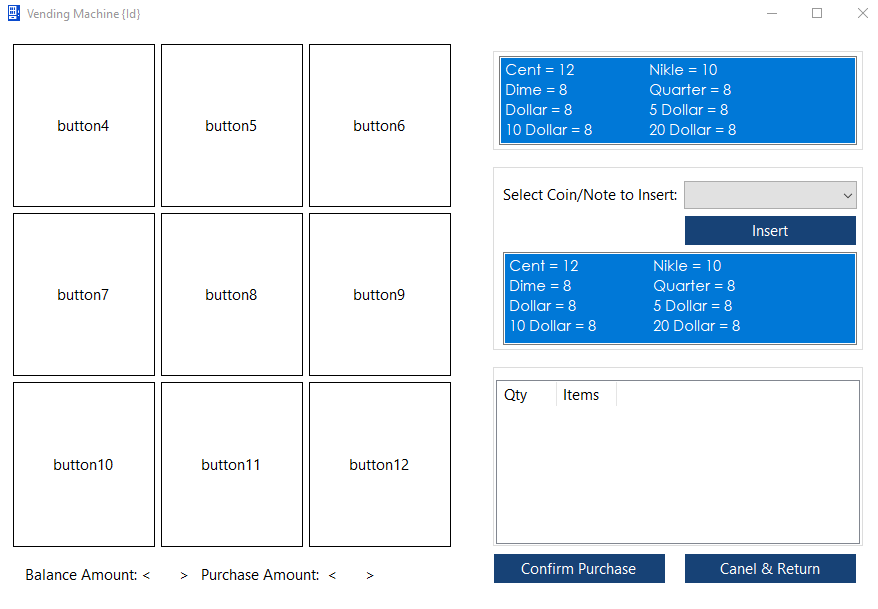
Mappers

Domain

Repositories

UI Layer

1. **GUI**



1. **DataBase**

What data do we need to save? models

How to save it? Repositories in Presentation Layer

Mapping between domain models and DataBase? Presentation Layer in Repository

What Database platform to use? SQL

What framework? Options are MS EF, Dapper, nHibernate!!

* 1. **DataBase Models**
     1. **VendingMachineRecord**

|  |  |  |
| --- | --- | --- |
| Id | Int | PK |
| Model | string |  |
| Manufacturer | string |  |
| Frequency | Int (days) |  |
| LastMaintDate | Date |  |
| GrandBalance | MoneyField |  |
| StartDate | Date |  |
| EndOfLifeDate | Date |  |
| State | string |  |

* + 1. **LocationRecord**

|  |  |  |
| --- | --- | --- |
| VendingMachineId | Int | FK |
| Street | string |  |
| City | string |  |
| Landmark | string |  |

* + 1. **InsideBalanceRecord**

|  |  |  |
| --- | --- | --- |
| VendingMachineId | Int | FK |
| CentCount | Int (1 cent) |  |
| NickelCount | int (5 cent) |  |
| DimeCount | int (10 cent) |  |
| QuarterCount | int (25 cent) |  |
| DollarCount | Int (1 $) |  |
| FiveDollarCount | Int (5 $) |  |
| TenDollarCount | Int (10 $) |  |
| TwentyDollarCount | Int (20 $) |  |

* + 1. **CellsRecord**

|  |  |  |
| --- | --- | --- |
| VendingMachineId | Int | FK |
| CellId | string |  |
| ItemId | Int |  |
| ItemQty | Int |  |

* + 1. **SellItemRecord**

|  |  |  |
| --- | --- | --- |
| ItemId | Int | PK |
| Name | string |  |
| Price | MoneyField |  |
| Barcode | string |  |
| Type | String |  |
| GrandTotal | int |  |
| GrandSellAmount | MoneyField |  |

* + 1. **UserRecord**

|  |  |  |
| --- | --- | --- |
| **Props** | **Primitive Type** | **Behaviors** |
| Id | int |  |
| UserName | string |  |
| UserPassword | string |  |
| UserPrivilege | string |  |

* 1. **Dapper:**

PM> Install-Package Dapper

<https://dapper-tutorial.net/>

**How it works:**

It is a three-step process.

* Create an IDbConnection object.
* Write a query to perform CRUD operations.
* Pass query as a parameter in the Execute method.

**Methods**

Dapper will extend your IDbConnection interface with multiple methods:

* [Execute](https://dapper-tutorial.net/execute)
* ExecuteReader
* ExecuteScalar<T>
* [Query](https://dapper-tutorial.net/query)
* [QueryFirst](https://dapper-tutorial.net/queryfirst)
* [QueryFirstOrDefault](https://dapper-tutorial.net/queryfirstordefault)
* [QuerySingle](https://dapper-tutorial.net/querysingle)
* [QuerySingleOrDefault](https://dapper-tutorial.net/querysingleordefault)
* [QueryMultiple](https://dapper-tutorial.net/querymultiple)

|  |
| --- |
| string sqlOrderDetails = "SELECT TOP 5 \* FROM OrderDetails;";  string sqlOrderDetail = "SELECT \* FROM OrderDetails WHERE OrderDetailID = @OrderDetailID;";  string sqlCustomerInsert = "INSERT INTO Customers (CustomerName) Values (@CustomerName);";  using (var connection = new SqlConnection(FiddleHelper.GetConnectionStringSqlServerW3Schools()))  {  var orderDetails = connection.Query<OrderDetail>(sqlOrderDetails).ToList();  var orderDetail = connection.QueryFirstOrDefault<OrderDetail>(sqlOrderDetail, new {OrderDetailID = 1});  var affectedRows = connection.Execute(sqlCustomerInsert, new {CustomerName = "Mark"});  Console.WriteLine(orderDetails.Count);  Console.WriteLine(affectedRows);  FiddleHelper.WriteTable(orderDetails);  FiddleHelper.WriteTable(new List<OrderDetail>() { orderDetail });  } |

**Parameter**

Execute and queries method can use parameters from multiple different ways:

* [Anonymous](https://dapper-tutorial.net/parameter-anonymous)
* [Dynamic](https://dapper-tutorial.net/parameter-dynamic)
* [List](https://dapper-tutorial.net/parameter-list)
* [String](https://dapper-tutorial.net/parameter-string)

|  |
| --- |
| // Anonymous  var affectedRows = connection.Execute(sql,  new {Kind = InvoiceKind.WebInvoice, Code = "Single\_Insert\_1"},  commandType: CommandType.StoredProcedure);  // Dynamic  DynamicParameters parameter = new DynamicParameters();  parameter.Add("@Kind", InvoiceKind.WebInvoice, DbType.Int32, ParameterDirection.Input);  parameter.Add("@Code", "Many\_Insert\_0", DbType.String, ParameterDirection.Input);  parameter.Add("@RowCount", dbType: DbType.Int32, direction: ParameterDirection.ReturnValue);  connection.Execute(sql,  parameter,  commandType: CommandType.StoredProcedure);  int rowCount = parameter.Get<int>("@RowCount");  // List  connection.Query<Invoice>(sql, new {Kind = new[] {InvoiceKind.StoreInvoice, InvoiceKind.WebInvoice}}).ToList();  // String  connection.Query<Invoice>(sql, new {Code = new DbString {Value = "Invoice\_1", IsFixedLength = false, Length = 9, IsAnsi = true}}).ToList(); |

**Result**

The result returned by queries method can be mapped to multiple types:

* [Anonymous](https://dapper-tutorial.net/result-anonymous)
* [Strongly Typed](https://dapper-tutorial.net/result-strongly-typed)
* [Multi-Mapping](https://dapper-tutorial.net/result-multi-mapping)
* [Multi-Result](https://dapper-tutorial.net/result-multi-result)
* [Multi-Type](https://dapper-tutorial.net/result-multi-type)

|  |
| --- |
| string sqlOrderDetails = "SELECT TOP 10 \* FROM OrderDetails;";  using (var connection = new SqlConnection(FiddleHelper.GetConnectionStringSqlServerW3Schools()))  {  var anonymousList = connection.Query(sqlOrderDetails).ToList();  var orderDetails = connection.Query<OrderDetail>(sqlOrderDetails).ToList();  Console.WriteLine(anonymousList.Count);  Console.WriteLine(orderDetails.Count);  FiddleHelper.WriteTable(orderDetails);  FiddleHelper.WriteTable(connection.Query(sqlOrderDetails).FirstOrDefault());  } |

**Utilities**

* [Async](https://dapper-tutorial.net/async)
* [Buffered](https://dapper-tutorial.net/buffered)
* [Transaction](https://dapper-tutorial.net/transaction)
* [Stored Procedure](https://dapper-tutorial.net/stored-procedure)

|  |
| --- |
| // Async  connection.QueryAsync<Invoice>(sql)  // Buffered  connection.Query<Invoice>(sql, buffered: false)  // Transaction  using (var transaction = connection.BeginTransaction())  {  var affectedRows = connection.Execute(sql,  new {Kind = InvoiceKind.WebInvoice, Code = "Single\_Insert\_1"},  commandType: CommandType.StoredProcedure,  transaction: transaction);  transaction.Commit();  }  // Stored Procedure  var affectedRows = connection.Execute(sql,  new {Kind = InvoiceKind.WebInvoice, Code = "Single\_Insert\_1"},  commandType: CommandType.StoredProcedure); |

* 1. **EntityFrameWork**

Project should be UI type

Install Nuget Package: Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

Microsoft.EntityFrameworkCore.Design

Create Single tone Context Class VendingMachineContext

**using** Microsoft**.**EntityFrameworkCore**;**

**public** **sealed** class VendingMachineContext **:** DbContext

**{**

**public** DbSet**<**CellsRecord**>** CellsRecords **{** **get;** **set;** **}**

**public** DbSet**<**InsideBalanceRecord**>** InsideBalanceRecords **{** **get;** **set;** **}**

**public** DbSet**<**LocationRecord**>** LocationRecords **{** **get;** **set;** **}**

**public** DbSet**<**SellItemRecord**>** SellItemRecords **{** **get;** **set;** **}**

**public** DbSet**<**UserRecord**>** UserRecords **{** **get;** **set;** **}**

**public** DbSet**<**VendingMachineRecord**>** VendingMachineRecords **{** **get;** **set;** **}**

**public** static VendingMachineContext InstanceFactory**(**string connectionString**)**

**{**

**if** **(**instance **==** **null)**

instance **=** **new** VendingMachineContext**();**

instance**.**connectionString **=** connectionString**;**

**return** instance**;**

**}**

**protected** **override** void OnConfiguring**(**DbContextOptionsBuilder optionsBuilder**)**

**{**

optionsBuilder**.**UseSqlServer**(this.**connectionString**);**

**base.**OnConfiguring**(**optionsBuilder**);**

**}**

**}**

To enable/support EF designtime, implement IDesignTimeDbContextFactory

**public** class VendingMachineContextFactory **:** IDesignTimeDbContextFactory**<**VendingMachineContext**>**

**{**

**public** static string Default\_Connection **=** @"Data Source=(LocalDB)\MSSQLLocalDB;" **+**

@"AttachDbFilename=C:\Users\Gemi\source\repos\VendingMachine\VendingMachine.Presentation\DataBase\MasterDataBase.mdf;" **+**

"Integrated Security=True"**;**

**public** VendingMachineContext CreateDbContext**(**string**[]** args**)**

**{**

**return** **new** VendingMachineContext**(**Default\_Connection**);**

**}**

**}**

Add PK (Id) to all tables for indexing

Add Data Annotations to classes to identify PK, FK, Column Names, amx length, etc..

using System.ComponentModel.DataAnnotations

using System.ComponentModel.DataAnnotations.Schema

Annotate restrictions to database models properties (max size, isNull, etc…)

(<https://www.c-sharpcorner.com/article/understanding-entity-framework-core-and-code-first-migrations-in-ef-core/>)

**public** class VendingMachineRecord

**{**

**[**Key**]**

**[**Column**(**"VendingMachineId"**)]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** int Id **{** **get;** **set;** **}**

**[**MaxLength**(**50**)]**

**public** string Model **{** **get;** **set;** **}**

**[**MaxLength**(**50**)]**

**public** string Manufacturer **{** **get;** **set;** **}**

**[**DefaultValue**(**90**)]**

**public** int Frequency **{** **get;** **set;** **}**

**[**DefaultValue**(**"getutcdate()"**)]**

**public** DateTime LastMaintDate **{** **get;** **set;** **}**

**public** string GrandBalance **{** **get;** **set;** **}**

**[**DefaultValue**(**"getutcdate()"**)]**

**public** DateTime StartDate **{** **get;** **set;** **}**

**[**DefaultValue**(**"getutcdate()"**)]**

**public** DateTime EndOfLifeDate **{** **get;** **set;** **}**

**[**DefaultValue**(**"Operational"**)]**

**public** string State **{** **get;** **set;** **}**

**}**

**Adding a Migration:**

Create DataBase Table from models and context class; The migrations feature in EF Core provides a way to incrementally update the database schema to keep it in sync with the application's data model while preserving existing data in the database

PM> add-migration -Name create\_tables -Context VendingMachineContext

**Update DataBase:**

PM> Update-Database

1. **General Rules**

* CQRS ⇒ Implement Command Query Responsibility Segregation all the time
* Value Objects are Immutable ⇒ values should not change, create new one

# **HeadQuarter Subdomain**

* Future expansion that each VM report back to HQ main account
* Report all balance details (grand and inside)
* Report Inventory details
* Alarm if further inventory update is needed
* HQ shall CRUD VMs
* Deleting VM does not delete its record, just end of life value

# **ATM Subdomain:**

* Add ATM subdomain
* Link with HQ account
* Link with VM