

CAPSTONE CLASS DOCUMENT

May 1, 2020




Table of Contents

Introduction.....	2
Objectives	2
Environment.....	2
Assumptions.....	3
Basic Section.....	4
Section: Setup	4
Section: Directory Apps & Setup	6
Section: Order Module Settings.....	7
Section: IngredientsDefinition Table & Application.....	9
Section: Products Table	10
Section: ProductIngredients Table & Application.....	11
Section: Products Application.....	11
Advanced Section	12
Section: Products Workflow	12
Section: App Settings & Menus.....	14
Section: Discounts Application.....	15
Section: Discounts Logic Blocks.....	17
Section: Discounts Workflow	18
Section: Discount Offers Application.....	19
Section: Orders Application.....	20
Section: Order Logic Blocks.....	23
Section: Order Workflow.....	24
Advanced II Section.....	25
Section: Write GL Transaction	25
Section: Batch Processing (Create Discount Offers)	27
Section: Reports.....	28
Bonus Section	28
Bonus Section: Validations.....	28
Bonus Section: Add Memo to GL	29
Bonus Section: Integrate Discounts in Orders	29
Bonus Section: Order Balances	29
Bonus Section: External Callout.....	29
Capstone Data Model and Workflows.....	30
Data Model.....	30
Workflows.....	31

Introduction

Each step should be followed sequentially. The document describes what should be built but intentionally leaves out how you should do it.

Pizza is a food enjoyed across the world. A critical piece to the culture of pizza, especially in the U.S., are pizza chains. Those chains typically follow the franchise model, where a corporation sells the rights to a store in a certain location.

Your goal is to build a "mini" module for those pizza franchises. The focus is on defining the stores themselves with related attributes, products, discounts, discount offers per store, and managing orders, in addition to tracking core statistical information on the orders. It will integrate with Nextworld's Financial module.

The capstone has a required section and a bonus section. The bonus section can be attempted if the required portion is completed prior to the expected end date.

Objectives

At the end of the project, you will have done this:

- Basic Section
 - A module settings application that defines core settings for the franchise product
 - Added search applications for Table Lookup fields
 - Viewed several Directory applications and added Directory records
 - Viewed the Chart of Accounts and added a new GL account
 - Built Data Items and Tables for the Ingredients application
 - Built Data Items and Tables for the Products application
 - Built an application to defined Products that are available to sell
 - Built an application to view ingredients for each Product
- Advanced Section
 - Built workflow for Products
 - Built an application to define Discounts
 - Built workflow for Discounts which includes approvals so that a manager can approve or reject
 - Built an application to attach Discounts to Stores
 - Built logic blocks for events in an application: hiding fields, showing fields, enabling, disabling, defaulting values
 - Built logic blocks for table events: pre-insert, pre-update
 - Built an application link which uses data mappings to default a filter
 - Built an automatic number and automatic number format
 - Built a Header Detail application to accept Orders
 - Built a workflow which calls a logic block in a transition between states
 - Built logic blocks to update a header based on detail lines
 - Built a logic block which calls Nextworld's WriteGLTransaction logic block - this will create a Journal Entries
 - Built an application which is used for a batch process: Create many Discount Offers at one time so the customer does not manually have to do it

Environment

Naming: To avoid name conflicts, **add your initials to the beginning of each object you create**. For example, nsTrnRWOrdersHeader.

Lifecycle: Create all objects in your individual student lifecycle

Family: Create all objects in the `Playpen` product family

Module: Personal

In some cases objects are already created for you which can be used in your applications. Please review the Data Model section; it lists what objects need to be created or what values can be used if they already exist.

Assumptions

In general, this is a simple system. A comprehensive system would include core features like cost management, inventory tracking, customer profiles, and an order portal. These are the core assumptions for the capstone:

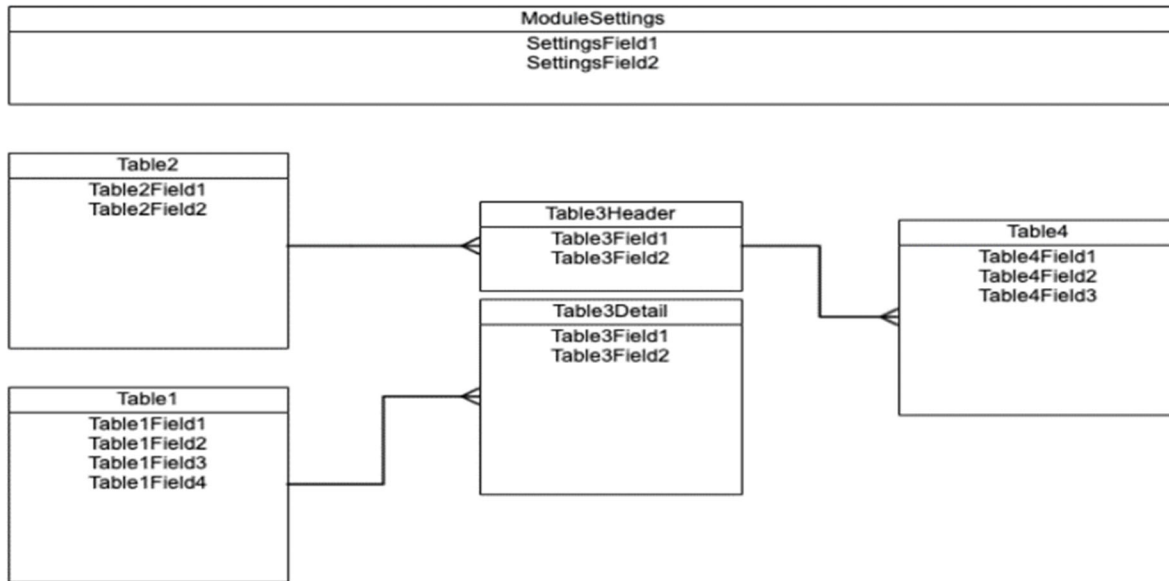
- Orders are taken by the franchise employees in the main Orders application and cannot be placed by external customers (no web or mobile orders)
- No custom orders (must order from the menu)
- No customer profile where they can log in
- The system does not track inventory and costs
- Each franchise shares the same product options
- Discounts are applied on a product / per order basis

Basic Section

Section: Setup

Context: Before reviewing the data model provided for the capstone, come up with your own data model based on your knowledge of Nextworld table and app patterns. The rest of the capstone will use the capstone provided but the exercise will be helpful because data modeling in Nextworld is tied to the patterns that we provide.

Example:




Task	Details
Come up with a data model	<p>Based on the requirements below, come up with a data model. The data model should include each table, the data items for the table, and the relationship between those tables.</p> <p>Requirements:</p> <ul style="list-style-type: none"> • I need to define module settings for the pizza franchise model • I need to define the stores that are part of my company structure • I need to define products <ul style="list-style-type: none"> ◦ Products will have a name, category (e.g. pizza, pasta), subcategory (e.g. single topping pizza, gluten free pasta), price • I need to define the ingredients for each product <ul style="list-style-type: none"> ◦ Each ingredient entry is tied to a product ◦ Ingredients will have a ingredient name, quantity, and unit of measure • I need to define discounts. This is a master list of discounts that are available, not whether they are active discounts for each store. <ul style="list-style-type: none"> ◦ The discount can be defined for a single product, sub category, or full product category. ◦ The discount can be for all stores or a single store • I need to apply discounts to individual stores and define when those discounts start and end. <ul style="list-style-type: none"> ◦ Reference a discount (table lookup) ◦ Reference a single store (table lookup) ◦ Start and end date • I need to define orders. <ul style="list-style-type: none"> ◦ A single order can have multiple products and quantities. • I need to send summarized order information to the financial module. This will be done by calling the WriteGLTransaction logic block. • Please show the general relationship between orders and the General Ledger. • I need to summarize order transactions in a balances file. <ul style="list-style-type: none"> ◦ I want to view per store, per order type, per year, per period, per product: <ul style="list-style-type: none"> ▪ total sales ▪ total discounts
Review and compare to suggested data model for the project	Review the data model we will actually use to build the project: Capstone Data Model & Workflows .
Think about the application pattern for each table	For each table, think about which app pattern makes the most sense.

Currency Manager	<p>We need to ensure we have a currency setup in the system. Go to the Currency Manager and add the following <u>if it doesn't exist</u>:</p> <ul style="list-style-type: none"> • Name: US Dollars • Currency Code: USD • Currency Symbol: \$ • Currency Decimals: 2 • Currency Format: True • Currency Default: True • Active: True
Base Module Settings	<p>Because each person completing the project will use their own data, we need to allow multiple companies. Base Module Settings are shipped to every customer. Go to Base Module Settings and ensure Allow Multiple Companies <u>is set to true</u>.</p>

Section: Directory Apps & Setup

Context: The Directory family is a core part of the Nextworld application suite. Rather than recreate the applications, we will use what already exists. You just need to add data to the Directory that we will use throughout the project. In addition, you will create a cash GL Account which you will reference in your module settings. We need to create the Company Directory record before we can create the cash account.

Task	Details
Add Data to Company Structure	<ul style="list-style-type: none"> • Add a Company <ul style="list-style-type: none"> ◦ Create a pizza company - you can name it whatever you'd like • Add Divisions data (hint: when hovering over your company in the tree, click "Create Entity" button) <ul style="list-style-type: none"> ◦ Attach a couple divisions to your company • Add Stores data <ul style="list-style-type: none"> ◦ Org Unit Type = Store ◦ Attach at least one store to a division
Create GL Account for Company	<p>Now that we have a Company created, we can create a Cash GL account. It is one of the few GL Accounts that are tied to a company.</p> <ul style="list-style-type: none"> • Go the the Chart of Accounts application. • Navigate through the tree 1000 Assets => 10100 Current Assets => 10200 Cash. Hover over 10200 and click "Create Entity" to create a new cash account. Add these properties: <ul style="list-style-type: none"> ◦ Account Number ◦ Account Name ◦ Financial Statement = Balance Sheet ◦ Normal Balance = Debit ◦ Cash = True ◦ Company = the company you created in the previous task

Add the GL Account to Module Settings	<ul style="list-style-type: none"> Go to your module settings application and create a global rule. Add the GL Account you just created to the Primary Cash Account field.
Add Employees	<ul style="list-style-type: none"> Go to the Employees application Create yourself as an employee <ul style="list-style-type: none"> Attach a store as the Primary Org Unit <ul style="list-style-type: none"> Organizations page Add a record <ul style="list-style-type: none"> Relationship Type = Org Unit Primary = True <p> This is an example of a relationship application that is not a "tree".</p>
<i>Checkpoint</i>	<p>You have created a:</p> <ul style="list-style-type: none"> Company Division Store Cash GL Account Assigned the Cash account to global module settings for your Company Employee


Section: Order Module Settings

Context: Module settings are a core piece of any product module built on the Nextbot platform. Unlike Application Settings, which are used to modify the look and feel of an app or default values, module settings are used to define behavior for a module. A setting will apply globally to all organizational units or can be applied for a single organizational unit. For the pizza module, we have a few settings that will be used throughout the rest of the capstone.

Before starting, review the documentation:

<https://apps.nextworld.net/#/drillink?NwDrillApplication=contentviewer&NwDrillForm=other&nwId=NextbotFamily.Nextbot.NextbotConcepts.NextbotApplicationsConcept.NextworldApplicationTypes.ModuleSettingsApplicationType>

Task	Details
Create Data Items	<ul style="list-style-type: none"> OrderTypeOptions Create the List Lookup (Multi-Select List Lookup) <ul style="list-style-type: none"> Values: <ul style="list-style-type: none"> CATER / Cater DINEIN / Dine In DEL / Delivery TAKE / Takeout PrimaryCashAccount

	<ul style="list-style-type: none"> StandardDiscount <ul style="list-style-type: none"> Mark as Percentage = True It should not allow values less than 0 and greater than 100. <p> Consider the Display Size of the Data Items</p>
Create Table	<ul style="list-style-type: none"> Table Type: Settings Settings Configuration page <ul style="list-style-type: none"> Make sure you Allow Global Settings and Allow Organizational Settings. Add Fields: <ul style="list-style-type: none"> GlobalSetting (this already exists, you did not create it) OrganizationalUnitName (this already exists, you did not create it) OrderTypeOptions <ul style="list-style-type: none"> AllowGlobal = True For the Organizational settings, in the Type column, select the Any type PrimaryCashAccount <ul style="list-style-type: none"> AllowGlobal = True For the Organizational settings, in the Type column, select the Any type
Table Attributes	<ul style="list-style-type: none"> Filtered Table Lookups: <ul style="list-style-type: none"> When selecting a value for an organizational unit, I can only select internal organizational units (hint: 'Internal Organization'=ContactType) When selecting a GL Account, I can only select Cash accounts (hint: 'True'=Cash) Indexes: <ul style="list-style-type: none"> The record marked as a GlobalSetting should always be at the top (hint: add primary index) I can only create 1 setting per organizational unit (hint: add unique index for OrganizationalUnitName)
Create Application	<p>Review: https://apps.nextworld.net/#!/drilllink?NwDrillApplication=contentviewer&NwDrillForm=other&nwId=NextbotFamily.Nextbot.NextbotConcepts.NextbotApplicationsConcept.NextworldApplicationTypes.ModuleSettingsApplicationType.ModuleSettingsApplicationConfigurationRequirements</p> <ul style="list-style-type: none"> Add List View fields: <ul style="list-style-type: none"> GlobalSetting OrganizationalUnitName OrganizationalUnitName_OrganizationalUnitType ... the custom settings fields you added Add Detail View fields: <ul style="list-style-type: none"> A1 - GlobalSettings A1 - OrganizationalUnitName_Name A2 - OrganizationalUnitName_OrganizationalUnitType B1 - OrderTypeOptions B2 - PrimaryCashAccount B3 - StandardDiscount Pages/Rows

	<ul style="list-style-type: none"> ○ A should be the Header row • If the magnifying glass is disabled for the <code>OrganizationalUnitName</code> table lookup, a search application needs to be added. <ul style="list-style-type: none"> ○ Use the base search app that already exists. Add a search action: <ul style="list-style-type: none"> ▪ Search Field: <code>OrganizationalUnitName_Name</code> ▪ Search App: <code>OrganizationalUnitsSearch</code> ▪ Setting Name: <code>InternalOrganizationFilter</code> ▪ Form: List ▪ Data Mappings: <ul style="list-style-type: none"> ▪ Add a constant value of “Search Org Units” for the Title • The primary cash account also needs a search application if it doesn't have one: <ul style="list-style-type: none"> ○ Search Field: <code>YourPrimaryCashAccount_AccountNumber</code> ○ Search App: <code>GLAccountSearchCash</code> ○ Setting Name: <code>GLCashAccountSettings</code> ○ Form: List ○ Data Mappings: <ul style="list-style-type: none"> ▪ Add a constant value of “Search Cash Accounts” for the Title
<i>Checkpoint</i>	<ul style="list-style-type: none"> • I can create a global setting <ul style="list-style-type: none"> ○ I can define order type options and a primary cash account • I can create an organizational unit setting <ul style="list-style-type: none"> ○ I can define order type options and a primary cash account • I cannot add more than one setting for the same org unit

Section: IngredientsDefinition Table & Application

Context: Every food product is made up of ingredients. In a kitchen many ingredients often are shared. Given that, it makes sense to have a master list of ingredients. The ingredients can then be selected by individual products. In a full implementation, the ingredients per product could be used to track cost and inventory. In our simpler implementation, they are used to allow products to define the ingredients needed for an order.

Task	Details
Create Data Items	<ul style="list-style-type: none"> • Use the Data Model to see what data items and list lookups need to be created. • Notes: <ul style="list-style-type: none"> ○ For the the <code>UnitOfMeasure</code> Data Item, make sure to use the existing Units of Measure List Lookup
Create Table	<ul style="list-style-type: none"> • Add fields based on the data model • Mark <code>UnitOfMeasure</code> and <code>Description</code> as Table Lookup fields

Table Attributes	<ul style="list-style-type: none"> • IngredientName should be the primary typeahead • IngredientName should sort A - Z • IngredientName should be unique
Create Application	<ul style="list-style-type: none"> • App Type: Advanced List • App Style: Standard
<i>Checkpoint</i>	<ul style="list-style-type: none"> • You are required to give each record a name and unit of measure • An ingredient name can only exist once • Ingredients are sorts A - Z

Section: Products Table



Context: Every pizza company sells food; that food can be considered a store's products. This application is used to define the products which are used throughout the rest of the module. It's a fairly simple setup. Each product has a name, category and subcategory as well as a listing of ingredients.

Given the capstone is a simpler implementation, the products are shared across all stores meaning that once a product is defined and active, it can be used by all stores.

Task	Details
Create Data Items	<ul style="list-style-type: none"> • Use the Data Model to create the List Lookups • Use the Data Model to create the Data Items • Notes: <ul style="list-style-type: none"> ◦ The ProductSubCategory List Lookup is dependent on the ProductCategory List Lookup
Create Table	<ul style="list-style-type: none"> • Define the dependent lookup between ProductSubCategory and ProductCategory in the table definition • You can create the ProductIngredients subtable data item but cannot add it to the Products table yet because the table does not exist yet. Subtable data items are linked to their actual tables by the name. The platform knows to grab the fields from the ProductIngredients table if it has a subtable with the same name. You will get an error if you try to add the subtable and the table does not exist. We will come back to it in the next steps. <p>Now that the table is created, you have to go back to edit any data items that are a table lookup on this table and add the Products table as the Related Table.</p>
Table Attributes	<ul style="list-style-type: none"> • ProductName should be the primary typeahead • ProductName should sort A - Z • ProductName should be unique



Section: ProductIngredients Table & Application

Context: This is the link between ingredients and products. We are using a subtable to define many ingredients per product. The platform helps us by linking records in the subtable to the parent record.

Task	Details
Create Data Items	<ul style="list-style-type: none">• Use the Data Model to create the Data Items
Create Table	<ul style="list-style-type: none">• Create the table and add the data items per the data model.
Table Attributes	<ul style="list-style-type: none">• Sort Product A - Z• The Product and Ingredient combination are unique
Create Application	<ul style="list-style-type: none">• Type: List Only• Read Only App - True• Layout - Since it's Read-Only, the Detail View isn't required. <p> This is an example of an application built over data that is entered through subtables. Since the table is of type "Main", we can build an application over it. We only want users to modify the ingredient information through the product applications, so we can make this one read-only.</p>
Update Products Table with Subtable	<ul style="list-style-type: none">• Add your ProductIngredients subtable data item to the Products table.• In addition, we need to setup common fields to default the <code>nwId</code> of the product to the Product field in the subtable. This is to improve the user experience. We don't want the user to have to type in the product they are already editing when adding an ingredient. To do that:<ul style="list-style-type: none">◦ In the Products table, in the Relationships tab, add a record to the Sub Lists → Common Fields subtable with your values. <p> How Are Subtables Linked to Parent Records? The platform handles this for you. When you save a record (Products) with a subtable (ProductIngredients), the ProductIngredient records are stored in their own table (ProductIngredients) with a link to the parent record. The link is stored in the <code>nwHeaderId</code> field. When a user pulls up a Product record, the platform checks if there are any subtables linked to it and retrieves the data automatically.</p>

Section: Products Application

Context: This is where stores create and manage their products. Since this is a simple implementation, every active product is available for each store.

Task	Details
Create Application	<ul style="list-style-type: none"> Type: Standard Style: Standard <p>Layout:</p> <ul style="list-style-type: none"> Header Row should have the four main fields: name, category, price, subcategory Description should be across the whole page The ingredients subtable have its own page and be across the whole page <p>Subtable Fields:</p> <ul style="list-style-type: none"> Ingredient_InгредиентName, Ingredient_UnitOfMeasure, and Quantity should be marked as: <ul style="list-style-type: none"> Hide on Table = False
 Checkpoint	<ul style="list-style-type: none"> You can define various products with different categories and subcategories ProductSubCategory is filtered based on the ProductCategory ProductName is unique ProductName sort A - Z You can add ingredients to Product in the subtable. The Product is defaulted (user does not have to select the product when adding an ingredient) You receive an error if you add the same ingredient twice in the subtable
Create Products Search Application	<p> All table lookups can have a search application. Often it makes sense to show the same search application for every table lookup on a table without having to define it everywhere it's used. By defining a default search application the table, any table lookups will use that search app unless it's overridden at the application.</p> <ul style="list-style-type: none"> Create App <ul style="list-style-type: none"> App Name: ProductsSearch App Type - Standard App Style - Mini Table: your products table Attach it as the default search app (in the Relationships page) of your Products table

Advanced Section


Section: Products Workflow

Context: Depending on a company's internal processes, a product could require several steps. It likely follows a business process before it is available for customers to order and enjoy. Given that requirement, we will add workflow to the Products application. Since each company could have a different process, the standard workflow process will be simple; a franchise could customize the workflow to add attach approvals or expand the lifecycle of a product.

Task	Details
Create List Lookups & Data Items	<ul style="list-style-type: none"> • Products Workflow Status - Format: VALUE / Description <ul style="list-style-type: none"> ◦ Values <ul style="list-style-type: none"> ▪ NEW / New ▪ ACTIVE / Active ▪ INACTIVE / Inactive • Products Workflow Type <ul style="list-style-type: none"> ◦ Values <ul style="list-style-type: none"> ▪ INIT / Initial ▪ ACTIVE / Active ▪ PENDING / Pending ▪ CLOSED / Closed ▪ ERR / Error
Create Workflow	<p>Go to the Workflow Definitions application</p> <ul style="list-style-type: none"> • State Lookup: Products Workflow Status • Table: Your Products table • State Type Lookup: Products Workflow Type • Details - see the workflow definition steps in the appendix • After adding the details, mark Production = True and Save.
<i>Checkpoint</i>	<ul style="list-style-type: none"> • Navigate to your Products application, you should now see the Status field as the farthest left column. • Click “Create” - you should now see the workflow status in the top right next to the Actions button. It should have the “New” value. • You can move the product from New to Active • You can move the product from Active to Inactive • You can move the product from Inactive to Active

Section: App Settings & Menus

Context: We've created several applications. We will add them to the menu so it's easier to access them while developing.


Task	Details
Create App Settings	<ul style="list-style-type: none">• Create app settings for: PizzaSettings, IngredientsDefinition, ProductIngredients, Products, ProductsSearch<ul style="list-style-type: none">◦ CustomizationPattern = AllowAdditions <p> It is our best practice and standard to create an app setting for every application. This allows customers to customize the app setting and make changes. Otherwise, customers would need to customize the application to attach a new app setting.</p>
Add menus for your applications	Create menus for: PizzaSettings, IngredientsDefinition, ProductIngredients, Products
<i>Checkpoint</i>	<ul style="list-style-type: none">• PizzaSettings, IngredientsDefinition, ProductIngredients, Products are all on the menu

Section: Discounts Application

Context: Every store offers discounts. This application is a master list of discounts which are available for stores to offer to customers. We want to provide some flexibility, which is why discounts can be defined as an amount or a percentage. The discounts can also be applied to a single product, sub category, or entire product category. Since stores are in different locations and may operate individually, another application (DiscountOffers) will be used to tie a discount to a store; this gives each store the freedom to choose which discounts to offer. Building this application will require several logic blocks to improve the end-user experience.


If a full system were being built, several other attributes would be added to make discounts robust: minimum quantities, max quantities, max discount amount, etc.

Task	Details
Create Data Items	<ul style="list-style-type: none">• Create List Lookups<ul style="list-style-type: none">◦ Discount Type<ul style="list-style-type: none">▪ AMT / Amount▪ PER / Percentage• Create data items based on the data model. Note that some of them were already created in a previous section.<ul style="list-style-type: none">◦ DiscountPercent - Formatting & Validations<ul style="list-style-type: none">▪ Decimals - 2▪ Minimum Value - 0▪ Maximum Value - 100◦ DiscountAmount - Formatting & Validations:<ul style="list-style-type: none">▪ No negative values
Create Discount Table	<ul style="list-style-type: none">• Add fields to the table• Table Lookup fields:<ul style="list-style-type: none">◦ Product, ProductCategory, ProductSubCategory, DiscountAmount, DiscountPercent
Table Attributes	<ul style="list-style-type: none">• DiscountName should be the Primary Typeahead• Define the dependent lookup between ProductSubCategory and ProductCategory• The Store field should be filtered to only internal organizations ('Internal Organization'=ContactType)• The Product filter should be filtered to only allow active products ('ACTIVE'=znwWorkflowInstance)• DiscountName should sort A - Z• DiscountName should be unique

Create Application	<p>Define the list and detail view. We will add logic blocks in a later section which will make the app more functional, including hiding / showing the number or percent based on the discount type.</p> <p>Layout:</p> <ul style="list-style-type: none"> List View - defined as you'd like Detail View <ul style="list-style-type: none"> Row A: DiscountName, DiscountType, ForAllStores, Store Row B: ProductCategory, ProductSubCategory, Product, DiscountAmount, DiscountPercent <ul style="list-style-type: none"> Note: you should stack Amount and Percent in the same row and column. Since only one will ever show, it will look normal. By stacking them, it avoids having a gap in the UI where one of the fields should be. <p>Extra:</p> <ul style="list-style-type: none"> 4 columns per row for both rows so that the rows look uniform Row B should have a row title of "Discount Details" Add a search action for the Store field: <ul style="list-style-type: none"> Search App: OrganizationalUnitsSearch Setting Name: InternalOrganizationFilter Form: List Data Mappings: <ul style="list-style-type: none"> Add a constant value of "Search Stores" for the title <p> The UI is not exactly how we want it yet. We will refine it in follow sections with logic blocks.</p>
Create Search Application	<ul style="list-style-type: none"> App Name: DiscountsSearch App Type - Standard App Style - Mini Table: your discounts table Read Only Application = True <p>After it's created:</p> <ul style="list-style-type: none"> Attach it to your Discounts table as the default search app
Checkpoint	<ul style="list-style-type: none"> You can create a discount You get an error if a discount name is used twice Discount name is sorted A - Z You can only pick internal org units in the Store field You can only pick active products in the Product field The discount percent cannot be less than 0 or greater than 100.

Section: Discounts Logic Blocks

Context: This is where we make the Discounts app interactive. As it stands, the app is usable but confusing. Using logic blocks we can hide/show/enable/disable fields to help the user understand what they need to do.

Task	Details
Create App Logic Blocks	<ul style="list-style-type: none">• Create logic block to hide/show fields<ul style="list-style-type: none">◦ Name: HideShowDiscounts◦ LB Type: UI◦ Table: Discounts◦ Requirements<ul style="list-style-type: none">▪ If the DiscountType is amount, show DiscountAmount and DiscountPercent should be hidden. The opposite is true if the DiscountType is percent.▪ Only show the Store field if ForAllStores is true• Name: EnableDisableDiscounts<ul style="list-style-type: none">◦ Table Type: UI◦ Table: Discounts◦ Requirements:<ul style="list-style-type: none">▪ If you pick a product, then category and subcategory are disabled and field value is cleared▪ If you pick a category or subcategory, product is disabled and field value is cleared <p></p> <ul style="list-style-type: none">• For the logic blocks that run on event actions, always consider whether they should also run on Form Init. If you expect fields to be disabled but only do that when a user tabs out of a field, then those fields will not be disabled when they go into an existing record.• For advanced list applications, consider why row entered actions may be needed
Attach LBs to Application	<ul style="list-style-type: none">• HideShowDiscounts<ul style="list-style-type: none">◦ Events:<ul style="list-style-type: none">▪ LB should be called when the form loads▪ There are <i>two</i> places where this LB should be called for Field Value Changed• EnableDisableDiscounts<ul style="list-style-type: none">◦ Events:<ul style="list-style-type: none">▪ LB should be called when the form loads▪ There are <i>three</i> places where this LB should be called for Field Value Changed

Create Table Triggers	<p>i As a best practice, all validations should happen at the table level with triggers or an action block. Adding validations in applications is used to improve the user experience and guide the user through the business process; however, adding a validation in the app won't keep invalid data from being stored in the database. For example, data may come into the Discounts table from a data import, external API call, or another external system. For the sake of time, we won't add all the validations at the table level.</p> <ul style="list-style-type: none"> Name: ValidateDiscounts <ul style="list-style-type: none"> Table Type: Validation Table: Discounts Requirements: <ul style="list-style-type: none"> If ForAllStores is false AND Store is empty, show an error. You can use the nsPtrMASTERMsg1 Data Item. At least one of these cannot be empty: ProductCategory, ProductSubCategory, Product. You can use the nsPtrMASTERMsg2 Data Item.
Attach Table Triggers	<ul style="list-style-type: none"> ValidateDiscounts - should run when a record is inserted or updated
<i>Checkpoint</i>	<ul style="list-style-type: none"> If DiscountType = Percentage, the percent field shows. If DiscountType = Amount, the amount field shows. You can create a discount with an amount type and discount type If For All Stores = False, the Store field shows. Vice versa if it is True. Product is disabled if a category or subcategory is selected. Category and subcategory are disabled if a product is selected. You receive an error if For All Stores = False and no Store is provided You receive an error if category, subcategory, and product are empty

Section: Discounts Workflow

Context: Depending on a company's internal processes, a discount could require several steps. It likely follows a business process before it is available for stores to use. Given that requirement, we will add workflow to the Discounts application. Since each company could have a different process, the standard workflow process will be simple; a franchise could customize the workflow to add attach approvals or expand the lifecycle.

Task	Details
Discounts Workflow	<ul style="list-style-type: none"> Go to the Workflow Definitions application <ul style="list-style-type: none"> State Lookup: nsTrnMASTER Discounts Workflow Status Table: Your Discounts table State Type Lookup: nsTrnMASTER Discounts Workflow Type Details - see the workflow definition steps in the appendix After adding the details, mark Production = True and Save.

Checkpoint

- The workflow defaults to New when creating a record
- You can transition the workflow to Active → Inactive and vice versa

Section: Discount Offers Application

Context: The Discounts application built in the prior steps is used to define a master list of discounts. Discount Offers is used to assign discounts to individual stores since stores can selectively offer promotions. For now, Discount Offers are manually created; later on we will create a batch process.

Task	Details
Create Discount Offers Data Items	<ul style="list-style-type: none">• Create Data Items based on the data model<ul style="list-style-type: none">◦ ActiveFrom<ul style="list-style-type: none">▪ Set it to default to current time in data item (hint: Formatting page)
Create Table & Table Attributes	<ul style="list-style-type: none">• Sorts on Active To (Z - A) and Discount (A - Z)• Unique index on all 4 fields• Primary typeahead field is Discounts (TL)• Filtered Table Lookups<ul style="list-style-type: none">◦ I can only select internal organizations in the store field◦ I can only select ACTIVE Discounts
Create Application	Type: Advanced List Layout: <ul style="list-style-type: none">• This will be an advanced list with no header row. Actions: <ul style="list-style-type: none">• Add a search action for the Store field (hint: this was done in previous applications)
Create App Setting & Menu	<ul style="list-style-type: none">• Add an app setting (and menu) that filters to only show discount offers that have an Active To greater than the current date and time
Create Logic Blocks	<ul style="list-style-type: none">• Name: ValidateDiscountOffers• Table: Validation• Table: DiscountOffers• Requirements:<ul style="list-style-type: none">◦ If the discount is not for all stores, validate that the Store in the Discount and the Store in the DiscountOffer match, otherwise show an error message (use the nsTrnMASTERMsg3 message).• Events - attach to the table as a Pre-Insert / Pre-Update trigger:

Add an App Link from Discounts to Discount Offers	Add a row action application link to your Discounts application. Call the row action "View Discount Offers". The mapping should be to the list view and will filter by nwId (Discount) to Discount (in the DiscountOffers). The purpose of the app link is to provide an easy way to see how many offers are using the discount.
<i>Checkpoint</i>	<ul style="list-style-type: none"> You can create a Discount Offer and attach it to a store If the Discount is marked as ForAllStores = false, if you create the Discount Offer with a matching Store, you can save the record If the Discount is marked as ForAllStores = false, if you create the Discount Offer with a Store that does not match, you cannot save the record and receive an error message In the List View you only see Discount Offers with an Active To greater than the current date and time In Discounts, you can click the "View Discount Offers" row action, it does an app link to Discount Offers and filters to only show offers for that discount.

Section: Orders Application

Context: This is the core application for the project. Now that you have defined stores, ingredients, products, discounts, and discount offers, we can build an application that accepts customer orders.

While it would be ideal to build a full-featured application that handles every possible order scenario, manages discounts based on quantities, and so on, much of that is outside the scope of this project. Our application will be fairly simple. One of the most challenging parts is Discounts. To avoid getting caught up in the complex logic it takes to make that work, discounts will be basic. If a detail line is marked as ApplyStandardDiscount = True, you will call a logic block which returns a 10% discount. There is a **bonus** section to build out the true discount logic.

Task	Details
Create Data Items	<ul style="list-style-type: none"> Notes: <ul style="list-style-type: none"> OrderType <ul style="list-style-type: none"> List Lookup - use the OrderTypeOptions LL you created for module settings LL Type: Single OrderId, OrderNumber - these are Automatic Numbers TimeOfOrder - define it to default to the current time TotalOrderAmount / TotalOrderDiscount <ul style="list-style-type: none"> No negative values

Create Tables	<p>Header</p> <ul style="list-style-type: none"> Attributes: <ul style="list-style-type: none"> Store filters to only show internal orgs Customer filters to only show customers (hint: filter on the ContactRoleGrouping field) Indexes: <ul style="list-style-type: none"> Sort TimeOfOrder (Z - A) OrderNumber is unique <p>Detail</p> <ul style="list-style-type: none"> Requirements <ul style="list-style-type: none"> Product filters to only show ACTIVE products Indexes: <ul style="list-style-type: none"> Can only have a Product <i>once</i> per order (hint: how are detail record tied to the header?) <p>Header/Detail</p> <ul style="list-style-type: none"> Ensure the detail table is added to the detail tables subtable and not the single detail field Add common fields for: <ul style="list-style-type: none"> OrderNumber (header) to OrderNumber (detail). This defaults the value. Store (header) to Store (detail). This defaults the value. We are adding this so we can filter discounts for the specific store. TransactionCurrency (header) to TransactionCurrency (detail). We need it in the detail so we can do currency calculations.
Create Automatic Numbers	<ul style="list-style-type: none"> Create automatic number definition for OrderNumber <ul style="list-style-type: none"> Create automatic number format. Add three parts to the format: <ul style="list-style-type: none"> OrderType field Constant of: “-” Special of: Automatic Number Then attach it to the OrderNumber definition in the Formats page. The format field is OrderId

i When creating the applications, you'll notice that there are many fields that come through as related table lookups fields for the Store and Customer fields and many aren't applicable. To exclude them from showing up in the application, in the table select Type Ahead Only.

Header App

- Layout:
 - These fields should be disabled:
 - Order Number
 - Order Id
 - Total Discount
 - Total Order
 - TransactionCurrency
 - GL Written
 - Two pages: 1) Details and 2) Support
 - The header row should have TimeOfOrder, OrderType, OrderId, and Store
 - The Support page should have Customer, OrderId, TransactionCurrency, GL Written
 - When defining the pages and rows, ensure "Show When Empty" is selected for the Details page
 - Also define a # of columns per row to add a consistent look
- Attributes:
 - Store should have a search action. Use the base search app that already exists. Add a search action:
 - Search Field: Store field
 - Search App: OrganizationalUnitsSearch
 - Setting Name: InternalOrganizationFilter
 - Form: List
 - Data Mappings: the title should say "Search Stores"
 - Customer should also have a search action - search the available search apps over the Directory and pick the customer one. The

Detail App:

- Layout
 - These fields should be disabled:
 - Discount Amount
 - Order Amount
 - List View:
 - In this order: Product, Price, Quantity, ApplyStandardDiscount, DiscountAmount, OrderAmount
 - Field Dependencies:
 - Since we are storing the transaction currency on the record, we can default the currency code for the currency fields. Use Field Dependencies (hint: Field Control) to default the currency. **i** This is a best practice: a transaction has a currency which defaults the code for the currency fields. This prevents users from inserting invalid data.

Create Applications

Section: Order Logic Blocks

Context: The logic blocks will make the app useful and helpful. They will handle defaulting values in the UI, calculations, and perform table validations.

Tasks	Details
Create Order Logic Blocks	<p>Header App:</p> <ul style="list-style-type: none"> Name: OrdersDefault <ul style="list-style-type: none"> Table: Orders (HeaderDetail) Type: Transaction Requirements: <ul style="list-style-type: none"> If Store is not empty, retrieve module settings (BaseSettings) with the Store. Assign the CompanyCurrency from settings to TransactionCurrency. Events: only called in one place (hint: Field Value Change is the action) <p>i When you use the Retrieve Module Settings action, and start from an organizational unit (Organizational Unit Settings), it will first check to see if there is a module setting for that org unit. If none exist for the org unit, the action will traverse the company structure hierarchy. If it doesn't find anything for those, it will retrieve the global setting values.</p> <p>Detail App:</p> <ul style="list-style-type: none"> Name: DefaultProductPrice <ul style="list-style-type: none"> Table: Orders (HeaderDetail) Type: Transaction Requirements: <ul style="list-style-type: none"> Default the product's price to the price field (hint: you will need to fetch the current row) Events: only called in one place Name: CalculateDiscountAmount <ul style="list-style-type: none"> Description: The LB calculates the discount amount for the line. This uses the StandardDiscount in module settings. Type: Transaction Table: Orders (HeaderDetail) Requirements: <ul style="list-style-type: none"> Calculate the discount amount based on the order amount and the standard discount amount in module settings Updates the order amount field (amount - discount = amount) Clears the discount if the discount is no longer being applied Resets the order amount if the discount is no longer being applied Set an error if a standard discount is being applied and has not been defined in module settings Name: CalculateOrderAmount <ul style="list-style-type: none"> Description: The LB is used to calculate the order amount whenever changes are made that would modify it. Type: Transaction Table: Orders (HeaderDetail) Requirements: <ul style="list-style-type: none"> If the row has the values needed, calculate the order amount Make sure the order amount does not include the discount amount (order amount - discount amount = order amount)

	<ul style="list-style-type: none"> Events: this LB is called on Field Value Changed of three different fields <p>Header - Table Trigger</p> <ul style="list-style-type: none"> Name: CalculateAmountsTrigger <ul style="list-style-type: none"> Description: We want to display the order total and discount total in the List View of the app. The List View is over the header table so we need to update those fields with the totals. Table: OrdersHeader Requirements: <ul style="list-style-type: none"> Update the total order amount and total order discount with the amounts in the order details Events: <ul style="list-style-type: none"> Add to the events that make sense <p>Detail - Trigger:</p> <ul style="list-style-type: none"> For this project's scope, you don't need to add any. However, as best practice, the defaulting and validations we do in the UI should also happen in a trigger. Since every table is an API endpoint, you can't assume data is coming in from the UI.
Checkpoint	<ul style="list-style-type: none"> You can create an order with different types For the detail lines, the Price is defaulted when the product changes For the detail lines, if ApplyStandardDiscount = true, the DiscountAmount is equal to 10% of the OrderAmount If Apply Standard Discount is removed, the discount amount is cleared and the order amount is updated For the detail lines, the order amount is equal to $(price * quantity) - order\ discount$ The total amount and total discount in the header are updated when you save the record

Section: Order Workflow

Context: As a final step for the core application, you will add workflow. The workflow will let stores process each order type through a different business flow. For example, a delivery order has more steps than a dine-in order.


Tasks	Details
Create Orders Workflow	<ul style="list-style-type: none"> Create Workflow Definition <ul style="list-style-type: none"> State Lookup: nsTrnMASTER Orders Workflow Status Table: Your Orders table State Type Lookup: nsTrnMASTER Orders Workflow Type Details - see the workflow definition steps in the appendix Create logic block: <ul style="list-style-type: none"> Name: OrdersWorkflowProcessor <ul style="list-style-type: none"> Table: Your OrdersHeader table Don't need to add any logic, drag in a comment action as a placeholder.


	<ul style="list-style-type: none"> • Define Workflow Definition. Since Workflow Types are used, there are steps involved: <ul style="list-style-type: none"> ◦ Create the sequences 100 - 700 ◦ Create the Workflow Types - see the workflow types in the appendix ◦ Complete the Workflow Definition with the workflow types • Mark the workflow as Active and Production
--	--

Advanced II Section

Section: Write GL Transaction

Context: Every financial transaction ends up in the GL. In a full implementation, we would properly account for cash received, credit cards, processing fees, etc. For now, we are going to bypass that and just create the GL transaction, assuming that customers paid with cash. In Nextworld, the logic block that controls inserts and updates to the GL is WriteGLTransaction. In order to properly call the logic block, we will need to retrieve information from several places. If all goes well, transactions will be created in the GL and can be viewed in the Journal Entries application.

Task	Details
Writing to GL	<ul style="list-style-type: none"> • Preparation Steps: <ol style="list-style-type: none"> 1. If you didn't do it in a previous step, create a Cash account for your company (see the Directory Apps & Setup section) 2. Create a module setting rule, selecting the primary cash account for your company (see the Directory Apps & Setup section) <ul style="list-style-type: none"> ◦ Create a fiscal calendar for company (Fiscal Calendars on menu). Make sure you run the <i>Generate Calendar</i> row action. ◦ Create a rule in GL Mapping Rules: <ul style="list-style-type: none"> ▪ Type: Revenue ▪ Org Unit: the company you created ▪ Item Class: leave empty ▪ GL Account: 41000 • Create logic block <ul style="list-style-type: none"> ◦ Name: WriteOrderGL ◦ Table: OrdersHeader • Core Logic: <ul style="list-style-type: none"> ◦ Create two variables: <ul style="list-style-type: none"> ▪ OrderDate (date) ▪ TotalOrderAmt (currency)  remember how to default a currency variable? ◦ Create HD Structure ◦ Call LB: RetrieveCompanyForOrgUnit. Pass in parameters: <ul style="list-style-type: none"> ▪ OrganizationalUnit = Store ◦ Retrieve GeneralAccountingSettings (to get the Primary Ledger) <ul style="list-style-type: none"> ▪ Type: Organizational Unit Settings ◦ Retrieve your Pizza Setting (to get Primary Cash Account) <ul style="list-style-type: none"> ▪ Type: Organizational Unit Settings

	<ul style="list-style-type: none"> ○ Use DateTime Decomposition for TimeOfOrder to store the date in the OrderDate variable ○ Insert header <ul style="list-style-type: none"> ▪ Company = Company (retrieve company call LB) ▪ Ledger = PrimaryLedger (GeneralAccountingSettings) ▪ TransactionCurrency = TransactionCurrency (order) ▪ TransactionDate = OrderDate variable ▪ TransactionType = Journal Entry ▪ HeaderMemo (optional) ○ Fetch all details for the order to calculate the total order amount. Assign to TotalOrderAmt variable ○ Call LB: DeriveGLAccount (for Revenue account). Pass in parameters: <ul style="list-style-type: none"> ▪ EntryType = Revenue ▪ OrganizationalUnit = Store ○ Insert detail for Cash <ul style="list-style-type: none"> ▪ GLAccount = PrimaryCashAccount from pizza settings ▪ GLLineNumber = 1 ▪ OrganizationalUnit = Store ▪ TransactionCurrency = TransactionCurrency ▪ TransactionDebitAmount = OrderAmount ○ Insert detail for Revenue <ul style="list-style-type: none"> ▪ GLAccount = GLAccount from called LB DeriveGLAccount ▪ GLLineNumber = 2 ▪ OrganizationalUnit = Store ▪ TransactionCurrency = TransactionCurrency ▪ TransactionCreditAmount = OrderAmount ○ Call the Write GL Transaction logic block <p> For the LB calls, use nsTrnMASTERMsg10 in the On Error section.</p>
Add a call of WriteOrderGL into the process	<p>We need to call the LB from somewhere. Where does it make sense to call it from?</p> <p>Hint: Before the workflow can move to Complete, we need to create the GL.</p>
Update GLWritten in the OrdersHeader	<p>After the GL is successfully created, we need to indicate that on the OrdersHeader. Add logic to update the GLWritten field.</p>
Checkpoint	<ul style="list-style-type: none"> • If the transaction fails, workflow goes to Error state. User can then manually resubmit. • If the transaction is successful, a Journal Entry is created. <ul style="list-style-type: none"> ○ The cash account matches what is in module settings ○ The revenue account matches the GL Mapping Rule ○ The detail amounts match the order amount from the order ○ The detail org units match the Store provided in the order ○ The transaction date matches the date in the order

Section: Batch Processing (Create Discount Offers)

Context: Right now it's a manual process to create a discount offer for each store if you want a discount to be available at each store. We can automate this. The batch process will create the discount offers via a batch process where the offers are either created for a single store or for all stores depending on how it is defined in the discount. To accomplish this, you will build an application which is accessed from the Discounts app via a row action.

Task	Details
CreateDiscountOffersLB Table	<ul style="list-style-type: none"> • Name: CreateDiscountOffersLB • Table Type: Work Table • Fields (all of them should already exist): <ul style="list-style-type: none"> ◦ Discount (TL) ◦ ActiveFrom ◦ ActiveTo
CreateDiscountOffersLB Application	<ul style="list-style-type: none"> • Name: CreateDiscountOffersLB • Type: Standard • Style: Mini App • Detail Layout: <ul style="list-style-type: none"> ◦ Put ActiveFrom and ActiveTo in the same row. Give Discount a row and column but mark it as "hidden" • Requirements: <ul style="list-style-type: none"> ◦ In the detail view, the Save button is hidden (hint: Control Actions)
Create Logic Block	<ul style="list-style-type: none"> • Name: CreateDiscountOffers • Purpose: It inserts Discount Offers for all stores or a single store based on how the Discount is defined • Type: Background Task • Table: CreateDiscountOffersLB • Requirements <ul style="list-style-type: none"> ◦ Set an error if either ActiveFrom or ActiveTo is missing ◦ Set an error if the discount is not active ◦ If the discount is for all stores, insert a discount offer for each store ◦ If the discount is for a single store, insert a discount for the single store ◦ If the discount and store combination already exist, update the active from and active to
Add LB as Action Block	<ul style="list-style-type: none"> • Attach CreateDiscountOffers as an Insert Action Block

Create app link <i>from</i> Discounts to show the CreateDiscountOffersLB application	<ul style="list-style-type: none"> The app link should: <ul style="list-style-type: none"> Go to the detail view Pass in the nwId of the Discount to the Discount field
<i>Checkpoint</i>	<ul style="list-style-type: none"> If a discount is for all stores, a discount offer is created for all stores If a discount is for a single store, a discount offer is created for the single store If a discount offer already exists for the discount and store combination, active from and active to are updated

Section: Reports

Context: Corporate and Franchise owners may want to see how well stores are performing, what products are selling well and how various discount campaigns performed. To accomplish this, you will build a reports summarizing data in a pdf format.

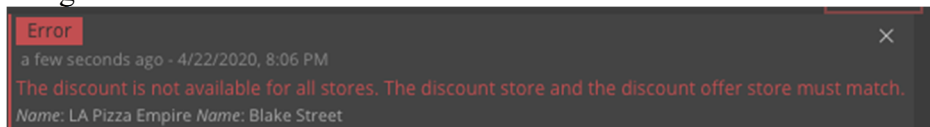
- Order Breakdown Report
 - For a date range (selected by user)
 - By store, show the following:
 - Total number of orders
 - Average order size - total order amount
 - Average discount per order
- Product Breakdown Report
 - For a data range (selected by user)
 - How many times was each product ordered

Bonus Section

Congratulations on completing the required parts of the capstone! It is uncommon for a project to be fully complete, so we have a bonus section if you want to would like to add to the existing functionality.

Bonus Section: Validations

- Add a validation to orders workflow processor logic block - set an error if there are no order detail rows.
- Add a validation to the Discounts application - set an error if a discount percent is not provided and the discount type is percent. Implement the same logic for the discount amount if the discount type is amount.
- In the trigger for DiscountOffers, add inputs to the nsTrnMASTERMsg3 set message that tells you the name of the store in the discount. This lets the user see the store that is expected while looking at the error message



- In the trigger for DiscountOffers, validate that Active To must be greater than Active From

Bonus Section: Add Memo to GL

- When creating a GL Transaction, concatenate “ORDER: ” and the OrderId and put it in the HeaderMemo

Bonus Section: Integrate Discounts in Orders

Right now we are not integrating Discounts and Discount Offers into the Orders application. We are using the Standard Discount module setting. Simply being able to define discounts is not enough, you need to be able to offer them to customers. These are the requirements for integrating with discounts:

- Be able to pick a discount in an order detail line. Should there be any filters for this?
- Create a logic block that:
 - Validates that the discount is valid based on the rest of the order line
 - Calculates the discount amount based on the discount type
 - Returns the discount amount
- Recalculate the order amount

Bonus Section: Order Balances

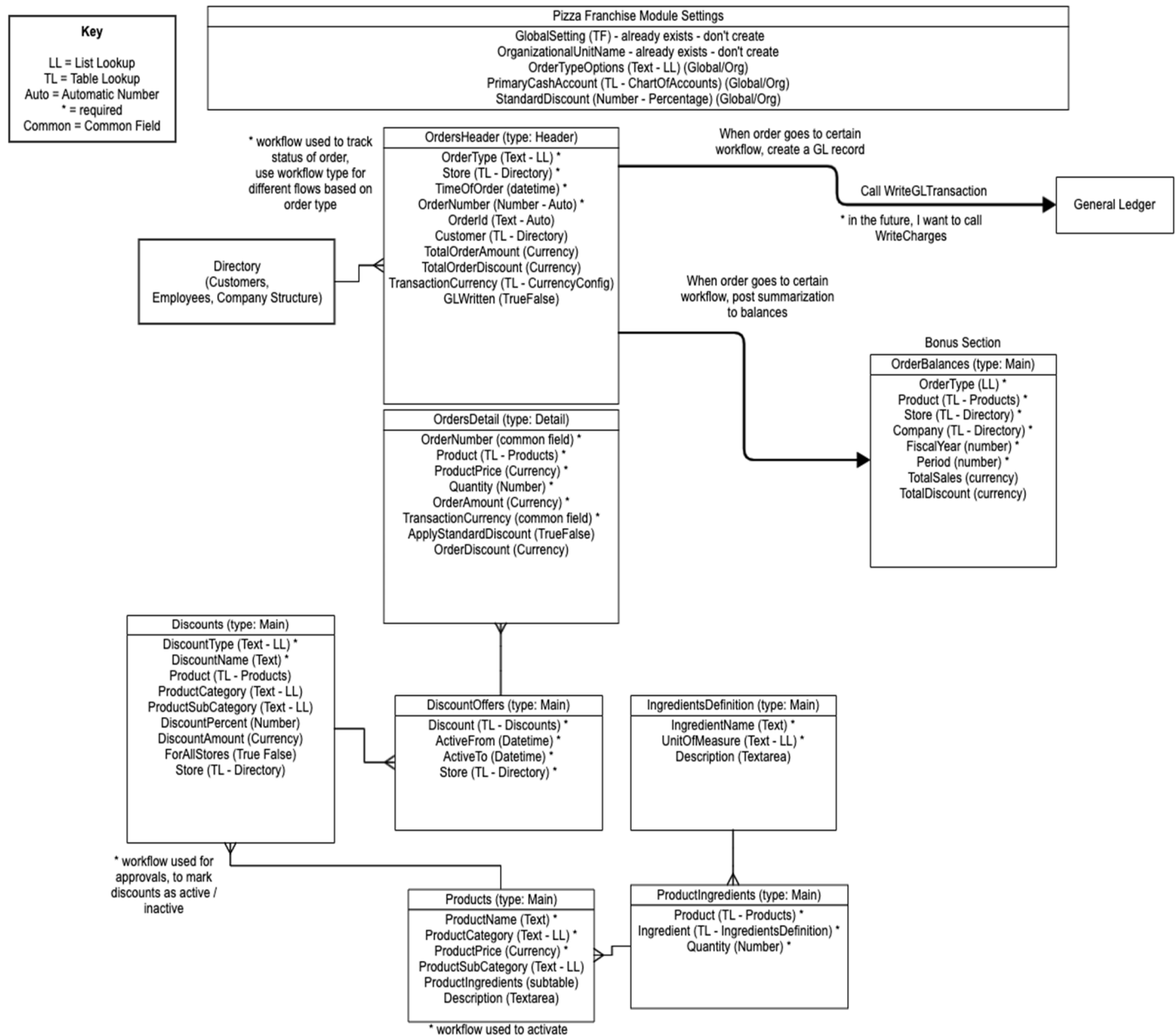
We need a way to see summarized order information, primarily TotalSales and TotalDiscounts. We want an inquiry screen for this information. Create the table, application, and logic block which accomplishes this. What should the logic block be? Where should this logic block be called from? See the Data Model for the fields.

Bonus Section: External Callout

Task	Details
External Callout	Not in scope

Capstone Data Model and Workflows

Data Model



Workflows

Products

Sequence	Workflow Type	Trigger	Duration	From State	From State Type	Condition / Logic Block	To State	To State Type
100		Manual		New	Initial		Active	Active
200		Manual		Active	Active		Inactive	Closed
300		Manual		Inactive	Closed		Active	Active

Discounts

Sequence	Workflow Type	Trigger	Duration	From State	From State Type	Condition / Logic Block	To State	To State Type	Action	Action Details
100		Manual		New	Initial		Active	Active		
200		Manual		Active	Active		Inactive	Closed		
300		Manual		Inactive	Closed		Active	Active		

Orders

Sequence	Workflow Type	Trigger	Duration	From State	From State Type	Condition / Logic Block	To State	To State Type
100		Manual		New	Initial		Kitchen	Kitchen
200		Manual		New	Initial		Finish Order	Active
300		Manual		Ready For Pickup	Active		Finish Order	Active
400		Auto Transition	Instant	Finish Order	Active	LB Success / OrderWorkflowProcessor	Complete	Closed
500		Auto Transition	Instant	Finish Order	Active	LB Error / OrderWorkflowProcessor	Error	Active

600		Manual		Out For Delive ry	Delive ry		Finish Order	Active
700		Manual		Error	Active		Finish Order	Active
800	Delivery Order	Manual		Kitche n	Kitche n		Out For Deliver y	Delivery
900	TakoutO rder	Manual		Kitche n	Kitche n		Ready For Pickup	Active
1000	CaterOr der	Manual		New	Initial		Waitin g Until Date	Active
1100	CaterOr der	Manual		Waitin g Until Date	Active		Kitche n	Kitchen
1200	CaterOr der	Manual		Kitche n	Kitche n		Out For Deliver y	Delivery
1300	CaterOr der	Manual		Kitche n	Kitche n		Ready For Pickup	Active