ASP.NET Dev Camp - Module 2

Expenses App Scenario Overview

Lab version: 1.0

Last updated: 1/13/2015

Estimated demo delivery time: **10 minutes**.



Contents

[Overview 3](#_Toc408919276)

[Prerequisites 3](#_Toc408919277)

[Setup 3](#_Toc408919278)

[Exercise 1: Expenses App Scenario Overview 4](#_Toc408919279)

[Task 1: Expenses Application Demo 4](#_Toc408919280)

[Task 2: Expenses Data Layer Overview 9](#_Toc408919281)

[Task 3: Expenses ASP.NET MVC Project Overview 12](#_Toc408919282)

# Overview

In this demo, we will introduce the Expenses application scenario and implementation to provide a concrete context for additional demos to follow.

# Prerequisites

The following are required to complete this demo:

* [Microsoft Visual Studio 2013](http://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx) (tested with Update 4)
* Expenses.Mvc codebase

# Setup

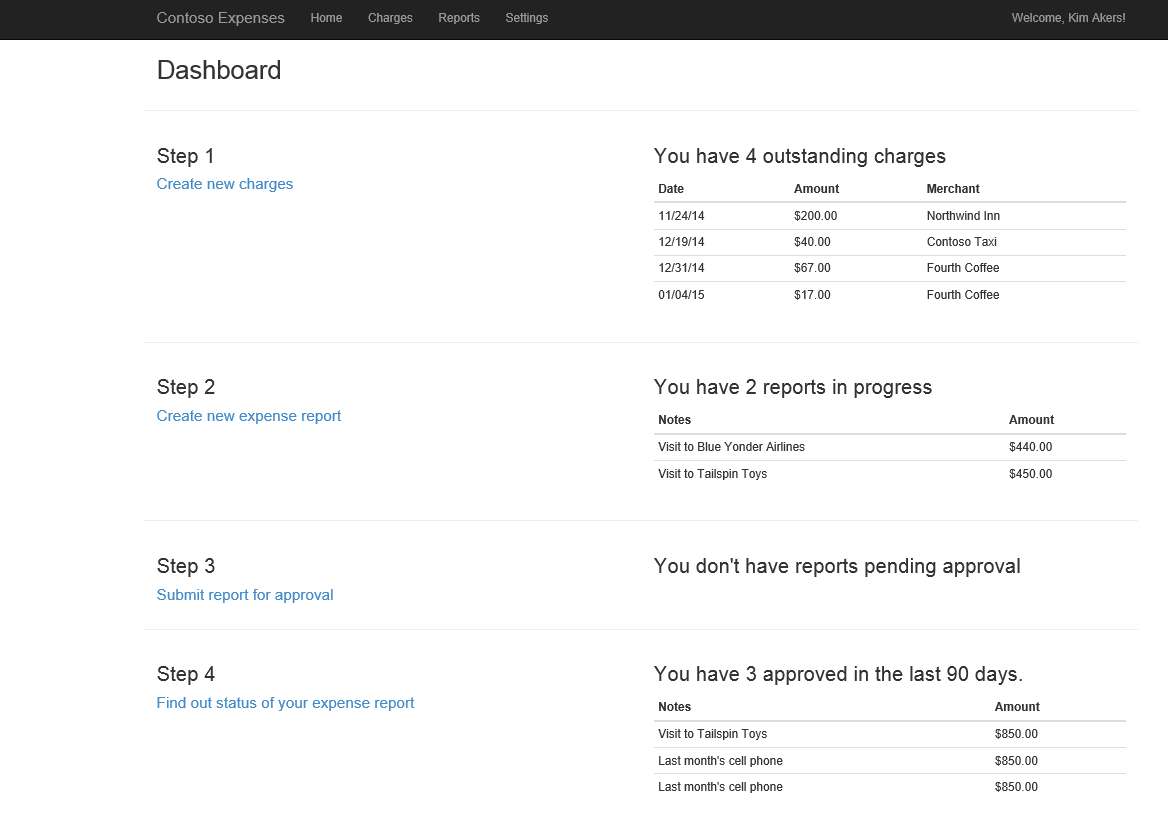
* Load and build the Expenses.Mvc solution to ensure that it builds correctly. Note that the first build requires Internet connectivity in order to download needed NuGet packages.

# Exercise 1: Expenses App Scenario Overview

In this exercise, we will start by introducing the Expenses ASP.NET MVC application scenario and implementation.

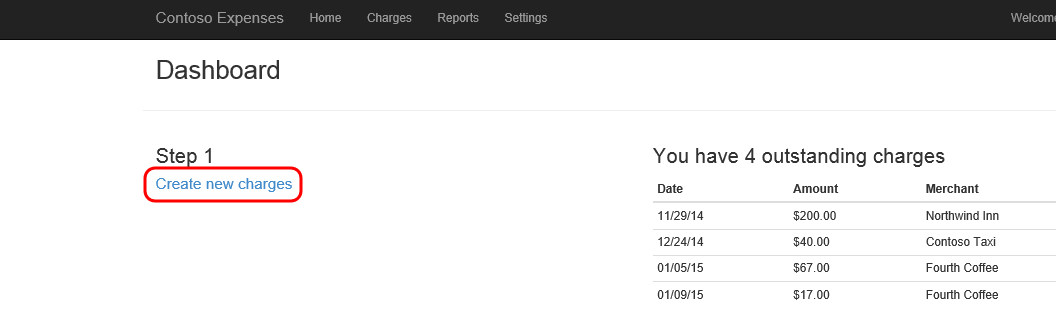
## Task 1: Expenses Application Demo

1. Load the Expenses.Mvc.sln solution in Visual Studio.
2. Set the Expenses.Web project as the startup project (right-click and select Set as StartUp Project).
3. Press F5 to the Expenses app. The app will be hosted locally in IIS Express. The first page that you should see is the Dashboard which shows outstanding charges, reports that are in various stages of the approval process.

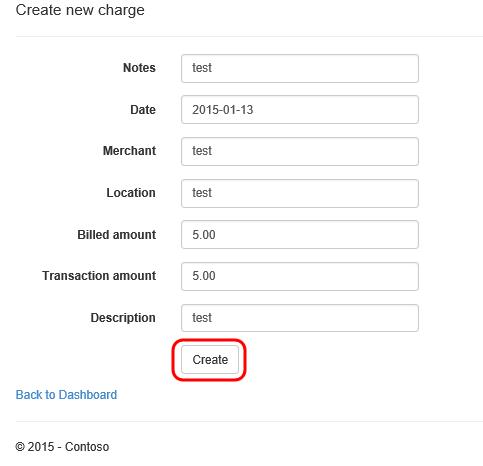


**Note:** This demonstration application does not actually provide login support, and thus the logged in user that is shown in the upper-right corner is simulated.

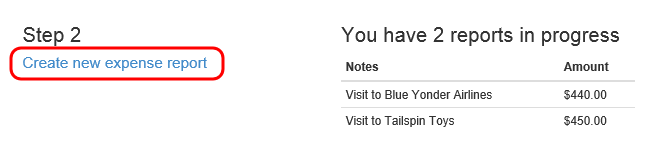
1. From the Dashboard, click the “Create new charges” link.



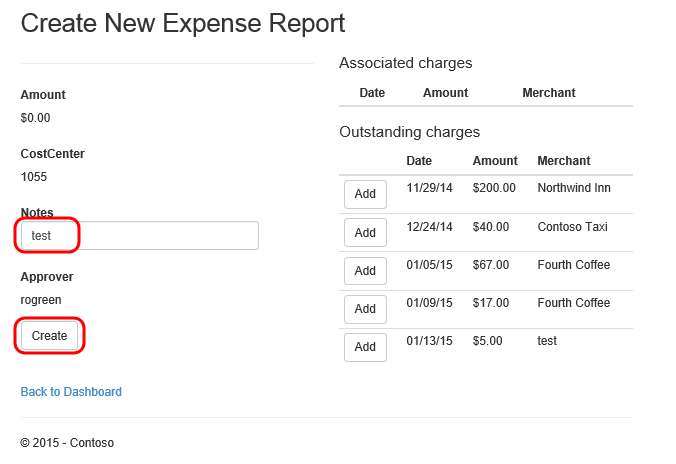
1. Provide the required fields for the new charge and then click Create.



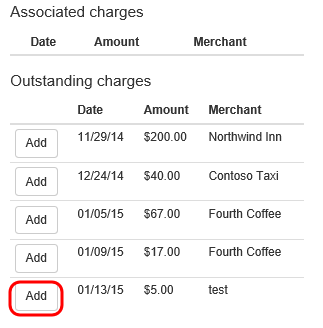
1. From the Dashboard, click the “Create new expense report” link.



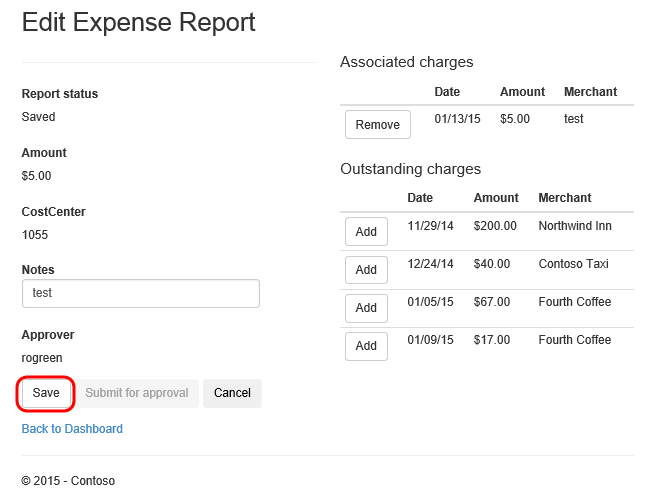
1. Provide some text in the Notes field and then click the Create button.



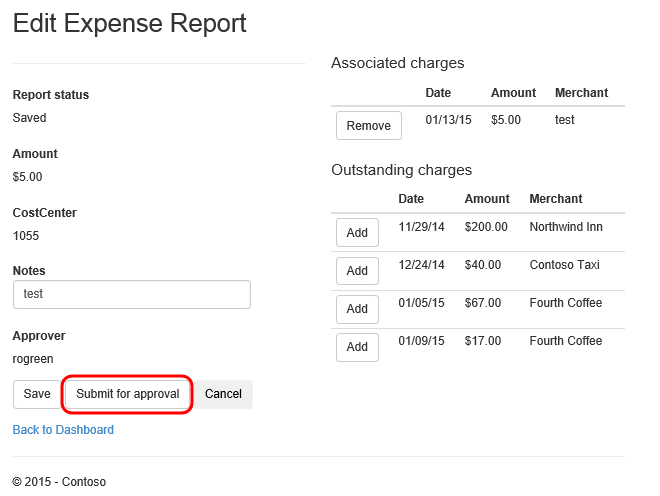
1. Click the Add button for the new charge in the Outstanding Charges section to associate it with the new expense report.



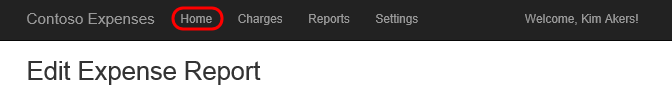
1. Click the Save button.



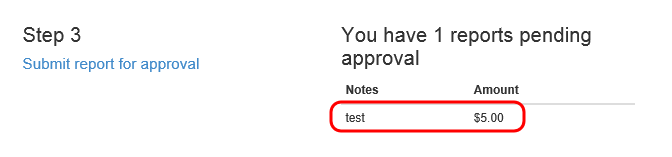
1. In this demo scenario, the next step in the workflow would be to submit the report to a manager for approval. Click the Submit for Approval button.



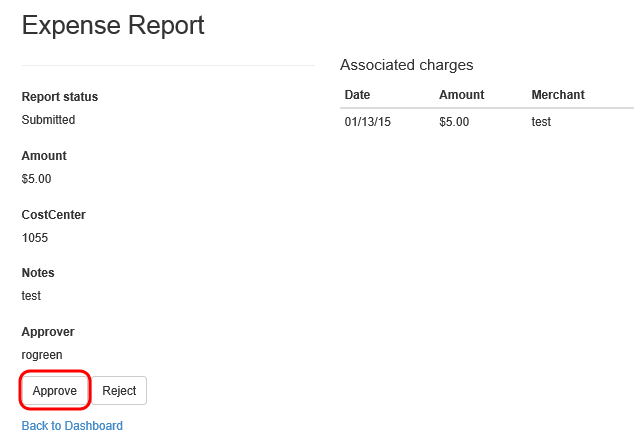
1. Return to the home view.



1. The final step in the expense report workflow is to get manager approval. Click on the report you created, which should now be at Step 3.



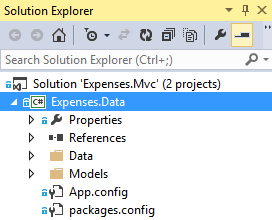
1. Click the Approve button (suspending disbelief momentarily and pretending you are now logged in as a manager).



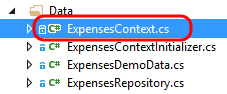
1. Close the browser window and return to Visual Studio.

## Task 2: Expenses Data Layer Overview

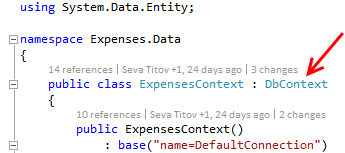
1. In Solution Explorer, expand the Expenses.Data project. The Data project defines a data layer that is used directly by the ASP.NET site.



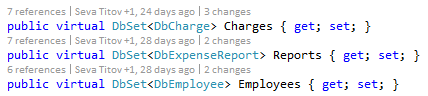
1. Open the ExpensesContext.cs file from the Data folder.



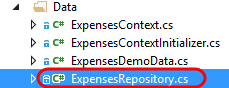
1. The ExpensesContext class uses Entity Framework Code First to define a data model that is persisted to SQL Server. Here you can see that the class derives from DbContext.



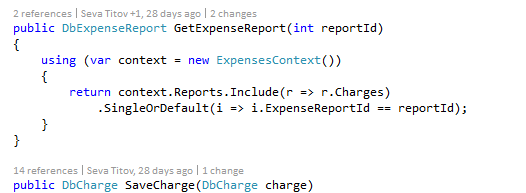
1. Scroll down and locate the DbSet definitions for Charges, Reports, and Employees.



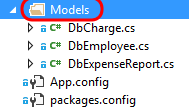
1. Open the ExpensesRepository.cs file from the Data folder.



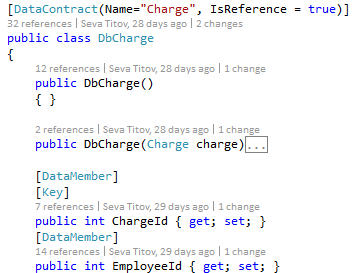
1. The ExpenseRepository class defines APIs for accessing the data layer, in order to provide typical CRUD operations against the database.



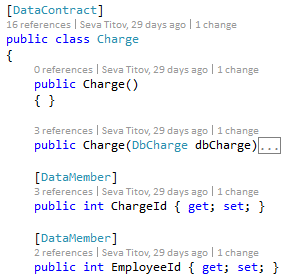
1. Expand and view the files located in the Models folder.



1. The Models folder contains the model objects that define the data model using Entity Framework. Open the DbCharge.cs file and take a look at the definition.

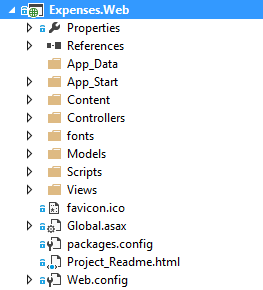


1. The CommunicationModels folder contains the model objects that are actually serialized and transferred over the wire back to client of the WCF service. These types of model objects are commonly referred to as Data Transfer Objects (DTOs). Open the Charge.cs file and take a look at the definition.



## Task 3: Expenses ASP.NET MVC Project Overview

1. Expand the Expenses.Web project node.



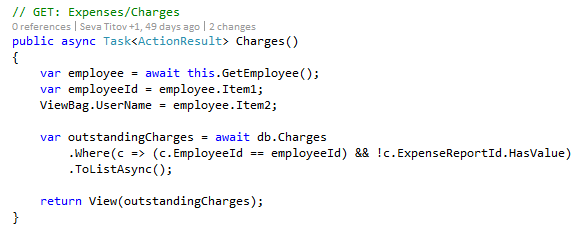
1. The Expenses ASP.NET web application is created using the MVC framework, which stands for Model-View-Controller. Open ExpensesController.cs from the Controllers folder in the editor.



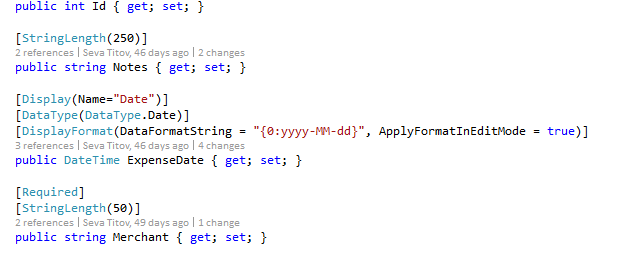
1. Note that this controller makes use of the data layer by making use of the ExpensesContext class.



1. Note that the scaffolding feature of Visual Studio was used to initially create this controller. Take a look at the Charges method, which is an asynchronous method that returns all the outstanding charges for the logged in user using the appropriate view.



1. Open ChargeModel.cs from the Models folder. Scroll through the code to take a look at a prototypical model class used in ASP.NET MVC. Note the public properties that use various data annotations to help validate data input from the user.



1. Open Charges.cshtml from the Views | Expenses folder. This view is tied to a list of charge model objects and makes use of the Razor syntax to define the creation of the needed HTML markup.

