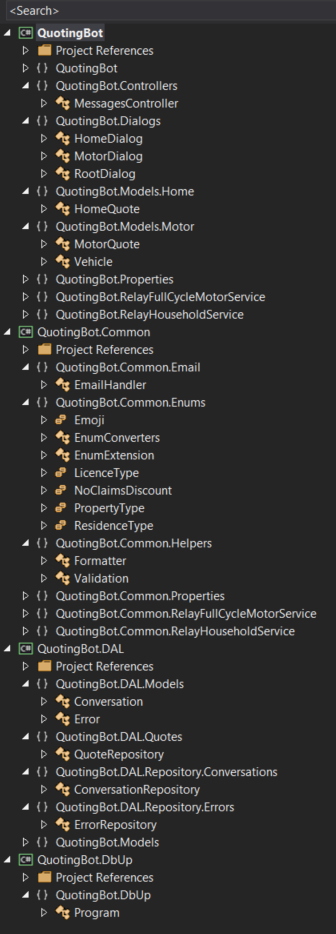
# Code Listing

## Code Dictionary



## Code Listing

### MessagesController.cs

using System.Net;

using System.Net.Http;

using System.Threading.Tasks;

using System.Web.Http;

using Microsoft.Bot.Builder.Dialogs;

using Microsoft.Bot.Connector;

using QuotingBot.Dialogs;

namespace QuotingBot.Controllers

{

[BotAuthentication]

public class MessagesController : ApiController

{

public async Task<HttpResponseMessage> Post([FromBody]Activity activity)

{

if (activity.Type == ActivityTypes.Message)

{

await Conversation.SendAsync(activity, () => new RootDialog());

}

else

{

HandleSystemMessage(activity);

}

var response = Request.CreateResponse(HttpStatusCode.OK);

return response;

}

private Activity HandleSystemMessage(Activity message)

{

switch (message.Type)

{

case ActivityTypes.DeleteUserData:

// Implement user deletion here

// If we handle user deletion, return a real message

break;

case ActivityTypes.ConversationUpdate:

// Handle conversation state changes, like members being added and removed

// Use Activity.MembersAdded and Activity.MembersRemoved and Activity.Action for info

// Not available in all channels

break;

case ActivityTypes.ContactRelationUpdate:

// Handle add/remove from contact lists

// Activity.From + Activity.Action represent what happened

break;

case ActivityTypes.Typing:

// Handle knowing tha the user is typing

break;

case ActivityTypes.Ping:

break;

}

return null;

}

}

}

### HomeDialog.cs

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Globalization;

using System.Threading.Tasks;

using System.Web.Script.Serialization;

using Microsoft.Bot.Builder.Dialogs;

using Microsoft.Bot.Builder.FormFlow;

using Microsoft.Bot.Connector;

using QuotingBot.Common.Email;

using QuotingBot.Common.Enums;

using QuotingBot.DAL.Quotes;

using QuotingBot.DAL.Repository.Conversations;

using QuotingBot.DAL.Repository.Errors;

using QuotingBot.Common.Helpers;

using QuotingBot.Models.Home;

using QuotingBot.Common.RelayHouseholdService;

namespace QuotingBot.Dialogs

{

[Serializable]

public class HomeDialog : IDialog<object>

{

private readonly Validation \_validation = new Validation();

private static readonly bool SendEmails = Convert.ToBoolean(ConfigurationManager.AppSettings["SendEmails"]);

private static readonly string Connection = ConfigurationManager.ConnectionStrings["QuotingBot"].ConnectionString;

private static readonly ErrorRepository ErrorRepository = new ErrorRepository(Connection);

public async Task StartAsync(IDialogContext context)

{

await context.PostAsync($"No worries - let's do it {Emoji.GrinningFace.GetDescription()}");

var homeQuoteFormDialog = FormDialog.FromForm(this.BuildHomeQuoteForm, FormOptions.PromptInStart);

context.Call(homeQuoteFormDialog, ResumeAfterHomeQuoteFormDialog);

}

private IForm<HomeQuote> BuildHomeQuoteForm()

{

OnCompletionAsyncDelegate<HomeQuote> getHomeQuotes = async (context, state) =>

{

await context.PostAsync("Getting your quotes...");

};

return new FormBuilder<HomeQuote>()

.Field(nameof(HomeQuote.FirstLineOfAddress))

.Field(nameof(HomeQuote.Town),

validate: async (state, value) => \_validation.ValidateTown(value))

.Field(nameof(HomeQuote.County),

validate: async (state, value) => \_validation.ValidateCounty(value))

.AddRemainingFields()

.Field(nameof(HomeQuote.NumberOfBedrooms),

prompt: "How many bedrooms are in the property? (0-9 bedrooms)",

validate: async (state, value) => \_validation.ValidateNumberOfBedrooms(value))

.Field(nameof(HomeQuote.YearBuilt),

validate: async (state, value) => \_validation.ValidateYearBuilt(value))

.Field(nameof(HomeQuote.FirstName),

validate: async (state, value) => \_validation.ValidateFirstName(value))

.Field(nameof(HomeQuote.LastName),

validate: async (state, value) => \_validation.ValidateLastName(value))

.AddRemainingFields()

.Field(nameof(HomeQuote.EmailAddress),

validate: async (state, value) => \_validation.ValidateEmailAddress(value))

.Confirm("Do you want to request a quote using the following details?\n\n" +

"\n\n" +

"Address: {FirstLineOfAddress}, {Town}, {County}\n\n" +

"Property type: {PropertyType}\n\n" +

"Residence type: {ResidenceType}\n\n" +

"Year built: {YearBuilt}\n\n" +

"No.of bedrooms: {NumberOfBedrooms}\n\n" +

"Name: {FirstName} {LastName}\n\n" +

"Contact number: {PrimaryContactNumber}\n\n" +

"Email: {EmailAddress}")

.OnCompletion(getHomeQuotes)

.Build();

}

private static async Task ResumeAfterHomeQuoteFormDialog(IDialogContext context, IAwaitable<HomeQuote> result)

{

var state = await result;

try

{

var quoteRepository = new QuoteRepository(Connection);

var conversationRepository = new ConversationRepository(Connection);

var reply = context.MakeMessage();

var homeService = new Household();

var homeWebServiceRequest = HomeQuote.BuildHomeWebServiceRequest(state);

var quotes = new List<HomeQuoteWebServiceResult>();

var response = homeService.GetQuotes(homeWebServiceRequest);

if (response.Quotes != null)

{

foreach (var quote in response.Quotes)

{

quotes.Add(quote);

}

reply.AttachmentLayout = AttachmentLayoutTypes.Carousel;

reply.Attachments = GetQuoteReceipts(quotes);

quoteRepository.StoreQuote

(

context.Activity.Conversation.Id,

response.Quotes[0].RelayQuoteId,

new JavaScriptSerializer().Serialize(quotes)

);

}

else

{

reply.Text = "Sorry, we couldn't get any quotes for you.";

}

await context.PostAsync(reply);

if (SendEmails)

{

var emailBodyForUser = EmailHandler.BuildHomeEmailBodyForUser(response.Quotes, state.FirstName, state.LastName, state.PrimaryContactNumber, state.EmailAddress,

state.FirstLineOfAddress, state.Town, state.County, state.PropertyType.GetDescription(), state.ResidenceType.GetDescription(), state.YearBuilt,

state.NumberOfBedrooms.ToString());

EmailHandler.SendEmailToUser(state.EmailAddress, $"{state.FirstName} {state.LastName}", emailBodyForUser);

var emailBodyForBroker = EmailHandler.BuildHomeEmailBodyForBroker(response.Quotes, state.FirstName, state.LastName, state.PrimaryContactNumber, state.EmailAddress,

state.FirstLineOfAddress, state.Town, state.County, state.PropertyType.GetDescription(), state.ResidenceType.GetDescription(), state.YearBuilt,

state.NumberOfBedrooms.ToString());

EmailHandler.SendEmailToBroker(state.EmailAddress, $"{state.FirstName} {state.LastName}", emailBodyForBroker);

}

conversationRepository.StoreConversation

(

context.Activity.Conversation.Id,

context.Activity.From.Id,

DateTime.Now.ToString(new CultureInfo("en-GB")),

new JavaScriptSerializer().Serialize(context)

);

}

catch (Exception exception)

{

var errorRepository = new ErrorRepository(Connection);

errorRepository.LogError(context.Activity.Conversation.Id, context.Activity.From.Id, DateTime.Now.ToString(), context.ConversationData.ToString(), exception.ToString());

throw;

}

finally

{

context.Done(state);

}

}

private static IList<Attachment> GetQuoteReceipts(List<HomeQuoteWebServiceResult> homeQuoteWebServiceResults)

{

var cards = new List<Attachment>();

foreach (var result in homeQuoteWebServiceResults)

{

if (result.NetPremium > 0)

{

cards.Add(GetReceiptCard(result));

}

}

return cards;

}

private static Attachment GetReceiptCard(HomeQuoteWebServiceResult homeQuoteWebServiceResult)

{

try

{

var receiptCard = new ReceiptCard

{

Title = $"{homeQuoteWebServiceResult.InsurerName}",

Facts = new List<Fact> { new Fact("Scheme", homeQuoteWebServiceResult.SchemeName) },

Tax = $"€{homeQuoteWebServiceResult.GovernmentLevyPremium}",

Total = $"€{homeQuoteWebServiceResult.NetPremium}",

Buttons = new List<CardAction>

{

new CardAction

(

ActionTypes.PostBack,

"Request a Callback"

)

}

};

return receiptCard.ToAttachment();

}

catch (Exception exception)

{

ErrorRepository.LogError(DateTime.Now.ToString(new CultureInfo("en-GB")), exception.ToString());

throw;

}

}

}

}

### MotorDialog.cs

using Microsoft.Bot.Builder.Dialogs;

using Microsoft.Bot.Builder.FormFlow;

using System;

using System.Threading.Tasks;

using QuotingBot.Models.Motor;

using System.Web.Script.Serialization;

using QuotingBot.DAL.Quotes;

using QuotingBot.DAL.Repository.Errors;

using Microsoft.Bot.Connector;

using System.Collections.Generic;

using System.Configuration;

using System.Globalization;

using System.Linq;

using QuotingBot.Common.Helpers;

using QuotingBot.Common.Email;

using QuotingBot.Common.Enums;

using QuotingBot.DAL.Repository.Conversations;

using QuotingBot.Common.RelayFullCycleMotorService;

namespace QuotingBot.Dialogs

{

[Serializable]

public class MotorDialog : IDialog<MotorQuote>

{

private readonly Validation \_validation = new Validation();

private static readonly bool SendEmails = Convert.ToBoolean(ConfigurationManager.AppSettings["SendEmails"]);

private static readonly string Connection = ConfigurationManager.ConnectionStrings["QuotingBot"].ConnectionString;

private static readonly ErrorRepository ErrorRepository = new ErrorRepository(Connection);

public async Task StartAsync(IDialogContext context)

{

await context.PostAsync("No problem!");

await context.PostAsync($"Let's get started {Emoji.GrinningFace.GetDescription()}");

var motorQuoteFormDialog = FormDialog.FromForm(this.BuildMotorQuoteForm, FormOptions.PromptInStart);

context.Call(motorQuoteFormDialog, this.ResumeAfterMotorQuoteFormDialog);

}

private IForm<MotorQuote> BuildMotorQuoteForm()

{

OnCompletionAsyncDelegate<MotorQuote> getMotorQuotes = async (context, state) =>

{

await context.PostAsync("Getting your quotes...");

};

return new FormBuilder<MotorQuote>()

.Field(nameof(MotorQuote.VehicleRegistration),

validate: async (state, value) =>

{

var result = new ValidateResult();

state.Vehicle = MotorQuote.GetVehicle(value.ToString());

if (!string.IsNullOrEmpty(state.Vehicle.Description))

{

result.IsValid = true;

result.Value = value.ToString().ToUpper();

result.Feedback = state.Vehicle.Description;

}

else

{

result.IsValid = false;

result.Feedback = $"Hmmm...I couldn't find a match for that registration {Emoji.ThinkingFace.GetDescription()} Please try another registration";

}

return result;

}

)

.Confirm(generateMessage: async (state) => new PromptAttribute("Is this your car?"))

.Field(nameof(MotorQuote.VehicleValue),

validate: async (state, value) => \_validation.ValidateVehicleValue(value))

.Field(nameof(MotorQuote.AreaVehicleIsKept),

validate: async (state, value) => \_validation.ValidateAreaVehicleIsKept(value))

.Field(nameof(MotorQuote.FirstName),

validate: async (state, value) => \_validation.ValidateFirstName(value))

.Field(nameof(MotorQuote.LastName),

validate: async (state, value) => \_validation.ValidateLastName(value))

.Field(nameof(MotorQuote.DateOfBirth),

prompt: "What is your date of birth? Enter date in DD/MM/YYYY format please",

validate: async (state, value) => \_validation.ValidateDateOfBirth(value))

.AddRemainingFields()

.Field(nameof(MotorQuote.EmailAddress),

validate: async (state, value) => \_validation.ValidateEmailAddress(value))

.Confirm("Do you want to request a quote using the following details?\n\n" +

"\n\n" +

"Car registration: {VehicleRegistration}\n\n" +

"Vehicle value: {VehicleValue}\n\n" +

"Area vehicle is kept: {AreaVehicleIsKept}\n\n" +

"Name: {FirstName} {LastName}\n\n" +

"Date of birth: {DateOfBirth}\n\n" +

"Licence Type: {LicenceType}\n\n" +

"No claims Discount: {NoClaimsDiscount} years\n\n" +

"Contact number: {PrimaryContactNumber}\n\n" +

"Email: {EmailAddress}")

.OnCompletion(getMotorQuotes)

.Build();

}

private async Task ResumeAfterMotorQuoteFormDialog(IDialogContext context, IAwaitable<MotorQuote> result)

{

var state = await result;

var reply = context.MakeMessage();

try

{

var quoteRepository = new QuoteRepository(Connection);

var conversationRepository = new ConversationRepository(Connection);

var motorService = new Common.RelayFullCycleMotorService.RelayFullCycleMotorService();

var riskData = MotorQuote.BuildIrishMQRiskInfo(state);

var messageRequestInfo = MotorQuote.BuildMessageRequestInfo();

var quotes = motorService.GetNewBusinessXBreakDownsSpecified(riskData, 100, true, null, messageRequestInfo);

if (quotes.Quotations != null)

{

if (quotes.Quotations.Length > 0)

{

quoteRepository.StoreQuote

(

context.Activity.Conversation.Id,

quotes.TransactionID,

new JavaScriptSerializer().Serialize(quotes.Quotations[0])

);

reply.AttachmentLayout = AttachmentLayoutTypes.Carousel;

reply.Attachments = GetQuoteReceipts(quotes.Quotations);

if (SendEmails)

{

var emailToUserBody = EmailHandler.BuildMotorEmailBodyForUser(quotes.Quotations,

state.FirstName, state.LastName, state.DateOfBirth,

state.PrimaryContactNumber, state.EmailAddress, state.VehicleRegistration,

state.Vehicle.Description, state.VehicleValue,

state.AreaVehicleIsKept, state.LicenceType.GetDescription(),

state.NoClaimsDiscount.GetDescription());

EmailHandler.SendEmailToUser(state.EmailAddress, $"{state.FirstName} {state.LastName}",

emailToUserBody);

var emailToBrokerBody = EmailHandler.BuildMotorEmailBodyForBroker(quotes.Quotations,

state.FirstName, state.LastName, state.DateOfBirth,

state.PrimaryContactNumber, state.EmailAddress, state.VehicleRegistration,

state.Vehicle.Description, state.VehicleValue,

state.AreaVehicleIsKept, state.LicenceType.GetDescription(),

state.NoClaimsDiscount.GetDescription());

EmailHandler.SendEmailToBroker(state.EmailAddress, $"{state.FirstName} {state.LastName}",

emailToBrokerBody);

}

await context.PostAsync(reply);

}

}

else

{

await context.PostAsync("Sorry, we were unable to get your a quote at this point.");

}

conversationRepository.StoreConversation

(

context.Activity.Conversation.Id,

context.Activity.From.Id,

DateTime.Now.ToString(new CultureInfo("en-GB")),

new JavaScriptSerializer().Serialize(context)

);

}

catch (Exception exception)

{

ErrorRepository.LogError(context.Activity.Conversation.Id, context.Activity.From.Id, DateTime.Now.ToString(new CultureInfo("en-GB")), context.ConversationData.ToString(), exception.ToString());

throw;

}

finally

{

context.Done(state);

}

}

private IList<Attachment> GetQuoteReceipts(IrishMQResultsBreakdown[] breakdowns)

{

return breakdowns.Select(breakdown => BuildReceiptCard(breakdown)).ToList();

}

private static Attachment BuildReceiptCard(IrishMQResultsBreakdown breakdown)

{

try

{

var receiptCard = new ReceiptCard

{

Title = $"Insurer: {breakdown.Premium.SchemeName}",

Facts = new List<Fact> { new Fact("Scheme", breakdown.Premium.Scheme.SchemeNumber) },

Tax = $"€{breakdown.Premium.PremiumAfterLevy - breakdown.Premium.PremiumBeforeLevy}",

Total = $"€{breakdown.Premium.TotalPremium}",

Buttons = new List<CardAction>

{

new CardAction

(

ActionTypes.PostBack,

"Request a Callback"

)

}

};

return receiptCard.ToAttachment();

}

catch (Exception exception)

{

ErrorRepository.LogError(DateTime.Now.ToString(new CultureInfo("en-GB")), exception.ToString());

throw;

}

}

}

}

### RootDialog.cs

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.Bot.Builder.Dialogs;

using QuotingBot.Common.Enums;

namespace QuotingBot.Dialogs

{

[Serializable]

public sealed class RootDialog : IDialog<object>

{

private static readonly string MotorInsuranceOption = $"Motor insurance {Emoji.Car.GetDescription()}";

private static readonly string HomeInsuranceOption = $"Home insurance {Emoji.House.GetDescription()}";

public async Task StartAsync(IDialogContext context)

{

await context.PostAsync("Hi, I'm Ava - your friendly quoting bot!");

context.Wait(MessageReceivedAsync);

}

public static void ShowQuoteOptions(IDialogContext context)

{

PromptDialog.Choice

(

context,

OnOptionSelected,

new List<string> { MotorInsuranceOption, HomeInsuranceOption },

"What can I get you a quote for today?",

"Hmmm...that's not a valid option. Please choose an option from the list."

);

}

private static async Task OnOptionSelected(IDialogContext context, IAwaitable<string> result)

{

try

{

var optionSelected = await result;

if (optionSelected == MotorInsuranceOption)

{

context.Call(new MotorDialog(), ResumeAfterOptionDialog);

}

else

{

context.Call(new HomeDialog(), ResumeAfterOptionDialog);

}

}

catch (Exception)

{

await context.PostAsync($"Oops! Something went wrong.");

}

}

private static async Task ResumeAfterOptionDialog(IDialogContext context, IAwaitable<object> result)

{

try

{

var message = await result;

}

catch (Exception ex)

{

await context.PostAsync($"Failed with message: {ex.Message}");

}

finally

{

context.Wait(MessageReceivedAsync);

}

}

public static async Task MessageReceivedAsync(IDialogContext context, IAwaitable<object> result)

{

ShowQuoteOptions(context);

}

}

}

### HomeQuote.cs

using System;

using QuotingBot.Common.Enums;

using QuotingBot.Common.RelayHouseholdService;

using PropertyType = QuotingBot.Common.Enums.PropertyType;

using ResidenceType = QuotingBot.Common.Enums.ResidenceType;

namespace QuotingBot.Models.Home

{

[Serializable]

public class HomeQuote

{

public static EnumConverters enumConverters = new EnumConverters();

public string FirstLineOfAddress;

public string Town;

public string County;

public PropertyType? PropertyType;

public ResidenceType? ResidenceType;

public int? NumberOfBedrooms;

public string YearBuilt;

public string FirstName;

public string LastName;

public string PrimaryContactNumber;

public string EmailAddress;

public static HomeWebServiceRequest BuildHomeWebServiceRequest(HomeQuote state)

{

var request = new HomeWebServiceRequest

{

PolicyHolders = new PolicyHolder[1],

Risks = new Risk[2]

};

var policyHolder = new PolicyHolder

{

EffectivePrimaryPolicyHolder = true,

OccupationType = OccupationType.QuantitySurveyor,

EmployersBusinessType = EmployersBusinessType.Unknown,

ProfessionalBodyType = ProfessionalBodyType.Unknown,

MaritalStatus = MaritalStatus.Single,

EmploymentType = EmploymentType.Employed,

FirstTimeBuyer = false,

Smoker = false,

Cancelled = false,

Declined = false,

Conviction = false,

DeclaredBankrupt = false,

SpecialConditions = false,

Relationship = RelationshipType.Unknown,

Contact = new Contact

{

Title = PersonTitle.Mr,

FirstName = state.FirstName,

Surname = state.LastName,

Address = new Address

{

BuildingName = "Dranagh",

StreetName = state.FirstLineOfAddress,

Town = state.Town,

County = state.County

},

DateOfBirth = new DateTime(1987, 03, 12, 00, 00, 00),

PhoneNumber = state.PrimaryContactNumber,

EmailAddress = state.EmailAddress

}

};

request.RelayNumber = "RE0930";

request.Password = "1IJ4^E?K]Syb>w";

request.BrokerId = "5016";

request.LoginId = "eQuote";

request.BrokerName = "First Ireland Risk Management";

request.ClientVersion = 0;

request.BusinessProcess = BusinessProcess.NewBusiness;

request.ProcessingType = InsurerConfirmationProcessingType.Standard;

request.Policy = new Policy

{

EffectiveStartDate = DateTime.Now,

VoluntaryExcess = 0,

BrokerPolicyReference = "NOV17-Y8AONF",

CorrespondenceContact = new Contact()

{

Title = PersonTitle.Unknown,

DateOfBirth = new DateTime(0001, 01, 01, 00, 00, 00)

}

};

request.PolicyHolders[0] = policyHolder;

request.Occupancy = new Occupancy

{

ResidenceType = enumConverters.ConvertResidencyType(state.ResidenceType),

ProposerType = ProposerType.Unspecified,

YearsLivingAtAddress = 0,

NumberOfPayingGuests = 0,

SocialWelfareLet = false,

IsFurnished = false,

NormalDaytimeOccupancy = false,

NumberOfDaysUnoccupiedPerWeek = 0,

NumberOfTimesLetInAYear = 0

};

request.Building = new Building

{

PropertyType = enumConverters.ConvertPropertyType(state.PropertyType),

PropertySubType = PropertySubType.DetachedHouse,

ConstructionDate = new DateTime(Convert.ToInt32(state.YearBuilt), 01, 01, 00, 00, 00),

ListedBuilding = false,

RoofConstruction = RoofConstructionType.Standard,

WallConstruction = WallConstructionType.Unknown,

RoofPercentage = 0,

NumberOfBedrooms = (int)state.NumberOfBedrooms,

NumberOfBathrooms = 3,

NumberOfSmokeDetectors = 2,

Alarm = new Alarm { AlarmType = AlarmType.Unspecified },

Locks = true,

NeighbourhoodWatchInArea = true,

Basement = false,

HeatingType = HeatingType.Electric,

BuildingSize = 0,

BuildingSizeUnitOfMeasurement = UnitOfMeasurement.Unknown,

GarageSize = 0,

GarageSizeUnitOfMeasurement = UnitOfMeasurement.Unknown,

FreeFromFlooding = true,

FreeFromGroundHeave = true,

FreeFromLandslip = true,

FreeFromSubsidence = true,

GoodRepair = true,

SafeInstalled = false,

UndertakeToMaintain = false

};

request.RiskAddress = new Address

{

StreetName = state.FirstLineOfAddress,

Town = state.Town,

County = state.County,

FreeText1 = $"{state.FirstLineOfAddress}, {state.Town}, {state.County}",

AddressMatchResults = new AddressMatchResult[]

{

new AddressMatchResult

{

ProvidedBy = AddressLookupProvider.Gamma,

GeoCode = "40291613",

MatchType = "region",

Reference = "4IZJT7KP6X734AQK",

MatchLevel = "700",

IsFallbackResult = false,

LookupResponse =

"<![CDATA[&lt;location type='region' territory='SPIKE\_GAMMA' score='99.99' xmlns='http://service.autoaddress.ie/'&gt;&lt;point x='275383.86' y='138100.57' coord-sys='ING' /&gt;&lt;info ecadId='1110030370' eircode=' Autoaddressid='4IZJT7KP6X734AQK' geover='Q117' geotype='L' georef='40291613' name='' text='Dranagh,Saint Mullins,Co. Carlow' addr1='Dranagh' addr2='Saint Mullins' addr3='Co. Carlow' matchLevel='700' matchResult'100' aa2MatchLevel='7' aa2MatchResult='300' smallarea='017020001' ecadSmallarea='48' /&gt;&lt;/location&gt;]]>"

}

}

};

request.Risks[0] = new Risk

{

Group = RateBreakdownGroup.HouseStructure,

SumInsured = 300000

};

request.Risks[1] = new Risk

{

Group = RateBreakdownGroup.HouseContents,

SumInsured = 25000

};

request.HomeRequestSource = HomeRequestSource.eQuote;

request.QuoteReference = "NOV17-Y8AONF";

request.FullQuoteRequest = false;

return request;

}

}

}

### MotorQuote.cs

using System;

using System.Configuration;

using System.Globalization;

using QuotingBot.Common.Enums;

using QuotingBot.DAL.Repository.Errors;

using QuotingBot.Common.RelayFullCycleMotorService;

namespace QuotingBot.Models.Motor

{

[Serializable]

public class MotorQuote

{

public static Common.RelayFullCycleMotorService.RelayFullCycleMotorService motorService = new Common.RelayFullCycleMotorService.RelayFullCycleMotorService();

private static readonly string Connection = ConfigurationManager.ConnectionStrings["QuotingBot"].ConnectionString;

private static readonly ErrorRepository \_errorRepository = new ErrorRepository(Connection);

public static EnumConverters enumConverters = new EnumConverters();

public string VehicleRegistration;

public string VehicleValue;

public string AreaVehicleIsKept;

public string FirstName;

public string LastName;

public string DateOfBirth;

public LicenceType? LicenceType;

public NoClaimsDiscount? NoClaimsDiscount;

public string PrimaryContactNumber;

public string EmailAddress;

public Vehicle Vehicle = new Vehicle();

public static Vehicle GetVehicle(string vehicleRegistration)

{

var vehicle = new Vehicle();

try

{

vehicle.AbiCode = motorService.GetVehicleLookup(

vehicleRegistration,

string.Empty,

string.Empty,

string.Empty,

string.Empty,

string.Empty,

string.Empty,

string.Empty,

string.Empty,

string.Empty,

"RE0098",

"relay1:0099",

VehicleLookup.Motor).ABICode;

if (!string.IsNullOrEmpty(vehicle.AbiCode))

{

vehicle = GetVehicleDetails(vehicle.AbiCode);

}

}

catch (Exception exception)

{

\_errorRepository.LogError(DateTime.Now.ToString(new CultureInfo("en-GB")), exception.ToString());

throw;

}

return vehicle;

}

private static Vehicle GetVehicleDetails(string ABICode)

{

var vehicle = new Vehicle();

try

{

var vehicleLookupItem = motorService.GetVehicleDetailsABI(ABICode);

vehicle.AbiCode = ABICode;

vehicle.Description = vehicleLookupItem.Description;

vehicle.Manufacturer = vehicleLookupItem.Manufacturer;

vehicle.Model = vehicleLookupItem.Model;

vehicle.BodyType = vehicleLookupItem.BodyType;

vehicle.EngineCapacity = vehicleLookupItem.EngineCapacity;

vehicle.NumberOfDoors = vehicleLookupItem.NumberDoors;

vehicle.FuelType = vehicleLookupItem.FuelType;

vehicle.YearOfFirstManufacture = vehicleLookupItem.YearOfFirstManufacture;

}

catch(Exception exception)

{

\_errorRepository.LogError(DateTime.Now.ToString(new CultureInfo("en-GB")), exception.ToString());

throw;

}

return vehicle;

}

public static IrishMQRiskInfo BuildIrishMQRiskInfo(MotorQuote state)

{

var riskInfo = new IrishMQRiskInfo();

riskInfo.Driver = new IrishDriverInfo[1];

riskInfo.Vehicle = new IrishVehicleInfo[1];

riskInfo.Cover = new IrishCoverInfo[1];

var driver = new IrishDriverInfo

{

PRN = 1,

RelationshipToProposer = "Z",

DriverLicenceNumber = "550956042",

Title = "005",

Forename = state.FirstName,

Surname = state.LastName,

Sex = "M",

MaritalStatus = "M",

LicenceType = enumConverters.ConvertLicenceType(state.LicenceType),

LicenceCountry = "IE",

ProsecutionPending = false,

LicenceRestrictionInd = false,

QualificationsInd = false,

NonMotoringConviction = false,

PrevRefusedCover = false,

OtherVehicleOwned = false,

PrevRestrictiveTerms = false,

RegisteredDisabled = false,

ClaimsIndicator = false,

PenaltyPointsIndicator = false,

ConvictionsInd = false,

MedicalConditionsInd = false,

ResidentOutsideIreland = false,

PermResident = true,

NonDrinker = false,

TempAdditionalDriver = false,

DateOfBirth = Convert.ToDateTime(state.DateOfBirth, new CultureInfo("en-GB")),

IrelandResidencyDate = new DateTime(2000, 04, 11, 02, 00, 00),

IrelandLicenceDate = new DateTime(2014, 08, 28, 02, 00, 00),

NameddriverNCDClaimedYears = 6,

ResidentWithProposer = false,

FullTimeUseOfOtherCar = false,

IsResidentWithProposer = false,

PrevImposedTerms = false,

Occupation = new IrishOccupationInfo[]

{

new IrishOccupationInfo

{

FullTimeEmployment = true,

OccupationCode = "SBB",

EmployersBusiness = "120",

EmploymentType = "E"

}

},

DrivesVehicle = new IrishDrivesVehicleInfo[]

{

new IrishDrivesVehicleInfo

{

VehicleReferenceNumber = 1,

DrivingFrequency = "M",

Use = "4"

}

}

};

riskInfo.Proposer = new IrishProposerInfo

{

ProposerType = enmProposerType.eIndividual,

TitleCode = "005",

Title = "Mr.",

ForeName = state.FirstName,

SurName = state.LastName,

Sex = "M",

MaritalStatus = "M",

DateOfBirth = Convert.ToDateTime(state.DateOfBirth, new CultureInfo("en-GB")),

Address = new IrishAddressInfo

{

Line1 = "1 Main Street",

Line2 = "Donegal",

Line3 = "County Donegal",

GeoCode = "38443614",

MatchType = "subbuilding",

MatchLevel = "100",

RatingFactor = "1.391",

MRACode = "MRA268067012",

SmallAreaIdentifier = 5354,

EcadIdentifier = 1700378046,

EcadMatchLevelCode = "2",

EcadMatchResultCode = "100",

Eircode = "D11F6E5",

ProvidedBy = AddressProvider.Gamma

},

Contact = new IrishContactInfo

{

Home = state.PrimaryContactNumber,

Email = state.EmailAddress

},

NCD = new IrishNCDInfo

{

ClaimedYearsEarned = enumConverters.ConvertNoClaimsDiscount(state.NoClaimsDiscount),

DrivingExperienceYears = 0,

ClaimedCountry = "IE",

ClaimedInsurer = "029",

PreviousPolicyNumber = "123456789",

DrivingExperiencePolicyExpiryDate = DateTime.Now.AddDays(1),

ClaimedDiscountType = "S",

ClaimedBonusProtectionType = "F",

ClaimedProtectedInd = false,

ProtectionRequiredInd = true,

DrivingExperienceProvenInd = true,

ClaimedProvenInd = false,

PreviousPolicyExpiryDate = DateTime.Now.AddDays(-5),

RebrokeYearsProvided = false,

RebrokeYears = 0

},

YearsAtHomeAddress = 0,

HomeownerInd = "N"

};

riskInfo.Policy = new IrishPolicyInfo

{

PolicyNumber = "QWERTY12345",

StartTime = "000100",

EndTime = "120000",

PreviousInsurer = "029",

QuoteAuthor = "RLY",

CurrencyRequired = "EUR",

InceptionDate = DateTime.Now,

StartDate = DateTime.Now,

EndDate = DateTime.Now.AddYears(1),

CurrentYear = DateTime.Now.Year,

PreviouslyInsuredInd = true,

SecondCarQuotationInd = false

};

riskInfo.Driver[0] = driver;

riskInfo.Vehicle[0] = new IrishVehicleInfo

{

PRN = 1,

Value = Convert.ToInt32(state.VehicleValue),

AnnualMilage = 10000,

BusinessMileage = 0,

PleasureMileage = 10000,

NonStandardAudioValue = 0,

CarPhoneValue = 0,

NoDriversFullLicence = 1,

NoOfSeats = 5,

ManufacturedYear = 2005,

FirstRegdYear = 2005,

ModelCode = state.Vehicle.AbiCode,

ModelName = state.Vehicle.Description,

KeptAt = "HA",

AreaKeptAt = "DX11",

CubicCapacity = state.Vehicle.EngineCapacity.ToString(),

BodyType = "5",

OvernightLocation = "2",

AreaRating = "DX11",

Owner = "1",

RegistrationNo = state.VehicleRegistration,

RegisteredKeeper = "1",

DateManufactured = new DateTime(state.Vehicle.YearOfFirstManufacture, 01, 01, 02, 00, 00),

DateFirstRegistered = new DateTime(state.Vehicle.YearOfFirstManufacture, 01, 01, 02, 00, 00),

DatePurchased = new DateTime(2017, 05, 01, 02, 00, 00),

ModifiedInd = false,

IrelandRegistered = true,

Imported = false,

SecurityDeviceInd = true,

TrailerInd = false,

SecondCarInd = false,

TemporaryAddVehicle = false,

TemporarySubInd = false,

LeftOrRightHandDrive = (char) 82,

ReferenceNumber = 1,

Security = new IrishSecurityInfo

{

Type = "1002"

},

Uses = new IrishUsesInfo

{

Code = "4"

},

DrivenBy = new IrishDrivenByInfo[]

{

new IrishDrivenByInfo

{

DriverReferenceNumber = 1,

DrivingFrequency = "M"

},

new IrishDrivenByInfo()

{

DriverReferenceNumber = 2,

DrivingFrequency = "F"

}

},

VehicleType = 0

};

riskInfo.Cover[0] = new IrishCoverInfo

{

Code = "01",

PeriodUnits = "2",

Period = "12",

CertificateNumber = "0",

StartTime = "000100",

StartDate = DateTime.Now.AddDays(1),

ExpiryDate = DateTime.Now.AddYears(1),

RequiredDrivers = "5",

VehicleRefNo = 1,

TotalTempMTA = 0,

TotalTempMTAInForce = 0,

TotalTempAddDriverInForce = 0,

TotalTempAddDriver = 0,

TotalTempAddVehicle = 0,

TotalTempSub = 0,

VoluntaryExcess = 300,

WindscreenLimit = 0

};

riskInfo.Intermediary = new IntermediaryInfo

{

Name = "RE0668",

Number = 0,

RIAccountIdentifier = "relay1:0099"

};

riskInfo.TransactionDetail = new TransactionDetails

{

BrokerFee = 0

};

riskInfo.DiscountInfo = new IrishDiscountInfo

{

IsWebQuote = false,

WebDiscountPercentage = 0

};

return riskInfo;

}

public static MessageRequestInfo BuildMessageRequestInfo()

{

var messageRequestInfo = new MessageRequestInfo

{

BreakdownsSpecified1 = new BreakdownsSpecified

{

BreakdownSpecified1 = new BreakdownType[]

{

BreakdownType.ExcessItems

}

}

};

return messageRequestInfo;

}

}

}

### Vehicle.cs

using System;

namespace QuotingBot.Models.Motor

{

[Serializable]

public class Vehicle

{

public string Manufacturer { get; set; }

public string Model { get; set; }

public string BodyType { get; set; }

public int NumberOfDoors { get; set; }

public int YearOfFirstManufacture { get; set; }

public int EngineCapacity { get; set; }

public string FuelType { get; set; }

public string Description { get; set; }

public string AbiCode { get; set; }

}

}

### EmailHandler.cs

using System;

using System.ComponentModel;

using System.Configuration;

using System.Net;

using System.Net.Mail;

using QuotingBot.Common.RelayFullCycleMotorService;

using QuotingBot.Common.RelayHouseholdService;

using QuotingBot.Common.Enums;

namespace QuotingBot.Common.Email

{

public class EmailHandler

{

public EmailHandler() { }

private static void SendCompletedCallback(object sender, AsyncCompletedEventArgs e)

{

// Get the unique identifier for this asynchronous operation.

String token = (string)e.UserState;

if (e.Error != null)

{

Console.WriteLine("[{0}] {1}", token, e.Error.ToString());

}

else

{

Console.WriteLine("Message sent.");

}

}

private static SmtpClient SetupSmtpClient()

{

var mailServerAddress = ConfigurationManager.AppSettings["MailServerAddress"];

var mailServerUser = ConfigurationManager.AppSettings["MailServerUser"];

var mailServerPassword = ConfigurationManager.AppSettings["MailServerPassword"];

var client =

new SmtpClient(mailServerAddress)

{

Credentials = new NetworkCredential(mailServerUser, mailServerPassword)

};

return client;

}

public static void SendEmailToUser(string toEmail, string toName, string body)

{

var emailSenderAddress = ConfigurationManager.AppSettings["SenderEmailAddress"];

var emailSenderName = ConfigurationManager.AppSettings["EmailSenderName"];

var emailFrom = new MailAddress(emailSenderAddress, emailSenderName);

var emailTo = new MailAddress(toEmail, toName);

var message = new MailMessage(emailFrom, emailTo)

{

Body = body,

Subject = "Insurace Quote",

IsBodyHtml = true

};

var client = SetupSmtpClient();

// Set the method that is called back when the send operation ends.

client.SendCompleted += new

SendCompletedEventHandler(SendCompletedCallback);

// The userState can be any object that allows your callback

// method to identify this send operation.

// For this example, the userToken is a string constant.

const string userState = "sending message";

client.SendAsync(message, userState);

}

public static string BuildHomeEmailBodyForUser(HomeQuoteWebServiceResult[] responseQuotes,

string firstName, string lastName, string contactNumber, string emailAddress,

string firstLineOfAddress, string town, string county, string propertyType,

string residenceType, string yearBuilt, string numberOfBedrooms)

{

string body;

body = $"Hi {firstName},<br><br>";

body += "Thanks for getting your home insurance quote with us.<br><br>";

body += "We've listed the quotes you received.<br><br>";

body += "<table border=\"1\"><tbody>";

body += "<tr><th>Insurer</th><th>Scheme</th><th>Total</th></tr>";

foreach (var quote in responseQuotes)

{

if (quote.NetPremium > 0)

{

body += $"<tr><td>{quote.InsurerName}</td><td>{quote.SchemeName}</td><td>€{quote.NetPremium}</td></tr>";

}

}

body += "</tbody></table><br><br>";

body += "Entered risk details:<br>";

body += $"<strong>Name:</strong> {firstName} {lastName}<br>";

body += $"<strong>Contact Number:</strong> {contactNumber}<br>";

body += $"<strong>Email:</strong> {emailAddress}<br>";

body += $"<strong>Address Line 1:</strong> {firstLineOfAddress}<br>";

body += $"<strong>Town:</strong> {town}<br>";

body += $"<strong>County:</strong> {county}<br>";

body += $"<strong>Property:</strong> {propertyType}<br>";

body += $"<strong>Residence:</strong> {residenceType}<br>";

body += $"<strong>Year Built:</strong> {yearBuilt}<br>";

body += $"<strong>No. of Bedrooms:</strong> {numberOfBedrooms}<br><br>";

body += "Thanks,<br>";

body += $"Ava - your friendly Quoting Bot {Emoji.GrinningFace.GetDescription()}";

return body;

}

public static string BuildMotorEmailBodyForUser(IrishMQResultsBreakdown[] quotes,

string firstName, string lastName, string dateOfBirth, string contactNumber, string emailAddress,

string vehicleRegistration, string vehicleDescription, string vehicleValue, string areaVehicleKept,

string licenceType, string noClaimsDiscountYears)

{

string body;

body = $"Hi {firstName},<br><br>";

body += "Thanks for getting your home insurance quote with us.<br><br>";

body += "We've listed the quotes you received.<br><br>";

body += "<table border=\"1 solid\"><tbody>";

body += "<tr><th>Insurer</th><th>Total</th></tr>";

foreach (var quote in quotes)

{

if (quote.Premium.TotalPremium > 0)

{

body += $"<tr><td>{quote.Premium.SchemeName}</td><td>€{quote.Premium.TotalPremium}</td></tr>";

}

}

body += "</tbody></table><br><br>";

body += "Entered risk details:<br>";

body += $"<strong>Name:</strong> {firstName} {lastName}<br>";

body += $"<strong>Date of Birth:</strong> {dateOfBirth}<br>";

body += $"<strong>Contact Number:</strong> {contactNumber}<br>";

body += $"<strong>Email:</strong> {emailAddress}<br>";

body += $"<strong>Vehicle Registration:</strong> {vehicleRegistration}<br>";

body += $"<strong>Vehicle Description:</strong> {vehicleDescription}<br>";

body += $"<strong>Vehilce Value:</strong> €{vehicleValue}<br>";

body += $"<strong>Area Vehilce Kept:</strong> {areaVehicleKept}<br>";

body += $"<strong>Licence:</strong> {licenceType}<br>";

body += $"<strong>No Claims Discount:</strong> {noClaimsDiscountYears}<br><br>";

body += "Thanks,<br>";

body += $"Ava - your friendly Quoting Bot {Emoji.GrinningFace.GetDescription()}";

return body;

}

public static void SendEmailToBroker(string toEmail, string toName, string body)

{

var emailSenderAddress = ConfigurationManager.AppSettings["SenderEmailAddress"];

var emailSenderName = ConfigurationManager.AppSettings["EmailSenderName"];

var emailFrom = new MailAddress(emailSenderAddress, emailSenderName);

var emailTo = new MailAddress(toEmail, toName);

var message = new MailMessage(emailFrom, emailTo)

{

Body = body,

Subject = "Customer Insurace Quote",

IsBodyHtml = true

};

var client = SetupSmtpClient();

// Set the method that is called back when the send operation ends.

client.SendCompleted += new

SendCompletedEventHandler(SendCompletedCallback);

// The userState can be any object that allows your callback

// method to identify this send operation.

// For this example, the userToken is a string constant.

const string userState = "sending message";

client.SendAsync(message, userState);

}

public static string BuildMotorEmailBodyForBroker(IrishMQResultsBreakdown[] quotes,

string firstName, string lastName, string dateOfBirth, string contactNumber, string emailAddress,

string vehicleRegistration, string vehicleDescription, string vehicleValue, string areaVehicleKept,

string licenceType, string noClaimsDiscountYears)

{

string body;

body = $"Hi,<br><br>";

body += "An insurance quote was given through the chatbot.<br><br>";

body += "We've listed the quotes given.<br><br>";

body += "<table border=\"1 solid\"><tbody>";

body += "<tr><th>Insurer</th><th>Total</th></tr>";

foreach (var quote in quotes)

{

if (quote.Premium.TotalPremium > 0)

{

body += $"<tr><td>{quote.Premium.SchemeName}</td><td>€{quote.Premium.TotalPremium}</td></tr>";

}

}

body += "</tbody></table><br><br>";

body += "Entered risk details:<br>";

body += $"<strong>Name:</strong> {firstName} {lastName}<br>";

body += $"<strong>Date of Birth:</strong> {dateOfBirth}<br>";

body += $"<strong>Contact Number:</strong> {contactNumber}<br>";

body += $"<strong>Email:</strong> {emailAddress}<br>";

body += $"<strong>Vehicle Registration:</strong> {vehicleRegistration}<br>";

body += $"<strong>Vehicle Description:</strong> {vehicleDescription}<br>";

body += $"<strong>Vehilce Value:</strong> €{vehicleValue}<br>";

body += $"<strong>Area Vehilce Kept:</strong> {areaVehicleKept}<br>";

body += $"<strong>Licence:</strong> {licenceType}<br>";

body += $"<strong>No Claims Discount:</strong> {noClaimsDiscountYears}<br><br>";

body += "Thanks,<br>";

body += $"Ava - your friendly Quoting Bot {Emoji.GrinningFace.GetDescription()}";

return body;

}

public static string BuildHomeEmailBodyForBroker(HomeQuoteWebServiceResult[] responseQuotes,

string firstName, string lastName, string contactNumber, string emailAddress,

string firstLineOfAddress, string town, string county, string propertyType,

string residenceType, string yearBuilt, string numberOfBedrooms)

{

string body;

body = $"Hi,<br><br>";

body += "An insurance quote was given through the chatbot.<br><br>";

body += "We've listed the quotes given.<br><br>";

body += "<table border=\"1\"><tbody>";

body += "<tr><th>Insurer</th><th>Scheme</th><th>Total</th></tr>";

foreach (var quote in responseQuotes)

{

if (quote.NetPremium > 0)

{

body += $"<tr><td>{quote.InsurerName}</td><td>{quote.SchemeName}</td><td>€{quote.NetPremium}</td></tr>";

}

}

body += "</tbody></table><br><br>";

body += "Entered risk details:<br>";

body += $"<strong>Name:</strong> {firstName} {lastName}<br>";

body += $"<strong>Contact Number:</strong> {contactNumber}<br>";

body += $"<strong>Email:</strong> {emailAddress}<br>";

body += $"<strong>Address Line 1:</strong> {firstLineOfAddress}<br>";

body += $"<strong>Town:</strong> {town}<br>";

body += $"<strong>County:</strong> {county}<br>";

body += $"<strong>Property:</strong> {propertyType}<br>";

body += $"<strong>Residence:</strong> {residenceType}<br>";

body += $"<strong>Year Built:</strong> {yearBuilt}<br>";

body += $"<strong>No. of Bedrooms:</strong> {numberOfBedrooms}<br><br>";

body += "Thanks,<br>";

body += $"Ava - your friendly Quoting Bot {Emoji.GrinningFace.GetDescription()}";

return body;

}

}

}

### Emoji.cs

using System;

using System.ComponentModel;

namespace QuotingBot.Common.Enums

{

[Serializable]

public enum Emoji

{

[Description("\U0001F604")]

GrinningFace,

[Description("\U0001F914")]

ThinkingFace,

[Description("\U0001F44D")]

ThumbsUp,

[Description("\U0001F698")]

Car,

[Description("\U0001F3E1")]

House,

[Description("\U0001F4E7")]

Email

}

}

### EnumConverters.cs

using System;

namespace QuotingBot.Common.Enums

{

[Serializable]

public class EnumConverters

{

public EnumConverters() { }

public string ConvertLicenceType(object value)

{

switch (value)

{

case LicenceType.FullIrish:

return "C";

case LicenceType.ProvisionalIrish:

return "B";

case LicenceType.FullEU:

return "F";

case LicenceType.FullUK:

return "C";

case LicenceType.Foreign:

return "I";

case LicenceType.InternationalLicence:

return "N";

case LicenceType.LearnerPermit:

return "G";

default:

return string.Empty;

}

}

public int ConvertNoClaimsDiscount(object value)

{

switch (value)

{

case NoClaimsDiscount.Zero:

return 0;

case NoClaimsDiscount.One:

return 1;

case NoClaimsDiscount.Two:

return 2;

case NoClaimsDiscount.Three:

return 3;

case NoClaimsDiscount.Four:

return 4;

case NoClaimsDiscount.Five:

return 5;

case NoClaimsDiscount.Six:

return 6;

case NoClaimsDiscount.Seven:

return 7;

case NoClaimsDiscount.Eight:

return 8;

case NoClaimsDiscount.NineOrMore:

return 9;

default:

return 0;

}

}

public RelayHouseholdService.PropertyType ConvertPropertyType(PropertyType? propertyType)

{

switch (propertyType)

{

case PropertyType.Bungalow:

return RelayHouseholdService.PropertyType.Bungalow;

case PropertyType.DetachedHouse:

return RelayHouseholdService.PropertyType.DetachedHouse;

case PropertyType.Flat:

return RelayHouseholdService.PropertyType.Flat;

case PropertyType.SemiDetachedHouse:

return RelayHouseholdService.PropertyType.SemiDetachedHouse;

case PropertyType.TerracedHouse:

return RelayHouseholdService.PropertyType.TerracedHouse;

default:

return RelayHouseholdService.PropertyType.Unknown;

}

}

public RelayHouseholdService.ResidenceType ConvertResidencyType(ResidenceType? residenceType)

{

switch (residenceType)

{

case ResidenceType.OwnerOccupied:

return RelayHouseholdService.ResidenceType.OwnerOccupied;

case ResidenceType.RentedFamily:

return RelayHouseholdService.ResidenceType.RentedFamily;

case ResidenceType.RentedStudents:

return RelayHouseholdService.ResidenceType.RentedStudents;

default:

return RelayHouseholdService.ResidenceType.Unspecified;

}

}

}

}

### EnumExtension.cs

using System;

using System.ComponentModel;

using System.Reflection;

namespace QuotingBot.Common.Enums

{

public static class EnumExtension

{

public static string GetDescription(this Enum value)

{

FieldInfo field = value.GetType().GetField(value.ToString());

object[] attribs = field.GetCustomAttributes(typeof(DescriptionAttribute), true);

return attribs.Length > 0 ? ((DescriptionAttribute)attribs[0]).Description : string.Empty;

}

}

}

### LicenceType.cs

using System;

using System.ComponentModel;

namespace QuotingBot.Common.Enums

{

[Serializable]

public enum LicenceType

{

[Description("Full Irish")]

FullIrish,

[Description("Provisional Irish")]

ProvisionalIrish,

[Description("Full EU")]

FullEU,

[Description("Full UK")]

FullUK,

[Description("Foreign Licence")]

Foreign,

[Description("International Licence")]

InternationalLicence,

[Description("Learner Permit")]

LearnerPermit

}

}

### NoClaimsDiscount.cs

using System;

using System.ComponentModel;

namespace QuotingBot.Common.Enums

{

[Serializable]

public enum NoClaimsDiscount

{

[Description("0")]

Zero,

[Description("1")]

One,

[Description("2")]

Two,

[Description("3")]

Three,

[Description("4")]

Four,

[Description("5")]

Five,

[Description("6")]

Six,

[Description("7")]

Seven,

[Description("8")]

Eight,

[Description("9+")]

NineOrMore

}

}

### PropertyType.cs

using System;

using System.ComponentModel;

namespace QuotingBot.Common.Enums

{

[Serializable]

public enum PropertyType

{

[Description("Bungalow")]

Bungalow,

[Description("Detached House")]

DetachedHouse,

[Description("Flat")]

Flat,

[Description("Semi-Detached House")]

SemiDetachedHouse,

[Description("Terraced House")]

TerracedHouse

}

}

### ResidenceType.cs

using System;

using System.ComponentModel;

namespace QuotingBot.Common.Enums

{

[Serializable]

public enum ResidenceType

{

[Description("Owner Occupied")]

OwnerOccupied,

[Description("Family Rental")]

RentedFamily,

[Description("Student Rental")]

RentedStudents

}

}

### Formatter.cs

using System;

namespace QuotingBot.Common.Helpers

{

[Serializable]

public class Formatter

{

public Formatter() { }

public string CapitilizeFirstLetter(string value)

{

char[] characters = value.ToCharArray();

characters[0] = char.ToUpper(characters[0]);

return new string(characters);

}

}

}

### Validation.cs

using Microsoft.Bot.Builder.FormFlow;

using System;

using System.Configuration;

using System.Globalization;

using System.Text.RegularExpressions;

using QuotingBot.DAL.Repository.Errors;

using QuotingBot.Common.Enums;

using System.Linq;

namespace QuotingBot.Common.Helpers

{

[Serializable]

public class Validation

{

private static readonly string Connection = ConfigurationManager.ConnectionStrings["QuotingBot"].ConnectionString;

private static readonly ErrorRepository ErrorRepository = new ErrorRepository(Connection);

private static readonly RelayFullCycleMotorService.RelayFullCycleMotorService MotorService = new RelayFullCycleMotorService.RelayFullCycleMotorService();

private readonly Formatter \_formatter = new Formatter();

public Validation() { }

public ValidateResult ValidateVehicleValue(object value)

{

var result = new ValidateResult

{

IsValid = false

};

if (decimal.TryParse(value.ToString(), out decimal returnValue))

{

result.IsValid = true;

result.Value = Math.Round(returnValue, MidpointRounding.AwayFromZero).ToString();

}

else

{

result.Feedback = $"The value {value} wasn't valid. Make sure you enter a number, like €2000.";

}

return result;

}

public ValidateResult ValidateFirstName(object value)

{

var firstName = value.ToString();

var result = new ValidateResult

{

IsValid = false

};

if(!string.IsNullOrEmpty(firstName))

{

result.IsValid = true;

result.Value = \_formatter.CapitilizeFirstLetter(firstName);

}

else

{

result.Feedback = "You need to provide a first name to continue.";

}

return result;

}

public ValidateResult ValidateTown(object value)

{

var town = value.ToString();

var result = new ValidateResult

{

IsValid = false

};

if (MotorService.GetAreaCodeList().Contains(town))

{

result.IsValid = true;

result.Value = value.ToString();

}

else

{

result.Feedback = $"Oh dear...I don't recognise that town. Can you check the spelling of '{town}' or try an area close by? Thanks {Emoji.ThumbsUp.GetDescription()}";

}

return result;

}

public ValidateResult ValidateCounty(object value)

{

var county = value.ToString();

var result = new ValidateResult

{

IsValid = false

};

if (MotorService.GetCountyCodeList().Contains(county))

{

result.IsValid = true;

result.Value = value.ToString();

}

else

{

result.Feedback = $"Oh dear...I don't recognise that county. Can you check the spelling of '{county}' or try an area close by? Thanks {Emoji.ThumbsUp.GetDescription()}";

}

return result;

}

public ValidateResult ValidateYearBuilt(object value)

{

var result = new ValidateResult

{

IsValid = false

};

if (IsYearBuiltValid(value.ToString()) && int.TryParse(value.ToString(), out int returnValue))

{

result.IsValid = true;

result.Value = returnValue.ToString();

}

else

{

result.Feedback = $"The value {value} wasn't valid. Make sure you enter a year in 'YYYY' format, like 2018.";

}

return result;

}

private bool IsYearBuiltValid(string yearBuilt)

{

string validYearPattern = @"^[0-9]{4}$";

Regex validYear = new Regex(validYearPattern);

return validYear.IsMatch(yearBuilt);

}

public ValidateResult ValidateLastName(object value)

{

var lastName = value.ToString();

var result = new ValidateResult

{

IsValid = false

};

if (!string.IsNullOrEmpty(lastName))

{

result.IsValid = true;

result.Value = \_formatter.CapitilizeFirstLetter(lastName);

}

else

{

result.Feedback = "You need to provide a last name to continue.";

}

return result;

}

public ValidateResult ValidateAreaVehicleIsKept(object value)

{

var area = value.ToString();

var result = new ValidateResult

{

IsValid = false

};

if (MotorService.GetAreaCodeList().Contains(area) || MotorService.GetCountyCodeList().Contains(area))

{

result.IsValid = true;

result.Value = value.ToString();

}

else

{

result.Feedback = $"Oh dear...I don't recognise that area. Can you check the spelling of '{area}' or try an area close by? Thanks {Emoji.ThumbsUp}";

}

return result;

}

public ValidateResult ValidateDateOfBirth(object value)

{

var result = new ValidateResult

{

IsValid = false

};

try

{

CultureInfo culture = new CultureInfo("en-GB");

var date = Convert.ToDateTime(value, culture);

if(IsProposerOfLegalDrivingAge(date))

{

result.IsValid = true;

result.Value = value.ToString();

}

else

{

result.Feedback = "Sorry, but we can't quote for anyone under the age of 17";

return result;

}

}

catch (Exception ex)

{

ErrorRepository.LogError(DateTime.Now.ToShortDateString(), ex.InnerException.ToString());

throw;

}

return result;

}

public ValidateResult ValidateEmailAddress(object value)

{

var result = new ValidateResult

{

IsValid = false

};

if(IsEmailAddressValid(value.ToString()))

{

result.IsValid = true;

result.Value = value.ToString();

}

else

{

result.Feedback = $"Please enter a valid email address {Emoji.Email}";

}

return result;

}

private bool IsProposerOfLegalDrivingAge(DateTime dateOfBirth) => dateOfBirth <= DateTime.Now.AddYears(-17);

private bool IsEmailAddressValid(string emailAddress)

{

string validEmailPattern = @"\b[A-Z0-9.\_%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b";

Regex validEmailAddress = new Regex(validEmailPattern, RegexOptions.IgnoreCase);

return validEmailAddress.IsMatch(emailAddress);

}

public ValidateResult ValidateNumberOfBedrooms(object value)

{

var result = new ValidateResult

{

IsValid = false

};

if (int.TryParse(value.ToString(), out int returnValue))

{

if (returnValue >= 0 && returnValue <= 9)

{

result.IsValid = true;

result.Value = returnValue.ToString();

}

else

{

result.Feedback = $"Sorry, but we can't quote for {value} bedrooms.";

}

}

else

{

result.Feedback = $"Sorry, {value} wasn't valid number of bedrooms.";

}

return result;

}

}

}

### Conversation.cs

namespace QuotingBot.DAL.Models

{

public class Conversation

{

public string ConversationId { get; set; }

public string UserId { get; set; }

public string ConversationDate { get; set; }

public string ConversationLog { get; set; }

public Conversation(string conversationId, string userId, string conversationDate, string conversationLog)

{

ConversationId = conversationId;

UserId = userId;

ConversationDate = conversationDate;

ConversationLog = conversationLog;

}

}

}

### Error.cs

using System;

namespace QuotingBot.DAL.Models

{

public class Error

{

public string ConversationId { get; set; }

public string UserId { get; set; }

public string ConversationDate { get; set; }

public string ConversationLog { get; set; }

public string ErrorMessage { get; set; }

public Error(string conversationId, string userId, string conversationDate, string conversationLog, string errorMessage)

{

ConversationId = conversationId;

UserId = userId;

ConversationDate = conversationDate;

ConversationLog = conversationLog;

ErrorMessage = errorMessage;

}

}

}

### QuoteRepository.cs

using Dapper;

using QuotingBot.Models;

using System.Data;

using System.Data.SqlClient;

namespace QuotingBot.DAL.Quotes

{

public class QuoteRepository

{

private string ConnectionString { get; }

public QuoteRepository(string connectionString) {

ConnectionString = connectionString;

}

public async void StoreQuote(string conversationId, string quoteId, string quoteInfo)

{

using (var connection = new SqlConnection(ConnectionString))

{

await connection.OpenAsync();

var quote = new Quote

(

conversationId,

quoteId,

quoteInfo

);

await connection.ExecuteAsync("usp\_Add\_Quote", quote, null, null, CommandType.StoredProcedure);

}

}

}

}

### ConversationRepository.cs

using Dapper;

using QuotingBot.DAL.Models;

using QuotingBot.DAL.Repository.Errors;

using System;

using System.Data;

using System.Data.SqlClient;

namespace QuotingBot.DAL.Repository.Conversations

{

public class ConversationRepository

{

private string Connection { get; }

public ConversationRepository(string connection)

{

Connection = connection;

}

public async void StoreConversation(string conversationId, string userId, string conversationDate, string conversationLog)

{

try

{

using (var connection = new SqlConnection(Connection))

{

await connection.OpenAsync();

var conversation = new Conversation

(

conversationId,

userId,

conversationDate,

conversationLog

);

await connection.ExecuteAsync(

"usp\_Add\_Conversation",

conversation,

null, null,

CommandType.StoredProcedure);

}

}

catch(Exception exception)

{

var errorRepository = new ErrorRepository(Connection);

errorRepository.LogError(conversationId, userId, DateTime.Now.ToString(), conversationLog, exception.ToString());

throw;

}

}

}

}

### ErrorRepository.cs

using Dapper;

using QuotingBot.DAL.Models;

using System;

using System.Data;

using System.Data.SqlClient;

namespace QuotingBot.DAL.Repository.Errors

{

public class ErrorRepository

{

private string Connection { get; }

public ErrorRepository(string connection)

{

Connection = connection;

}

public async void LogError(string conversationId, string userId, string conversationDate, string conversationLog, string errorMessage)

{

using (var connection = new SqlConnection(Connection))

{

await connection.OpenAsync();

var error = new Error

(

conversationId,

userId,

conversationDate,

conversationLog,

errorMessage

);

await connection.ExecuteAsync("usp\_Add\_Error", error, null, null, CommandType.StoredProcedure);

}

}

public async void LogError(string conversationDate, string errorMessage)

{

using (var connection = new SqlConnection(Connection))

{

await connection.OpenAsync();

var quote = new Error

(

Guid.Empty.ToString(),

Guid.Empty.ToString(),

conversationDate,

string.Empty,

errorMessage

);

await connection.ExecuteAsync("usp\_Add\_Error", quote, null, null, CommandType.StoredProcedure);

}

}

}

}

### Program.cs

using System;

using System.Linq;

using System.Reflection;

using DbUp;

namespace QuotingBot.DbUp

{

public static class Program

{

// -s "QuotingBotAlias" -d "QuotingBot" -u "QuotingBotDeployment" -p "QuotingBotDeployment" "-create" "-createlogins" "-test"

static int Main(string[] args)

{

var connectionString = args.FirstOrDefault()

?? "Server=PCONNOLLY\\SQL2014; Database=QuotingBot; Trusted\_connection=true";

//?? "Server=DESKTOP-HL69CK9\\PCONNOLLY; Database=QuotingBot; Trusted\_connection=true";

EnsureDatabase.For.SqlDatabase(connectionString);

var upgrader = DeployChanges

.To

.SqlDatabase(connectionString)

.WithScriptsEmbeddedInAssembly(Assembly.GetExecutingAssembly())

.LogToConsole()

.Build();

var result = upgrader.PerformUpgrade();

if (!result.Successful)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(result.Error);

Console.ResetColor();

return -1;

}

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Success!");

Console.ResetColor();

return 0;

}

}

}