4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разархивация файла перед импортом

Одна из частей программного средства для управления продажами машин осуществляет импорт машин в базу данных. Импорт осуществляется из файлов, которые предоставляют поставщики машин. При этом различные поставщики машин могут предоставлять файлы в различных форматах.

Иногда файлы предоставляются в виде архива, который содержит не только файл с описанием списка машин, но и графические данные для машин. Это могут быть их фотографии, купоны для скидок или некоторые другие данные.

Для осуществления импорта машин в систему их архива необходимо его предварительно разархивировать. Для этого был написан класс `ArchiveExtractor`, который осуществляет данное действие.

Перед импортом, осуществляется проверка расширения файла. Если выбранный файл имеет расширение .zip, то он разархивируется:

```
var extension = Path.GetExtension(filePath);
var comparer = StringComparer.IgnoreCase;

return ArchiveExt.Contains(extension, comparer);
```

После проверки файла на архив начинается непосредственно процесс разархивации файла. Для чтения файла используется сторонняя библиотека под названием SharpZipLib, которая позволяет работать с файлом архива как с объектом. При необходимости можно расширить функциональность и работать с другими типами архивов. Данная библиотека это позволяет. Однако такой задачи не стояло.

Выполняется считывание файла с помощью библиотеки:

```
FileStream fileStream = File.OpenRead(archive);
zipFile = new ZipFile(fileStream);
```

Затем итерируя по объекту, формируется полный путь к заархивированным файлам. После этого в том же каталоге создается точно такая же структура папок, которая содержится в архиве:

```
string entryFileName = zipEntry.Name;
string zipToPath = Path.Combine(folder, fileName);
var buffer = new byte[4096];
string directory = Path.GetDirectory(zipToPath);
if (directory.Length > 0)
    Directory.CreateDirectory(directory);
```

Для каждой папки архива происходит чтение в поток файла и последующая его запись в соответствующую папку. Таким образом, в том же каталоге формируется папка, которая содержит полную структуру и все файлы из архива:

```
using (FileStream writer = File.Create(ZipToPath))
{
    Stream stream = zipFile.GetStream(zipEntry);
    StreamUtils.Copy(stream, writer, buffer);
}
```

После создания структуры папок на диске и копирования в них файлов их архива, происходит закрытие потока чтения и записи и освобождения ресурсов:

```
zipFile.IsStreamOwner = true;
zipFile.Close();
```

Дополнительно после разархивации для того, чтобы экономить место на жестком диске происходит удаление архива:

```
if (File.Exists(path))
{
     File.Delete(path);
}
```

4.2 Разбор файла импорта для машин

После осуществления разархивации, в случае если файл предоставляется поставщиком машин в виде архива, осуществляется его разбор.

Импорт осуществляется для файла, который представлен в формате CSV. Данный формат представляет собой текстовый документ, который осуществляет табличное представление данных. Значение каждой колонки отделяется друг от друга специальным символом, чаще всего запятой.

Однако поставщики машин могут предоставлять файлы, значения в которых разделены различными способами. Для этого перед осуществлением разбора происходит анализ файла, и определяется, каким образом там разделены данные. Анализ файла происходит для следующих разделителей:

```
FeedFileFormat.Csv, () => new TextFeedParser(','),
FeedFileFormat.Pipe, () => new TextFeedParser('|'),
FeedFileFormat.Tab, () => new TextFeedParser('\t'),
FeedFileFormat.Inter, () => new TextFeedParser('@')
```

После того, как для разбираемого файла определен его разделитель, происходит считывание заголовка файла. В нем содержатся названия колонок файла и соответственно название полей в таблице для импортирования. Происходит считывание первой непустой строки из файла:

```
string result;
do
{
    result = reader.ReadLine();
} while (
    result != null &&
    String.IsNullOrWhiteSpace(result)
    );

return result;
```

Для хранения полей из заголовка файла бал создан специальный класс `ParsedHeader`, который содержит коллекцию строк, представляющих все считанные колонки из файла.

После считывания всех полей из файла происходит его проверка на корректность. На сайте, перед импортом, производится настройка соответствия полей из файла и полей из базы данных.

Если какие либо из полей отсутствует в файле импорта, то файл определяется как некорректный, и дальнейший его разбор не производится.

После считывания заголовка файла происходит считывание каждой строчки файла, которые представляют собой описание машины в табличном представлении:

```
var values = line.Split(delimiter, SplitOpt.None);
var fields = header.Fields
                   .Zip(values)
                   .ToReadOnlyCollection();

var parsedLine = new ParsedLine(fields);

return ProcessingResult.Create(parsedLine);
```

Для хранения каждой строчки файла также был создан специальный класс `ParsedLine`, который представляет собой коллекцию объектов полей, описывающих каждую машину в отдельности.

После выполнения разбора файла на основе прочитанных строк, осуществляется создание сущностей, которые могут использоваться в дальнейшем для импорта машин в базу данных.

Кроме того, каждый шаг этапа разбора файла логируется в файл.

Поэтому очень просто понять на каком этапе произошла ошибка и почему файл является некорректным.

Это упрощает работу с программным средством, так как при обнаружении неправильного файла можно не только сообщить об этом поставщику машин, однако и указать место, где находится ошибка.

Это позволяет быстро исправить ее и таким образом поддерживать актуальность данных на сайте максимально быстро.

4.3 Запуск импорта файлов для нескольких веб-сайтов

Для осуществления запуска импорта для нескольких файлов был создан класс `FeedImporter`, который осуществляет управление процессами запуска импорта и обработки файлов. Так один процесс импорта может одновременно обслуживать несколько сайтов, то необходимо для удобства как-то управлять очередью запуска файлов на импорт.

Для начала в методе `Import` происходит подготовка всех необходимых данных для осуществления импорта. Создаются объекты для разбора файла, происходит предварительная обработка файлов, происходит анализ файла на формат.

Затем запускается новый процесс, который осуществляет непосредственно разбор файла и последующий импорт данных в систему:

```
var taskSource = new TaskCompletionSource<bool>();
var parser = createParser(importTask.FileFormat);
var reader = File.OpenText(importTask.FilePath);
var resources = new IDisposable[] {textReader};

var parsedSource = parser.Parse(reader, importTask)
                    .UnwrapErrors(HandleItemErrors)
                    .ObserveOn(Scheduler.Default);
```

В процессе обработки файла могу происходить ошибки и исключительные ситуации. Ошибки просто логируются и выводятся в дальнейшем как дополнительная информация. Исключительные ситуации останавливают импорт и сообщают пользователю о том, что произошло. При возникновении исключительных ситуаций, дальнейший импорт данных в систему не производится:

```
if (exception == null)
{
    throw new ArgumentNullException("ex");
 }

resultBuilder.OnImportError(exception);
```

4.4 Фильтрация и вывод машин по заданным параметрам

Для отображения на сайте списков машин используются уникальные страницы. Эти страницы могут быть созданы администраторами сайта с помощью специального интерфейса в администраторской части. При создании списка машин указываются фильтры, которые могут быть применены к данному списку.

После создания такой страницы, пользователь сайта, зайдя на страницу со списком машин, может применять для них удобные для него фильтры.

При этом, кроме непосредственно обновления данных о машинах на странице, при применении определенного фильтра, также необходимо обновлять данные в самих фильтрах, чтобы исключить отображения пустых страниц.

Для реализации данной функциональности был создан класс `CarListController`. Он осуществляет управление запросами со страницы со списками машин.

Перед отображением на страницу машин, он инициализирует необходимые фильтры, которые были выбраны для нее:

```
var carListPart = services.ContentManager
      .Get(carListId, VersionOptions.Published)
      .As<CarListPart>();

var filterOptions = new CarFilterOptions
{
     IsShowFilter = carListPart.IsShowFilter,
     IsShowModelFilter = carListPart.IsModelFilter,
     IsShowBodyFilter = carListPart.IsBodyFilter,
     IsShowMakeFilter = carListPart.IsMakeFilter,
     IsShowPriceFilter = carListPart.IsPriceFilter,
     IsShowTypeFilter = carListPart.IsTypeFilter,
     IsShowDealerFilter = carListPart.IsDealFilter
};
```

После того, как пользователь применят на странице определенный фильтр, или осуществляет поиск по параметрам, содержимое страницы должно обновиться. При этом выполняется запрос к сервису на получение необходимой информации для формирования ссылки на детали машины:

```
var carProj = carListServ.GetForCarList(carList);
```

После этого определяются фильтры которые были выбраны пользователем на странице и согласно этим фильтрам происходит отбор машин для отображения на странице.

Данное действие происходит следующим обрзазом:

```
var carFilter = carListFilter.FindFilter(carList);
var carRecordList = carService.GetActive(options);
carRecordList = carFilter.Filter(carRecordList);
```

При этом в строке запроса браузера должно поменяться значение фильтра на тот, который был применен пользователем. Все фильтры передаются на сервер GET запросом и отображаются в строке соответствующим образом.

Затем переданные параметры сереализуются в объект, который затем можно использовать для определения машин для вывода на страницу пользователю.

Параметры добавляются в строку запроса с помощью добавления в коллекцию путей необходимых параметров:

```
routeData.Values.Add("Stock", options.Stock);
routeData.Values.Add("Model", options.Model);
routeData.Values.Add("Year", options.Year);
routeData.Values.Add("Body", options.Body);
routeData.Values.Add("SortBy", options.SortBy);
```

Кроме непосредственно обновления списка машин на странице необходимо осуществить обновление данных в фильтрах. Так как при применении одних фильтров зачастую данные в других меняются, то нужно убрать оттуда лишние записи, чтобы пользователь не получал пустую страницу при запросе.

Для выполнения данной операции мы также получаем список машин согласно выполняемой фильтрации. Затем для полученного списка мы проверяем, какие из значений оставшихся фильтров можно применять к данному списку. Те значения, которые приводят к пустому списку, из вывода исключаются:

```
var list = queryBefore.Select(x => x.Trim)
                          .Distinct().ToList();
var result = list
       .Where(x => !string.IsNullOrEmpty(x)
              && !string.IsNullOrWhiteSpace(x));
```

Для того чтобы вернуть значения по умолчанию для фильтров, в каждом из них присутствует опция All, которая возвращает все значения данного фильтра назад на страницу.

При этом список машин, отображаемых на странице, также обновляется и для него перестает применять данный фильтр.

4.5 Управление фильтрацией для заданных параметров

При осуществлении фильтрации на странице необходимо учесть много параметров. Так как каждая машина имеет большое количество характеристик, то был предусмотрен и уникальный фильтр для каждой из имеющихся характеристик.

Данные фильтры могут быть установлены для списков машин при их создании или при редактировании. При этом значения в списках отображаются в текстовом виде для удобства пользователя.

Однако при осуществлении фильтрации необходимо делать переход от строковых представлений данных к численным. Это необходимо, так как в базе данных параметры для машин представлены в различных форматах. Поэтому при фильтрации необходимо осуществлять конвертацию типов данных, или даже добавлять дополнительную логику для осуществления запросов.

Примером может послужить следующий случай. На страницу пользователю выводится фильтр для цен, который представлен строковым представлением некоторого диапазона.

При выборе данного фильтра на сервер приходит лишь строковое его представление. Необходимо осуществить переход к численной форме, затем отфильтровать список машин по заданным параметрам. После этого машины необходимо вернуть клиенту для отображения на страницы.

Для представления цен в системе были введены некоторые константы стандартного вида:

```
private const decimal K10 = 10000m;
private const decimal K20 = 20000m;
```

Для выбора значений цен для отображения на странице пользователя происходит выбор цен для каждой машины из полученного отфильтрованного по другим параметрам фильтра. Все найденные цены сохраняются во временную коллекцию:

```
foreach (var carRecord in carlist)
{
    if (carRecord.InternetPrice.HasValue
        && carRecord.InternetPrice > MinPrice)
        result.Add(carRecord.InternetPrice.Value);
    else if ((carRecord.InternetPrice == null
            || carRecord.InternetPrice < MinPrice)
            && carRecord.MSRP.HasValue
            && carRecord.MSRP > MinPrice)
               result.Add(carRecord.MSRP.Value;
}
```

После осуществления операции по выборы цен для списка машин получается коллекция значений всех цен, которые могут быть отображены на странице.

Затем необходимо определить, какие из диапазонов цен применимы для полученного списка машин.

Для этого производится проверка на то, кто каждая из цен в коллекции попадает в определенный диапазон для фильтрации. Если после проверки всех цен какой-то из диапазонов оказывается пустым, тогда он не включается в результирующий ответ на клиент. Таким образом, исключаются параметры для фильтрации по ценам, которые могут привести к пустому списку машин на странице.

Данная проверка осуществляется следующим образом:

```
if (price > MinPrice && price <= K10)
{
     return GetEnumDesc (Filter.Under10K);
}
if (price > K10 && price <= K20)
{
     return GetEnumDesc (Filter.Price10To20K);
}

return GetEnumDesc (Filter.Over20K);
```

Эта проверка выполняется в цикле, и при получении диапазона соответствующего некоторый цене, он сохраняется в коллекцию. Только те значения, которые попали в коллекцию, выводятся на странице пользователя:

```
IList<string> result = new List<string>();

foreach (var price in pricesList)
{
    var range = GetRangeForPrice(price);

    if (!result.Contains(range))
    {
        result.Add(range);
    }
}

result = result.OrderBy(x => x).ToList();
```

Таким образом, реализуются требования по отображению в фильтрах тех параметров, выбор которых на странице не приведет к пустому списку.

4.6 Отображение набора акций для машин

В разрабатываемом программном средстве имеется функциональность по добавлению определенных акций к машинам. Функционируют они по несложному алгоритму.

Администраторы сайта имеют возможность создать специальные акции для машин. При создании они выбирают параметры, по которым акция будет применяться на наборы машин. Например, такими параметрами могу быть тип машины, её номер, год выпуска, цена и т.д.

После создания акций для машин, есть возможность создавать специальные списки для них. На таких списках отображаются все машины, на которые действует та или иная акция.

При этом имеется функциональность добавления к специальному списку машин ранее созданные акции. Кроме того, на одну машину может действовать сразу несколько акций. Это все нужно учитывать при выводе машин в списках и отображении для них акций.

Дополнительно для акций существуют сроки действия. Поэтому необходимо предусмотреть возможность получения действующих в системе акций, у которых срок действия меньше текущей даты:

```
var now = DateTime.Now;

IQueryable<SpecialRecord> specialQuery = repositiry
    .Table
    .Where(
       s =>
       s.ActiveDateFrom <= now
       && s.ActiveDateTo >= now
       &&(
         (s.SpecialText != null
             && s.SpecialText != string.Empty)
         || (s.DiscountPrice != null
             && s.DiscountPrice > 0)
         || (s.DiscountText != null
             && s.DiscountText!=string.Empty)
       )
    );

return    specialQuery.OrderByDescending(x=>x.Id)
                      .ToList();
```

После того, как акции созданы в системе, нужно добавить их для отображения на списках машин. Для этого, при создании списка, необходимо выбрать их из списка ранее созданных.

При этом акции можно динамически переназначать на списки. Поэтому предусмотрен механизм обновления акций для списков.

При обновлении акций для списков мы сначала получаем все акции, которые были применены к нему ранее. Выбираем из них те, которые были отменны администратором и удаляем со списка:

```
var notUsed = oldSpecialForList
        .Where(sls => sls.SpecialRecord != null
        &&!specialIds.Contains(sls.SpecialRecord.Id
        ));

foreach (var specialListSpecialRecord in notUsed)
{
    sLSRepository.Delete(specialListSpecialRecord);
}
```

Затем мы определяем те из выбранных, которые нужно добавить к списку и создаем их:

```
var oldSpecialForListIds = oldSpecialForList
                .Select(x => x.SpecialRecord.Id);

var newIds = specialIds
                .Where(id => !oldSpecialForListIds
                .Contains(id))
                .ToArray();

var specialsForAdd = specialService
                        .GetAll(newIds);

foreach (var specialRecord in specialsForAdd)
{
     sLSRepository
         .Create(new SpecialListSpecialRecord
                {
                   SpecialListRecord = listRecord,
                   SpecialRecord = specialRecord
                });
}
```

При выводе машин в списке необходимо отобразить для них акции. Для определения, какая акция действует на текущую машину, сначала выбираются все акции для списка. Затем для каждой акции проверяются её параметры, и если они совпадают с машиной, то данная акция отображается.