Attachments

# Общая архитектура

Вложение (attachment) представляет собой отдельную сущность. У нее есть свойства Name, Mime и непосредственно данные (Stream).

Эта сущность может храниться как в отдельной таблице для каждого элемента, так и в общей таблице вложений c дополнительной таблицей связей.

Для предотвращения несанкционированного доступа к вложениям используется механизм формирования токенов доступа. Токен это просто хеш строки, содержащей идентификатор текущей сессии, идентификатор пользователя и дополнительный ключ доступа (guid) связанный с этим вложением. Проще всего добавить этот ключ в виде поля с типом uniqueidentifier и значением по умолчанию newid().

Работа с вложениями фактически состоит из трех практически независимых частей.

1. Отображение списка вложений. Тут все просто и можно использовать стандартные средства платформы. Однако загружать сам поток данных сразу не получится. Нужны какие-то отдельные инструменты для работы с этим потоком. Простейший вариант – загрузка (download) данных по клику на ссылке. Если мы работаем с картинками, имеет смысл показать их Preview. Кроме того, для предотвращения несанкционированного доступа необходимо сформировать токен доступа. Для этого в набор данных нужно включить поле со специальным типом !Token. При обработке этого поля система возьмет значение (это будет guid) и сформирует вместо него токен доступа.  
   Обратите внимание, что значения этого токена будут разными в разных сессиях работы с браузером.
2. Получение содержимого вложения с сервера (картинка или загрузка) с использованием токена доступа.
3. Выгрузка (upload) вложения на сервер. Сохранить данные можно где угодно (например в БД). У этого вложения появляется идентификатор. По нему можно получить содержимое для загрузки или отображения картинки. Обратите внимание, что в саму модель этот идентификатор запишется только при сохранении модели. Кроме идентификатора загрузки при сохранении данных сформируется токен доступ. Его тоже нужно записать в модель (сохранять не нужно, при загрузке он все равно сформируется заново).

В модели за работу с файлами вложений отвечает раздел files в model.json. Как обычно, это объект, где имя свойства определяет имя действия, а значения – описывающий их объект. Каждое свойство – одно действие.

Структура объекта для действия с файлами

{  
 files: {  
 actionName: {  
 type: 'sql'|'clr'|'parse',  
 model: 'modelname',  
 clrType: 'assemblytype'  
 parse: 'excel'  
 }  
 }  
}

В этом документе будем рассматривать только хранение вложений в БД.

# Пример с пояснениями

Пусть нам нужно сохранять в БД вложения к контрагенту. Будем хранить все это в трех таблицах – контрагенты, вложения и связи между ними.

## Таблицы

-- таблица контрагентов  
create table a2.Agents (  
 Id bigint,  
 Name nvarchar  
)

-- таблица вложений  
create table a2.Attachments (  
 Id bigint,  
 Name nvarchar(255),  
 Mime nvarchar(255),  
 Stream varbinary(max)  
 AccessKey uniqueidentifier constraint DF\_Attachments\_AccessKey default (newid())  
)

-- таблица связей  
create table a2.AttachmentLinks (  
 AgentId bigint,  
 AttachmentId bigint  
)

## Конечная точка

Конечная точка пусть будет \catalog\agents. Файл model.json:

{  
 "schema": "a2",  
 "model": "Agent",  
 "dialogs": {  
 "editAgent": {  
 "model": "Agent",  
 "view": "editAgent.dialog",  
 "template": "editAgent.template"  
 }  
 },  
 "files": {  
 "attachment": {  
 "type": "sql",  
 "model": "Agent.Attachment"  
 }  
 }  
}

Свойства контрагента редактируются в диалоге.

В файле указан раздел files с единственным действием – это означает что можно будет работать с вложениями с использованием пути \catalog\agent\attachment.

## Форматы хранимых процедур

Для работы с вложением используются две хранимые процедуры. Одна для чтения вложения (суффикс .Load), вторая для его записи (суффикс .Update). Процедуры НЕ стандартные.

### Процедура .Load

Эта процедура получает следующие параметры:

* @TenantId int = null – только в мультитенантном окружении,
* @UserId bigint – идентификатор пользователя,
* @Id bigint – идентификатор вложения

Процедура должна вернуть единственный набор со следующими полями (имеют значение имена полей, порядок не важен)

* Id – bigint – идентификатор вложения
* Mime nvarchar(255) – mime тип
* Name nvarchar(255) – имя вложения,
* [Size] bigint (не обязательно) – размер вложения в байтах
* Stream varbinary(max) – сам поток данных
* Token uniqueidentifier – поле для формирования токена доступа. Этот элемент должен соответствовать тому. что возвращет процедура загрузки модели, из которой будет вызываться это вложение.

### Процедура .Update

Эта процедура получает следующие параметры:

* @TenantId int = null – только в мультитенантном окружении,
* @UserId bigint – идентификатор пользователя,
* @Name nvarchar(255) – имя вложения
* @Mime nvarchar(255) – mime тип
* @Stream varbinary(max) – сам поток данных

Процедура должна вернуть единственный набор со следующими полями (имеют значение имена полей, порядок не важен)

* Id – bigint – идентификатор вложения
* Token – uniqueidentifier – поле для формирования токена доступа.

## Модель

{  
 Agent: {  
 Id:50,  
 Name: "My first agent"  
 Attachments:[  
 LinkId: 234 /\*Id связи – сохраняется при сохранении модели \*/,  
 **Id**: 11, /\*Id вложения из таблицы Attachments \*/  
 Token: " vYFirX6IDEQrmgKo16Xqmp3N-g7Iu0XyocHBGM2Rdzw1",  
 **Name**:"image1",  
 **Mime**:"image/png",  
 ]  
 }  
}

Обратите внимание, что значение поля Token это не сам guid вложения, а сформированное системой специальное знечение.

## Разметка

Сам диалог редактирования свойств контрагента вполне обычный.

Фрагмент разметки, связанный с работой с вложениями.

<Tab Header="Attachments">  
 <Header Content="Attachments"/>  
 <List ItemsSource="{Bind Agent.Attachments}" Style="Underlined">  
 <ListItem Icon="FileContent">  
 <ListItem.Header>  
 <Hyperlink Content="{Bind Name}"   
 Command="{BindCmd File,  
 Url='/catalog/agent/attachment',   
 Argument={Bind Id}, FileAction=Print}"/>  
 </ListItem.Header>  
 <ListItem.Content>  
 <FlexList Orientation="Horizontal">  
 <Span Content="{Bind Id}"/>  
 <Span Content="{Bind Mime}"/>  
 <FileImage Url="/catalog/agent/attachment" Value="{Bind Id}"   
 Width="50" Height="50"/>  
 </FlexList>  
 </ListItem.Content>  
 <ListItem.CommandBar>  
 <CommandBar>  
 <Button Icon="Clear" Command="{BindCmd Remove, Argument={Bind}}"/>  
 </CommandBar>  
 </ListItem.CommandBar>  
 </ListItem>  
 </List>  
 <UploadFile Url="/catalog/agent/attachment" Delegate="uploadAttachment"  
 Accept="application/pdf,image/\*"/>  
</Tab>

### Используемые элементы

List – обычный список для отображения списка вложений. Связан с коллекцией Agent.Attachments.

Hyperlink – связан с командой (Сommand = "File") работы с файлом. Свойства команды:

* Url – путь к конечной точке вместе с действием над файлом (/catalog/agent/attachment – действие file.attachment в конечной точке /catalog/agent)
* Argument – идентификатор вложения
* FileAction – что делать с вложением. Возможные значения:
  + Show (default) – показать в отдельном окне
  + Download – загрузить как файл
  + Print – сразу открыть в окне печати

Button – кнопка удаления связана с командой Remove и просто удаляет элемент из коллекции.

UploadFile – элемент управления для загрузки файла на сервер. Свойства элемента:

* Url – путь к конечной точке вместе с действием над файлом (/catalog/agent/attachment – действие file.attachment в конечной точке /catalog/agent)
* Accept – (не обязательно) допустимые mime типы файлов.
* Delegate – делегат, который будет вызван после сохранения выбранных файлов. В качестве аргумента ему будет переда массив объектов с результатом добавления вложений в БД.

## Шаблон

В данном случае нас интересует только делегат. Фрагмент шаблона:

const template = {  
 delegates: {  
 uploadAttachment  
 }  
};  
  
module.exports = template;  
  
function uploadAttachment(result) {  
 console.dir(this);  
 console.dir(result);  
 let coll = this.Agent.Attachments;  
 for (let i = 0; i < result.length; i++)  
 coll.$append(result[i]);  
}

# Токены безопасности

Для предотвращения несанкцинированного доступа к данным предусмотрен механизм токенов безопасности. Он работает следующим образом.

При формировании ссылок на вложения (обычно это будет объект TAttachment в котором будет какой-то идентификатор к модели добавляется специальное поле c типом !Token. Это поле типа uniqueidentifier.

В процессе построения модели данных, значение этого поля будет использовано для формирования токена безопасности (с использованием Id сессии и Id текущего пользователя).

При попытке открыть/просмотреть/загрузить вложение система формирует ссылку с указанием этого токена безопасности.

Например фрагмент кода:

select [Attachments!TAttachment!Object] = null, [Id!!Id] = Id, [Name!!Name] = Name,  
 [Token!!Token] = AccessKey  
from a2.Attachments where …

Сформирует такую модель:

{  
 Id: 123,  
 Name: "MyFileName",  
 Token: "vYFirX6IDEQrmgKo16Xqmp3N-g7Iu0XyocHBGM2Rdzw1"  
}

При работе с вложением (например методом $attachment конроллера будет сформирована такая ссылка:  
/attachment/show/123?base=path/to/elem&token=vYFirX6IDEQrmgKo16Xqmp3N-g7Iu0XyocHBGM2Rdzw1

Отметим, что в метаданных модели запоминается имя свойства токена. (element.\_meta\_.$token). Это означает, что фактически имя свойства может быть любым (по соглашению обычно называется именно Token), но важно не забыть пометить поле специальным типом !Token.

Если рассматривать реализацию, то токен это просто хеш SHA256 от строки $"\_{sessionId}\_{token}\_{userId}\_".

При открытии ссылки при помощи процедуры загрузки (которая вернет объект для токена) будет построен соответствующий токен безопасности.  
Если построенный токен не соответствует указанному в запросе – будет сгенерирована ошибка 403.

Обратите внимание, что сам токен (guid) ни при каких обстоятельствах не должен попадать на клиент. Иначе, зная алгоритм хеширования и Id сеанаса можно вычислить хеш и получить доступ к вложению.