Компьютерная программа A2v10

Общее описание

Автор: Кухтин Александр Анатольевич

Оглавление

Назначение и фунции

Система предназначена для работы в качестве front-end решения для автоматизации работы предприятия.

Доступ к системе осуществляется черз web-браузер и не требует установки каких-либо клиентских библиотек.

Система предоставляет разработчику высокоуровневый предметно-ориентированный «конструктор» прикладных решений. Прикладное решение является «динамическим» в том смысле, что после изменений в решении, не требуется компиляции проекта и его развертывания.

Модель хранения подразумевает использование SQL-совместимой базы данных.

Прикладная логика разрабатывается на языке JavaScript. Представления бизнес-объектов описываются на языке Xaml и не требуют работы с HTML и CSS.

Платформа

Система реализована с использованием технологического стека .NET корпорации Microsoft.

В качестве СУБД используется Microsoft SQL Server.

Для реализации клиентской части web-приложения используется JavaScript фреймворк Vue.JS

Для построения отчетов и печатных форм применяется отчетная система Stimulsoft Reports.Web.

Используемые технологии

Технология	**Используется для**
Microsoft ASP.NET.MVC	Web-приложение
Internet Information Server (IIS)	Хостинг Web-приложения
Microsoft Chakra (ChakraCore)	Серверная обработка JavaScript

VueJS	Клиентская часть Web-приложения
T-SQL	Работа с БД
Windows Workflow Foundation	Работа с бизнес-процессами
Windows Communication Foundation	Интеграция с внешними системами (SOAP)
StimulSoft Reports.Web	Отчеты и печатные формы документов
Ninject	Библиотека поддержки DI
XAML	Реализация представлений

Аутенификация пользователей стандартная для технологии ASP.NET с использованием протокола https.

Архитектура системы

Система реализована как web-приложение на платформе Microsoft ASP.Net.MVC.

Клиентская часть приложения написана как SPA-приложение (Single Page Application) с использованием JavaScript-фреймворка VueJS.

Следует отметить, что прикладное решение использует высокоуровневые абстракции, связанные с предметной областью и не опускается на уровень html/css и взаимодействия с браузером и сетью.

Модель

Основным элементом системы с точки зрения прикладного программиста является **модель данных** (далее **модель**). Модель описывает один элемент предметной области и включает следующие элементы

- Данные
- Поведение
- Представления

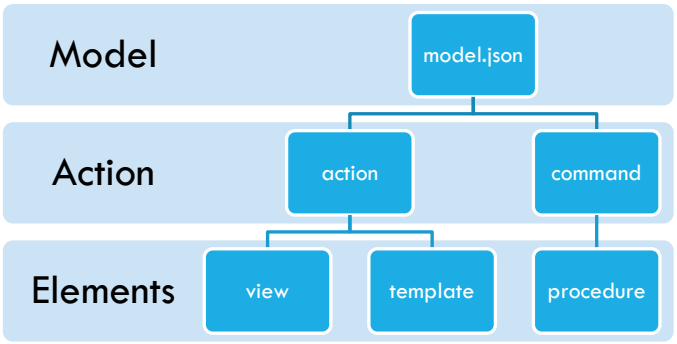
Модель включает все данные, которые необходимы для ее обработки. Обратите внимание, что понятие модели «высокоуровневое», то есть не привязано к конкретным таблицам базы данных.

Примером прикладной модели данных может служить «приходная накладная».

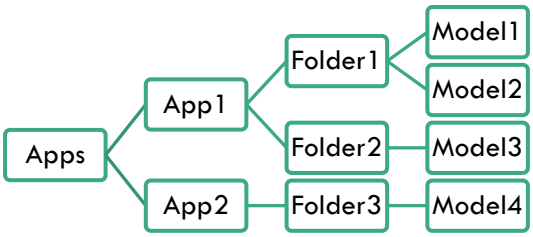
Модель поддерживает действия (**actions**). Для каждого действия может задаваться шаблон поведения (**template**) и представление (**view**).

Кроме действий модель поддерживает команды (commands), которые служат для выполнения каких-либо операций со всей моделью.

Физически модель представляет собой каталог с файлами. Обращение к модели осуществляется по относительному пути к каталогу модели. Основным файлом в этом каталоге является файл **model.json**. В этом файле описана структура модели и есть ссылки на остальные файлы.



Прикладное приложение фактически представляет собой каталог в котором хранятся описания моделей. Приложения хранятся в общем каталоге **Apps**, ссылка на который указывается в файле конфигурации приложения.



Запрашиваемая модель определяется системой роутинга web-приложения и, фактически, указывается в URL-запросе.

Архитектура хранения данных

Данные хранятся в реляционной базе данных Microsoft SQL-Server. Для доступа к данным система всегда вызывает хранимые процедуры и никогда не использует «сырой SQL». Такой подход позволяет исключить атаки типа sql-injection и обеспечивает высокое быстродействие за счет предварительной компиляции планов запросов. Кроме этого, многие бизнес-задачи реализованы на языке T-SQL, что тоже значительно повышает быстродействие.

Структура хранения – классическая нормализованная реляционная БД. Примеры таблиц:

Схема	*Таблица*	*Описание*
a2	Documents	Таблица документов
a2	Agents	Контрагенты
a2	Entities	Таблица объектов учета (товаров, услуг и т.д.)
a2security	Users	Системная таблица – пользователи БД.

Структура БД

Структура базы данных – стандартная реляционная нормализованная.

Маршрутизация

Маршрутизация запросов из клиентской части приложения фактически представляет собой прямое отображение на структуру папок приложения. Например, запрос с
`URL = \document\invoice\edit`
вызывает действие **edit** в папке `.\document\invoice` текущего активного приложения.

Дальнейшие действия зависят от того, что написано в разделе actions\edit файла `.\document\invoice\model.json`.

Данные

Атрибуты **model** и **schema** корневого элемента файла **model.json** представляет собой основу для формирования имен хранимых процедур для работы с данными.

Для получения данных система всегда вызывает хранимую процедуру. Имя этой процедуры зависит от текущего действия (**action**). Это может быть элемент или список элементов. Для действия с элементом вызывается процедура с суффиксом «**.Load**», а для списка элементов – процедура с суффиксом «**.Index»**.

Хранимая процедура возвращает произвольное количество наборов данных (**recordset**) в заданном формате. Для получения дополнительных метаданных используется специальный синтаксис имен полей. В процессе обработки этих наборов и их метаданных формируется иерархический объект, который и представляет собой модель данных.

Например, при загрузке модели `document\invoice` загружается информация в виде объекта **Document** (документ), у которого есть вложенные объекты **Agent** (контрагент), **Warehouse** (склад) и т.д.

Пример модели:

Тело хранимой процедуры:

```
-- первый набор – документ
select [Document!TDocument!Object] = null, Id = DocumentId, [Date] =
      DocumentDate, No=DocumentNo,
      [Agent!TAgent!RefId] = AgentId,
      [Warehouse!TWarehouse!RefId] = WarehouseId
from a2.Documents where DocumentId = @Id;


-- второй набор – контрагент
select [!TAgent!Map], [Id!!Id] = Id, [Name] = Name, RegDate
from a2.Agents where Id in (...)

-- третий набор – склады
select [!TWarehouse!Map], [Id!!Id] = Id, Name, Address, Capacity
from a2.Warehouses where Id in (...)
```

Результат (json):

```
{
      Document: {
            Id: 400,
            Date: "2017.11.20T00:00:00Z",
            No: "OS-2013",
            Agent: {
                  Id: 1123,
                  Name:"Company name",
                  RegDate: "1994.10.05T00:00:00Z"
            }
            Warehouse: {
                  Id: 2233,
```

```
                Name: "Основной склад",
                Address: "Киев",
                Capacity: 10000
            }

    }
}
```

Модель данных это не просто объект JavaScript. Это специальный объект, который ведет себя как обычный объект, но в то же время обладает реактивностью. При изменениях данных автоматически обновляются все представления и происходят события. События могут обрабатываться в шаблоне.

Шаблоны

Шаблоны поведения представляют собой модули на языке JavaScript и представляют собой обычные программы. Шаблоны бывают клиентские и серверные. Клиентский шаблон выполняется как на клиенте, так и на сервере, серверный – только на сервере.

Шаблон представляет собой объект, которые может включать следующие свойства:

properties	Вычисляемые свойства, которые добавляются к объектам модели
events	Обработчики событий, которые происходят в модели.
commands	Команды, которые могут выполняться как на клиенте, так и на сервере.
validators	Правила проверки достоверности отдельных элементов модели.

Вычисляемые свойства представляют собой мощный механизм, позволяющий сильно упростить реализацию сложной бизнес-логики приложения. Такое свойство, это функция, которая вызывается при попытке чтения свойства. Возвращаемый функцией результат рассматривается как значение свойства. При расчетах функция может обращаться к другим вычисляемым свойствам и т.д. Таким образом легко строить обработчики зависимых данных без использования модели событий.

Вычисляемые свойства могут как дополнять существующие в модели статические свойства, так и заменять существующие. Например, если есть документ с несколькими строками, а общая сумма должна быть равна сумме всех строк, то достаточно определить вычисляемое свойство **Document.Sum**, которое будет суммировать суммы по всем строкам документа. При любых изменениях свойство будет вычислено заново и сумма по документу будет всегда правильной.

Обработчики событий представляют собой функции, которые вызываются при изменениях данных. Привязка обработчика выполняется по имени события с полным путем к фрагменту данных. Например, запись

```
events: {
    "Document.Agent.Name.change": agentNameChange
}
```

означает, что при изменении имени контрагента (любым способом, будь то программой или представлением) будет вызвана функция `agentNameChange.`

Обратите внимание, что использование вычисляемых свойств резко снижает необходимость работы с событиями

Команды в шаблоне используются в случае, когда нужно получить управление при выполнении пользователем каких-либо действий (например, пользователь нажал на кнопку формы).

Команды могут выполнять любые действия, в том числе выполнять серверные команды, показывать диалоги и т.д. При работе с командами широко используется механизм обещаний (**Promises**) языка **JavaScript**.

Серверные шаблоны

При обработке действий на сервере могут использоваться серверные шаблоны. Они предназначены для выполнения каких-либо действий с моделью.

Отметим, что при работе на сервере используются и клиентские шаблоны тоже. Совместное использование клиентских элементов проверки достоверности и серверных шаблонов позволяет полностью исключить «подделку» данных (даже авторизованными пользователями) и полностью быть уверенным в правильности сохраняемых данных.

Кроме того, серверные шаблоны могут включать обработчики серверных событий (например, **beforeSend** – перед отправкой. Такие обработчики могут менять данные (например, «вырезать» лишние объекты из модели) без необходимости менять sql-код загрузки модели.

Серверные шаблоны, совместно с представлениями, поддерживают механизм утверждений (clauses). Этот механизм позволяет выполнять условный рендеринг элементов представления.

Например, если текущий пользователь не должен видеть финансовых данных и есть соответствующее правило, то в клиентское приложение вообще не будет передаваться код разметки и данные связанные с финансами.

Представления

Представления могут быть написаны на языке XAML или html. Основным языком является XAML.

Представления на языке XAML являются высокоуровневым представлением разметки (аналог технологий WPF и Silverlight) и транслируются в html для работы с текущей моделью данных.

Для связывания элементов управления в данными используется механизм двусторонних привязок (Binding). При изменении данных модели – представление автоматически обновляется и наоборот.

В представлениях реализован механизм утверждений (clauses), который позволяет реализовать условную трансляцию элементов в зависимости от условий.

В отличие от html, язык Xaml является «жестким», то есть недопустимые теги и правила вызывают исключения. Это позволяет упростить работу прикладного программиста.

Безопасность

Безопасность системы обеспечивается несколькими принятыми техническими решениями. Они работают совместно и обеспечивают надежную и устойчивую работу системы.

Аутентификация

Аутентификация пользователей использует стандартные решения Microsoft.Owin.Security. Для коммуникации с сайтом используется SSL с доверенными сертификатами безопасности.

Пароли пользователей не хранятся в открытом виде. Хранятся только хеши паролей (с «солью»). Таким образом никто (даже администратор системы) не имеет доступа к паролям пользователей.

Смена пароля выполняется пользователем самостоятельно. Предусмотрена реализация политики паролей (проверка сложности, политика смены пароля и т.д.)

Разделение сайтов

Для обеспечения безопасности можно использовать разделение сайтов. Например, может быть развенруто два сайта – один основной (интранет) и внешний (интернет). Внешний сайт (фактически это сайт компании), на котором реализована система on-line продаж, представляет собой отдельный сайт во внешней (видимой из интернета) подсети. База данных этого сайта изолирована от основной. Обмен данными между базами производится путем синхронизации специальными средствами. Таким образом, даже при компрометации внешнего сайта, основная база данных системы остается недоступной.

Списки контроля доступа

Для авторизации пользователей используется концепция "списка контроля доступа" (ACL).

К любой сущности системы (например – вид документа, элемент интерфейса) может быть привязан список контроля доступа. Такой список содержит перечень пользователей или групп пользователей) и флажки разрешений/запретов для различных действий с сущностью (видимость, редактирование, удаление, и т.д.). Причем разные сущности могут иметь различный набор флажков. Например, для документа есть флажок «Проведение», а для элемента интерфейса такого флажка нет.

Для флажков используется трехзначная логика (разрешено, запрещено, не установлено). Такой подход обеспечивает высокую гибкость настройки доступа.

При попытке выполнения действия над любой сущностью, сначала проверяется список контроля доступа и, если там установлен запрет – действие отклоняется.

Разделение интерфейсов

Для более гибкого управления доступом в системе имеется несколько интерфейсов пользователя. На текущий момент используется три интерфейса: основной, интерфейс агента и интерфейс администратора.

Выбор интерфейса выполняется динамически при входе пользователя в систему.

Например, в интерфейсе агента по продажам имеются только необходимые для него элементы. Таким образом повышается надежность управления доступом. Администратор просто не может ошибиться при раздаче прав на элементы интерфейса агента по продажам.

Для упрощения отладки администратор системы имеет возможность сменить текущего пользователя и выполнять действия от его имени без необходимости повторного входа. Эта технология не требует знания пароля пользователя, что дополнительно повышает безопасность.

Бизнес-процессы

Для работы с бизнес процессами используется технология «Windows Workflow Foundation». Бизнес – процессы представляют собой программы, которые выполняются «эпизодами» в зависимости от внешних условий и событий.

Программы бизнес-процессов разрабатываются в визуальном конструкторе, который входит в состав Microsoft Visual Studio.

Помимо типовых блоков бизнес процессов, разработаны специальные блоки (Activities), которые позволяют связывать бизнес-процессы с сущностями системы, такими как, документ, объект учета, контрагент и т.д.

Одна и та же сущность может обрабатываться различными бизнес-процессами, как параллельно, так и последовательно.

В системе, как правило, используются бизнес-процессы, построенные как конечный автомат (state machine). В этом случае процесс описывается как совокупность состояний и переходов между ними. Переходы инициируются триггерами (внешними событиями).

Некоторые процессы могут быть приостановлены по таймеру. Этот механизм используется для того, чтобы поставить исполнителю задачу с конкретным сроком исполнения.

Возможна параллельная обработка задач в бизнес-процессах.

Для обработки таймеров и автоматического запуска процессов, созданных внешними по отношению к системе модулями разработан специальный сервис фоновой обработки.

Этот сервис работает как служба Windows и работает независимо от хост-процесса поддержки приложения.

Интеграция с внешними системами

Для интеграции с внешними системами используется технология Windows Communication Foundation (WCF). Основным протоколом реализации интеграции является SOAP (поскольку это требование внешних систем). В случае необходимости возможна работа по протоколу REST.