# Desktop приложения

## Популярни софтуерни архитектури

Преди да се запознаем в детайли с desktop приложенията ще споменем няколко често прилагани софтуерни архитектури.

### Трислойна архитектура

Трислойната архитектура предполага разбиване на имплементацията на софтуерната система на три отделни слоя – слой за достъп до данните, бизнес слой и презентационен слой. Всеки от слоевете е проектиран така, че да не зависи от вътрешната реализация на другите слоеве. Това позволява лесна промяна в имплементацията на един от слоевете, без да се налагат промени в другите слоеве.

* Слоят за достъп до данните изолира другите слоеве от конкретната използвана СУБД, както и от вътрешната структура на базата данни. Структурата на данните, които се предоставят на бизнес слоя, е отражение на предметната област.
* Бизнес слоят съдържа реализацията на желаното поведение на системата (бизнес логиката). Той използва слоя за достъп до данните, за да чете/записва данни в базата данни, и предоставя на презентационния слой всичко, което му е необходимо за транформиране на данните в удобен за потребителя вид.
* Презентационният слой получава пълната информация за данните в системата и я трансформира в подходяща за консумация форма. Това най-често е потребителският интерфейс на приложението, но може да включва и други модули за въвеждане и/или извеждане на информация, като например сканиране и печатане на документи и т.н.

Голяма част от реалните софтуерни системи ползват някакъв вариант на трислойна архитектура.

### Клиент/сървър системи

Клиент/сървър системите са разпределени софтуерни системи, които най-често са разделени на един сървър и множество клиенти. Сървърът на системата обслужва клиентите осигурява централизиран достъп до данните. Комуникацията между клиент и сървър се инициира от страна на клиента.

Клиент/сървър системите обикновено имат трислойна (или подобна) архитектура. В зависимост от това кои слоеве са разположени в сървъра и кои – в клиента, се различават тънки и дебели клиенти.

#### Тънки клиенти (thin clients)

Тънките клиенти съдържат единствено презентационния слой на системата (или дори само част от него). Предимството им е, че осигуряват минимален брой пролуки в сигурността на данните (тъй като всички операции с тях се случват на сървъра) и не изискват наличие на допълнителна инфраструктура (например библиотеки) на клиентската машина. Основният недостатък е пълната зависимост от сървъра за манипулация с данните.

#### Дебели клиенти (thick clients)

Дебелите клиенти съдържат поне част от бизнес слоя на системата (а може да съдържат дори целия бизнес слой и част от слоя за достъп до данните). Предимството им е, че предоставят възможност за лесно разширяване на поведението на клиентите, но с цената на по-ниска сигурност на данните и по-голяма зависимост от външни за системата фактори.

### Peer-to-Peer системи

P2P системите са разпределени софтуерни системи, отделните части на които са равноправни помежду си. Предимството им пред клиент/сървър системите е липсата на централизираност, водеща до по-висока надеждност. Недостатъците им са понижена сигурност на данните и по-голяма сложност на реализацията.

## Desktop приложения. Предимства и недостатъци

Desktop приложенията са приложения, които взаимодействат с потребителя чрез прозорци (форми) и в общия случай се изпълняват в самостоятелно (изолирано от останалите процеси на машината).

### Предимства

* Предоставят лесен достъп до ресурсите на операционната и файловата система;
* По-лесни са за реализиране на сложно поведение на потребителския интерфейс;
* Удобни са за реализиране на дебели клиенти;
* Очевиден избор са за създаване на локални (неразпределени), както и на P2P приложения.

### Недостатъци

* Възможно е да изискват инсталиране на допълнителни библиотеки за да работят;
* Трудни са за обновление когато са част от клиент/сървър система;
* Обикновено са тясно свързани с операционната система (най-очевидното изключение е Java).

## Структура на desktop приложенията

### Прозорци

Потребителският интерфейс на desktop приложенията е разделен на прозорци (форми). Обикновено един от прозорците е главен. При някои desktop приложения изпълнението приключва след затварянето на главния прозорец, а при други – след затварянето на всички отворени прозорци.

Прозорците имат видима област – обикновено правоъгълна, с определна височина и ширина. Обикновено имат заглавна лента и системни бутони. Някои прозорци са с фиксиран, а други – с променлив размер.

### Контроли

Всеки прозорец може да съдържа множество контроли във видимата си област. Контролите са визуални компоненти с определено предназначение – например поле за въвеждане на текст, радио бутон или дървовиден изглед. Когато е необходимо специално поведение, което стандартните контроли не могат да предоставят, е предвидено създаването на потребителски контроли с поведение, имплементирано от програмиста.

Някой контроли играят ролята на контейнери, които могат да съдържат други контроли. Така на практика всеки прозорец съдържа дървовидна йерархия от контроли, в чийто корен стои самият той.

### Събития

Събитията предоставят начин за изпълнение на дефинирани от програмиста действия в определени ситуации – например при натискането на бутон във формата, избирането на елемент от списък или въвеждането на символ в текстово поле. Всяка контрола предоставя набор от събития, към които могат да бъдат закачени един или повече методи, които да се изпълнят при настъпване на съответното събитие.

### Message Loop

За разлика от конзолните приложения, в desktop приложенията програмистът не се занимава с имплементацията на основната част от кода на програмата. Почти цялото изпълнение на програмата протича в специален цикъл, наречен *message loop* и реализиран в имплементацията на платформата. Това е цикъл, който на всяко извъртане изтегля съобщение от специална опашка за съобщения (*message queue*), определя получателя на съобщението и му го предава за обработка. Болшинството съобщения се поставят в опашката на приложението от операционната система. Всяко съобщение описва някакво настъпило събитие – например преместване курсора на мишката, натискане на клавиш на клавиатурата, преместване на прозорец. Възможно е също програмно изпращане на съобщение от програмиста до опашката на приложението или дори до опашката на друго приложение.

Тъй като почти целия код, написан от програмиста в едно desktop приложение, е разположен в методите – обработчици на събития, то този код винаги се изпълнява по време на обработката на някакво съобщение в message loop-а.

Обработката на съобщенията в message loop-а се случва последнователно. Не се препоръчва продължителното изпълнение на код в някой от методите – обработчици на събития, понеже това води до забаяне в обработката на следващите съобщения, създавайки впечатлението, че приложението е „замръзнало” (not responding).

### Модални диалози

Модалните диалози са прозорци, които ограничават достъпа до всички други прозорци на приложението, докато не бъдат затворени. При затварянето на модален диалог се указва модален резултат (OK, Cancel, Yes, No и т.н.), който да бъде предаден обратно на кода, предизвикал отварянето на модалния диалог.

Любопитен факт: имплементацията на модален диалог кара message loop-а на приложението да пренебрегва всички съобщения, отнасящи се до другите прозорци (или създава заменя главния message loop със собствен, който има желаното поведение); така се създава илюзията, че модалният диалог „покрива” всичко останало.

# Windows Forms

Платформата Windows Forms е част от .NET Framework, предназначена за създаването на managed desktop приложения. Тя предоставя стандартния за Windows набор от контроли и в основата си може да се разглежда като managed wrapper на Win32 API. Разбира се, Windows Forms предоставя и много допълнения към тази стандартна палитра.

Създаването на Windows Forms приложение започва със създаване на Windows Forms Application проект. Той автоматично съдържа една празна форма и главен метод на приложението, който създава инстанция на формата и я показва на екрана.

Windows Forms класовете са имплементирани в библиотеката System.Windows.Forms и са достъпни през едноименното пространство от имена.

## Форми

Формите (прозорците) в Windows Forms наследяват класа Form. За създаване на нова форма Visual Studio предоставя Windows Form файлов шаблон. Някои от свойствата на формите са:

* Size – това е размерът на формата;
* Text – това е текстът, който се изписва в заглавната лента;
* StartPosition – определя къде да бъде разположена инстанция на формата при създаването си (на определеното по подразбиране от Windows място, в центъра на екрана, на фиксирана от разработчика позиция);
* FormBorderStyle – определя стила на рамката на формата (дебела, тънка, разрешаваща промяна на размерите или не и др.);
* MinimizeBox, MaximizeBox – указват дали да бъдат видими съответно бутона за минимизиране и максимизиране на формата;
* ShowInTaskbar – указва дали формата да има асоцииран с нея бутон в лентата със задачи на Windows;
* ShowIcon – указва дали формата да има иконка в лявата част на заглавната си лента;
* AcceptButton – указва бутон от формата, който да бъде автоматично натиснат при натискане на клавиша Enter;
* CancelButton – указва бутон от формата, който да бъде автоматично натиснат при натискане на клавиша Escape.

По подразбиране Windows Forms приложенията се стартират с инстанция на главна форма, така че изпълнението на приложението автоматично да приключи при нейното затваряне.

За показване на екрана на допълнителни форми (които не са модални) се използва методът Show().

Формите предоставят богат набор от събития, позволяващи включването на код на разработчика в критични моменти, като например преди изобразяване на екрана или при опит за затваряне на формата.

## Контроли

Windows Forms предоставя стандартния набор от Windows контроли – Label, TextBox, ListBox, ComboBox, CheckBox, RadioButton, Button, ListView, TreeView. MonthCalendar и DateTimePicker се използват за избор на дата и/или час. RichTextBox се използва за реадктиране на текст с богато форматиране (цвят, шрифт и т.н.). WebBrowser служи за визуализиране на HTML документи.

Повечето контроли имат следните свойства:

* Parent – това е родителската контрола (контейнер) на контролата.
* Size – това е размерът на контролата;
* Location – това са координатите на горния ляв ъгъл на контролата спрямо горния ляв ъгъл на контейнера ѝ;
* Margin – това е дебелината на рамката на контролата (от четирите страни);
* Dock – указва дали контролата да бъде „прилепена” към някой от ръбовете на контейнера (заемайки цялата дължина на ръба); възможно е и да запълва цялото свободно пространство в контейнера (DockStyle.Fill);
* Anchor – позволява „закачване” на контролата за един или повече от ръбовете на контейнера, така че при промяна размерите на контейнера, разстоянието между закачените ръбове на контролата и контейнера да остава едно и също;
* Text – това е текста на контролата;
* Enabled – указва дали взаимодействието на потребителя с контролата е разрешено или не;
* Visible – указва дали контролата е видима или не;
* ForeColor – това е цвета на текста;
* BackColor – това е цвета на фона;
* Font – това е шрифта на текста;
* TabIndex – това е поредния номер на контролата в рамките на контейнера ѝ; определя реда на обхождане на контролите в клавиша Tab;
* Tag – това е произволен обект, закачен към контролата.

Всяка контрола предоставя и специфични за нея свойства, както и множество събития, свързани с функционалността ѝ.

## Контейнери

Контейнерите са специални контроли, които могат да съдържат други контроли.

### Panel

Това е най-простият контейнер – служи за невизуално групиране на контроли.

### GroupBox

Този контейнер служи за визуално групиране на контроли.

### FlowLayoutPanel

Този контейнер автоматично подрежда съдържащите се в него контроли една след друга (или една под друга).

### SplitContainer

Този контейнер разделя видимата си област на две части (хоризонтално или вертикално), границата между които може да се премества от потребителя.

### TabControl

Този контейнер предоставя възможност за странициране на видимата си област.

### TableLayoutPanel

Това е контейнер, който позволява дефинирането на редове и колони с абсолютни или относителни височини/ширини. За всяка от съдържащите се в него контроли се указва клетка, в която контролата да бъде разположена. В допълнение може да се посочи дали контролата да заема повече от един ред или колона.

## Събития

Обработчици на събития се създават и закачат лесно през визуалния редактор на формата. Всяко събитие предоставя информация за контролата или компонентата, която е изпратила събитието, както и специфични за събитието параметри.

## Модални диалози

Модалните диалози са просто форми, който са показани модално с метода ShowDialog() вместо Show(). Всяка форма има свойство DialogResult, което по подразбиране има стойност DialogResult.None. Когато тази стойност бъде променена, диалогът автоматично се затваря и стойността се предава като резултат на ShowDialog() метода.

Бутоните също имат свойство DialogResult. Ако стойността му е различна от DialogResult.None, след натискане на бутона свойството на формата автоматично се приравнява на свойството на бутона, затваряйки модалния диалог.

## Data Binding

При попълване на списъчна контрола (ListBox и ComboBox) може да се използва нейното свойство DataSource, което приема стойности от тип IList или IListSource. Контролата автоматично добавя всички елементи на източника в своя списък от елементи. В допълнение могат да се използват свойствата DisplayMember и ValueMember, за да се посочи съответно кое свойство на елемент от източника да се използва за визуализиране на елемента в текстова форма и кое – за генериране на стойност на свойството SelectedValue (по подразбиране се използва целият елемент).

За жалост ListView и TreeView не поддържат data binding, което налага ръчното пълнене на елементите им.

WPF предлага много широки възможности за data binding не само на списъчни контроли, а на почти всички свойства на всички контроли. Докато Windows Forms приложенията са предимно control-driven, приложенията в WPF са предимно data-driven.