

СЪЗДАВАНЕ НА СИСТЕМИ НА БАЗАТА НА КЛИЕНТ–СЪРВЪР ТЕХНОЛОГИИ

СЪДЪРЖАНИЕ

I.Традиционни клиент-сървър системи	3
I.1.Въведение в клиент-сървър технологиите	3
I.1.1.Кратка история	3
I.1.2.Понятие за клиент-сървър	4
I.1.3. Характеристика на клиент/сървър моделът на обработка на данните ...	7
I.2.Традиционна клиент-сървър архитектура.....	12
Двуслойна архитектура	14
Трислойна архитектура	15
I.2.4. Клиент-сървър архитектура на уеб приложения	20
I.2.5.Трислойна архитектура на уеб приложения	21
I.2.6. Сложна (N-слойна) архитектура	23
I.3.Методологически принцип ACID.....	24
I.4. Типове сървъри.....	25
I.4.1 Общи сървъри	25
I.4.2. Файлов сървър	26
I.4.2.Сървъри за транзакции	27
I.4.3 Сървър за бази данни.....	27
I.4.4. Сървъри на приложенията.....	31
I.4.5.Сървъри за съвместна работа (Groupware Servers)	31
I.4.6.Уеб сървър.....	32
I.5.Инструментариум за разработка на Клиент-Сървър системи	33

I.5.1. Разработка на Client-Server приложение във Visual Studio .NET с Visual Basic.Net.....	33
I.5.2.Инструментариум за разработване на уеб базирани клиент-сървър системи	42
I.6.Методика за разработка на настолни клиент-сървър приложения	57
I.6.1. Разработване на компонентите на ниво потребителски интерфейс с <i>Windows Forms</i>	57
II.РАЗПРЕДЕЛЕНИ СИСТЕМИ.....	70
II.1.Дефиниране на разпределена система	70
II.2. Среден слой (Middleware).....	72
II.3 Типове middleware софтуер	74
Използвани литературни източници:	80

I.Традиционни клиент-сървър системи

I.1.Въведение в клиент-сървър технологиите

I.1.1.Кратка история

Информационните технологии са неотменима част от управлението на днешните организации. Клиент-сървър технологиите са част от използваните в една организация информационни технологии.

Организациите знаят как да се възползват от традиционните фирмени технологии за производство и реализация на стоките си и да увеличават стойността чрез използването им. Сега организациите очакват да получават все по-голяма стойност от инвестициите си в областта на информационните технологии.

Голяма част от информационните системи, съществуващи в организациите са създавани не едновременно, а в различни периоди от време, за различни машини и са използвани различни технологии, обменът на данни между тях е силно затруднен. Организациите трудно биха се уеднаквили всички системи, или подменили всички от тях с нови, тъй като това изисква значителни инвестиции. Това, което може да се направи обаче, е да се уеднаквят и стандартизират комуникационните им възможности.

Организациите, които са готови за глобалния пазар и искат успешно да комуникират и да се конкурират с други организации, имат различни технологични възможности на разположение. Конкурентната глобална икономика, особено в условията на криза, ще запрати в небитието тези, които не могат или не желаят да осъществяват комуникация с останалите в условия на конкурентно начало.

Всички организации трябва да търсят начини да извличат повече стойност от инвестициите си в технологии. Усеща се стремеж и желание да преосмислят съществуващите организационни структури и бизнес практики. Наблюдава се също така желание да се подобри програмното осигуряване да се осигури минимализация на вложените средства и увеличаване на ефективността.

Създаването на системи на базата на клиент-сървър технологии е един много ефективен източник на инструменти, които дават повече пълномощия на служителите, натоварват ги с власт, но и с отговорност.

Създаването на работни станции, овластяване на работна група, запазване на съществуващите инвестиции, дистанционно управление на мрежата и задвижвания от пазара бизнес са силите, които създават необходимостта от изграждането на системи на базата на клиент-сървър технологии.

Създаването на системи на базата на клиент-сървър технологии изисква хибридни умения, които включват обработка на транзакции, проектиране на база данни, опит в комуникациите, графичен дизайн на потребителския интерфейс и не на последно място задълбочени познания за начина на функционирането на интернет. Всички приложения изискват познаване на обектите и инфраструктурата на компонентите им.

През годините от възникването на клиент-сървър технологиите до наши дни те непрекъснато се развиват, като нови и нови характеристики се включват в тях. Някои автори разглеждат дори съвремените обработки в облак (cloud computing), като висока степен на развитие на клиент-сървър технологиите.

Исторически първите системи клиент-сървър се появяват още през 80-те години на миналия век. Терминът "клиент-сървър" се използва при свързване на персонални компютри в мрежа.

Моделът клиент-сървър е основата на днешните информационни системи.

Предишният модел на обработка беше основан на централизирани ресурси съсредоточени на едно място. Главният компютър предоставя достъп до бази от данни чрез множество терминали.

Клиент-сървър моделът е по-нататъшно развитие на архитектурата на файловете сървъри в локални мрежи, където системите комуникират с файл сървър, неразполагащ с мощен процесор.

I.1.2.Понятие за клиент-сървър

Понятието клиент-сървър е много широко и се използва с много значения. Софтуерни приложения, хардуер, приложна архитектура, цели информационни системи се означават като клиент-сървър.

Ще разгледаме различните аспекти на използването на понятието, като започнем с клиент-сървър от гледна точка на управлението на мрежата.

Клиент-сървър от гледна точка на управлението на мрежата

Клиентът и сървърът са обобщени логически понятия, които комуникират чрез мрежа, за изпълнение на някаква задача.

Комуникацията чрез мрежа обаче може да се осъществява и от равноправни компютри, когато всеки от тях функционира и като клиент и като сървър. Всеки потребител сам администрира ресурсите на своя компютър. Такъв тип комуникация се нарича „peer to peer” или равноправна комуникация между компютри в мрежа.

Тук вече става въпрос за начин, по който се администрира компютърната мрежа, т.е. когато говорим за клиент-сървър комуникация в мрежа, акцентът **не е** върху архитектурата на приложението и как различните програми се разполагат върху машините. Акцентът е върху начинът на управление на комуникацията в мрежата. При клиент-сървър мрежата комуникацията се управлява от сървъра, а при равноправната мрежа компютрите са равнопоставени.

При „peer to peer комуникацията в мрежа участниците, отдават част от своите ресурси – процесор, дисково пространство и др. на разположение на другите мрежови участници без необходимост от централен сървър. Участниците са едновременно доставчици и потребители на ресурси в контраст на традиционния клиент-сървър модел, където само сървърите дават, а клиентите потребяват информация. Всеки компютър в P2P мрежа може да бъде едновременно и файлов сървър, както и клиент. Единствените изисквания за компютър, за да се присъедини към P2P мрежа, са интернет връзка и P2P софтуер.

Общи софтуерни P2P програми са BearShare, FilesFire, Morpheus и др. Когато някоя от тези програми се свърже с P2P мрежа, като например Gnutella, това позволява на компютъра достъп до хиляди други системи в мрежата.

Веднъж свързан към мрежата, P2P софтуера ви позволява да търсите за файлове на компютрите на други хора. Междувременно, други потребители на мрежата могат да търсят файлове на вашия компютър, но обикновено само в рамките на една папка, която сте определили да споделите.

В P2P мрежите споделянето на файлове е лесно и удобно, но това доведе до софтуерното пиратство, незаконното сваляне на музика и нарушаване на авторските права.

Характеристиките на клиент-сървър модела на комуникация в мрежа могат да се обобщят в следното:

1. Услуги (сървиси). Клиент-сървър е основно връзка между процеси изпълнявани на различни машини. Сървърът е доставчик на услуги. Клиентът е консуматор на услуги. Клиент-сървър мрежата осигурява строго разделяне на функциите, базирана на идеята на услугите (сървисите).
2. Споделяне на ресурси. Сървърът може да обслужва много клиенти и в да осигурява техния достъп до споделени ресурси.
3. Асиметрични протоколи. Съществува връзка много към едно между клиента и сървъра. Клиентът винаги инициира диалога чрез заявки за услуги. Сървърът очаква пасивно заявка от клиента. В някои случаи обаче, клиентът може да направи заявка към обект на сървъра, който изпълнява услуга, изискваща обратна връзка с клиента. Това позволява на сървъра да инициира извикване на клиента.
4. Прозрачност на местоположението. Някои сървърни процеси могат да се изпълняват на същата машина, на която е клиента или на различна машина, намираща се на друго място. Клиент-сървър организираният софтуер обикновено маскира местоположението на сървъра от клиента, чрез пренасочване на извикванията, когато е необходимо. Една програма може да играе ролята на клиент, сървър или и на двете.
5. Смяна на платформите. Идеалният случай на клиент-сървър софтуер е той да е независим от хардуера, операционната система или платформата, на която работи. Трябва да могат да се сменят и нагаждат една към друга платформите на клиента и сървъра.
6. Размяна базирана на съобщения. Клиентите и сървърите са слабо свързани елементи, които комуникират помежду си чрез размяна на съобщения. Съобщението е механизъм за предоставяне на отговор при заявка за услуга.
7. Инкапсулиране на услугите. Съобщението към сървъра съобщава за исканата услуга. Работа на сървъра е да определи как ще се изпълни заявката, в каква последователност ще се изпълнят дейностите и какви програми ще се активират. Сървърът може да бъде препрограмиран, надстройван, обновен, но това не трябва

да засяга клиентите, докато механизма на размяна между тях се извършва чрез съобщения.

8. **Мащабируемост.** Клиент-сървър системите могат да се мащабират хоризонтално или вертикално. Хоризонталното мащабиране означава да се добавят или премахват клиентски работни станции само чрез единично и несложно действие. Вертикалното мащабиране означава или преминаване към по-голям и по-мощен сървър или разпределение на обработката и натоварване на множество сървъри.
9. **Интегритет.** Програмните модули на сървъра и клиента могат да се управляват централно, което означава по-лесно опазване целостта (интегритета) на споделените данни.

Недостатъци на клиент-сървър модела на администриране на мрежата в сравнение с равноправната (P2P) мрежа

Трафикът между клиента и сървъра може да се претовари при увеличаване на броя на заявките от клиента. Може да се достигне до момент, в който сървърът не може да обработи едновременните заявки от клиентите и да блокира.

При равноправната мрежа, пропускателната способност се увеличава, колкото повече възли се добавят. Общата пропускателна способност на равноправната мрежа, може да бъде приблизително изчислена като сумата от честотите на всеки възел в тази мрежа.

На клиент-сървър мрежата липсва стабилността на P2P мрежата. Достатъчно е сървърът да се претовари и състоянието става критично - заявките на клиентите не могат да бъдат изпълнени. В P2P мрежите, ресурсите обикновено са разпределени сред много възли. Даже един или повече възли да бъдат повредени и откажат изтегляне на файл например, останалите възли ще имат данните, необходими за завършване на изтеглянето.

I.1.3. Характеристика на клиент/сървър моделът на обработка на данните

Клиент-сървър моделът на администриране на мрежата дава възможност да се използват най-рентабилно потребителският

интерфейс, съхранението на данни и достъпът до услуги в рамките на организацията.

Клиент-сървър моделът осигурява технологични средства, чрез които предишните инвестиции да са в унисон с текущите технологични възможности. Организациите имат възможност за използване на технологията за предоставяне на бизнес решения. Достъпът до повече услуги от всички и конкуренцията на пазара, допълнително увеличава необходимостта организацията да се възползва от предимствата, които предоставят приложенията изградени на база клиент-сървър модела.

Клиент-сървър обработката на данни се извършва директно на обучен, знаещ потребител, който има способността да действа адекватно при грешки в данните и който има възможността да се възползва от предоставената информация.

Разширено споделяне на данни

Данните, които се събират при един нормален бизнес процес и се обработват на сървъра са незабавно достъпни до всички оторизирани потребители.

Използването на SQL за дефиниране и манипулиране с данните, осигурява поддръжка на отворен достъп от клиентския софтуер.

SQL предоставя на всички оторизирани потребители достъп до информация чрез средство, което е в съответствие с техните бизнес нужди. Едни и същи данни се предлагат с една и съща стойност на всички потребители, като това се осигурява от прозрачни мрежови услуги.

Интегрирани услуги

В клиент-сървър модела цялата информация която използва клиента, му се предоставя непосредствено на неговия компютър.

Потребителят може да използва всички допълнителни програмни средства, които съдържа неговата десктоп система, за да представи получената информация.

Споделяне на ресурси чрез общи устройства

Клиент-сървър моделът предоставя възможности за постигане на истински отворена изчислителна система. Могат да се създават приложения без оглед на хардуерните платформи или техническите характеристики на софтуера.

Потребителите могат да получат клиентски услуги и прозрачен достъп до услуги, предоставяни от бази данни, комуникации и сървъри на приложения.

Операционната система и хардуерната платформа са независими от приложението и маскирани от инструментите за разработка, които се използват за изграждане на приложението.

Потребителят създава и стартира "събития" (event), които управляват бизнес приложенията, обработващи бизнес процесите.

Взаимозаменяемост и съвместимост на данните

SQL е стандартен език за дефиниране и достъп до данни. Той се поддържа за много доставчици. Почти всички инструменти, които се използват за разработка на клиент-сървър приложения, очакват позоваване на сървър за база данни, достъпна чрез SQL.

Мрежовите услуги осигуряват прозрачна връзка между клиента и локалния или отдалечения сървър. С някои допълнителни приложения за бази данни, даден потребител може да състави консолидиран изглед от данни, които всъщност се обработват под хетерогенни платформи.

Разработчиците вече не са принудени да използват само една платформа за поддържане на данни. Клиент-сървър моделът дава възможност да се правят голям обем актуализации на различни места и въпреки това да се запази целостта на данните.

Системи, разработени с използването на SQL, по своята същност са прозрачни по отношение съхранението на данни, тяхното местоположение и технологична платформа.

Тази прозрачност позволява на данни от базата и таблици да бъдат премествани, копирани без значение местоположението им или софтуерната платформа. Тази функция е особено ценна при възприемането на нови технологии, или ако бизнес логиката изисква данните да се преместят по-близо до техния собственик.

Независимост между разположението на данните и тяхната обработка

При предишните системи, базирани на големи машини с терминална връзка, потребителят трябваше да научава и борави с голям брой команди, функционални клавиши, съобщения за грешки, методи за навигация, начини за обработка, средства за сигурност и пр., за да може да получи достъп. Синтаксисът за достъп до всяка система беше уникален, в зависимост от използваната софтуерна платформа.

Потребителят днес е поставен в центъра на съвременните информационни системи. Той очаква всичко да му бъде лесно, да се свързва със системата и получава необходимите му данни, без значение къде се намират те и как се обработват.

Независимост от промяна (инкапсулиране)

Клиент-сървър архитектурата дава възможност ролите и отговорностите на изчислителна система да бъдат разпределени между няколко независими компютри, които са познати помежду си само чрез мрежата. Това създава допълнително предимство на тази архитектура: по-голяма лекота на поддръжката. Възможно да се замени, ремонтира, или дори да се премести сървър, без клиентите да разберат, без да бъдат засегнати от тази промяна. Тази независимост от промяна се посочва като инкапсулиране.

Сигурност на данните

При клиент-сървър моделът на обработка, данните се съхраняват на сървъри, които имат по-голям контрол на сигурността от клиентите. Сървърите могат да осигурят достъп до своите ресурси само на тези клиенти, които имат съответстващи разрешения за добавяне или промяна на данните. Данните се съхраняват централизирано и това дава възможност за едно управление и контрол върху тях.

Съществуват и са на разположение много клиент-сървър технологии, които са предназначени за гарантиране на сигурността.

Разработени са също така и много средства за "олекотяване" на потребителския интерфейс при използването му, но запазване на сигурността на данните.

1.1.4. Клиент-сървър като завършена компютърна система

Днес много често терминът клиент-сървър се използва за обозначаване на завършени компютърни системи, които включват хардуер и софтуер в едно цяло.

Завършената софтуерна/хардуерна система има от една страна сървър, представляващ софтуерна услуга, стартираща се на специално предназначен за нея компютър. Това са например сървър за база данни, файлов сървър, пощенски сървър, печатен сървър. От друга страна тази система има клиент, специализирана програма, осъществяваща заявки за услуги към сървъра.

Понятието се използва и за посочване на цялостна трислойна архитектура на информационна система, част от интернет пространството, която включва няколко слоя вложени един в друг, всеки имащ отделна трислойна структура.

Трислойната архитектура включва клиентски слой, среден слой и слоят на информационната система (Enterprise Information System). Информационната система от своя страна, може да се състои от презентационен слой (потребителски интерфейс), слой на бизнес логиката и съхранение на данните.

Клиентският слой е уеб базиран графичен потребителски интерфейс за взаимодействие с клиентите, средния слой е комбинация от уеб контейнери и сървлети. Слойт на информационната система съдържа бизнес логиката и системите за управление на бази данни, необходими за приложенията на информационната система. Базата данни е достъпна чрез корпоративния софтуер за връзка, уеб компонентите или компонентите на клиентското приложение.

I.2.Традиционна клиент-сървър архитектура

Когато говорим за клиент-сървър технологии, трябва да се има предвид, че това са всички техники и средства които се използват на базата на клиент-сървър архитектурата.

Преди да дефинираме какво е клиент-сървър архитектура, трябва да поясним доста понятия, като започнем с понятието архитектура.

Дефиницията за архитектура залегнала в стандарта ANSI/IEEE Std 1471-2000 “IEEE Recommended Practice for Architectural Description of Software-Intensive Systems” [1] е:

Архитектурата е фундаменталната организация на една система, нейните компоненти, отношенията между тях и обкръжаващата среда, и принципите на управление и развитие.

За да се опише архитектурата на дадена система, трябва да се открият нейните базови компоненти, да се посочат връзките помежду им и с обкръжаващата среда, да се обвържат с целите и задачите на разглежданата система. Архитектурата се представя като най-общо описание на структурата на една система и начинът, по който тя функционира. Описанията на архитектурата, представени в графичен, словесен или смесен вид, се наричат *модели* на архитектурата.

Приложният софтуер включва програмите, които извършват обработка на информация за целите на дадена организация.

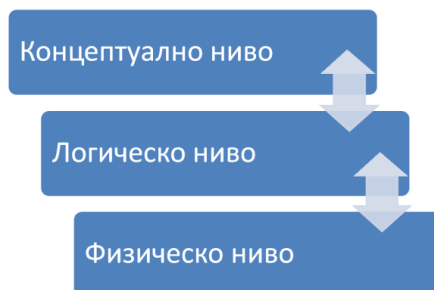
Приложната архитектура представя приложните програми, използваните бази данни и връзките между тях.

Описанието на приложната архитектура трябва да покаже как множество приложни програми работят заедно в рамките на една информационна система.

Този тип архитектура включва:

- Описание на автоматизираните услуги и бизнес процеси.
- Описание на интерфейса и взаимодействието между различните приложни програми.
- План за разработка на нови приложения и ревизиране на стари програмни продукти, базиран на целите, задачите и използваните в организацията технологии.

За пълното описание на приложната архитектура се използват трите нива на абстракция и обобщение – концептуално, логическо и физическо (фиг. 2).



фигура 1. Нива на детайлизация и абстракция при описание на приложната архитектура

Концептуалното ниво на приложната архитектура се използва да се дефинират изискванията от гледната точка на потребителите за използваните приложения. На това ниво не се засягат технологии и използвани технически средства.

Логическото ниво на приложната архитектура включва описание на управлението на данните, необходимата обработка на данните, и взаимодействието между отделните части на програмните системи.

Физическото ниво на приложната архитектура свързва логическия модел с физическото разположение върху устройства на отделните елементи. Всеки елемент на физическото ниво обикновено е описан в процеса на проектиране като хардуерен или софтуерен компонент, с техническите му характеристики и изисквания.

Почти всяка информационна система включва следните три вида обобщени функции:

- **Потребителски интерфейс** (презентационни функции) - функции реализиращи взаимодействието на информационната система с потребителя – въвеждане на данни и извеждане на информация.
- **Бизнес логика** - функции по обработка на информацията в съответствие със специфичните правила, характерни за съответната приложна област. Описанието на бизнес логиката включва описания на бизнес правилата и бизнес процесите. Бизнес правилата – това са операциите, дефинициите и ограничителните условия, които се прилагат от организацията при постигането на целите ѝ.
- **Съхранение и извличане на данни** от бази данни или от отделни файлове.

Когато функциите на информационната система са обособени в програмен компонент така, че могат да бъдат инсталирани и

изпълнявани самостоятелно на отделна машина, те формират слой [tier] на информационната система.

Слой на информационната система – това са функции на информационната система, които са обособени в програмен компонент така, че могат да бъдат инсталирани и изпълнявани самостоятелно на отделна машина.

Класификацията на приложните архитектури е в зависимост от това, в каква степен трите вида функции – презентационните, бизнес логиката и съхранението на данните - са обособени в слоеве.

Според броя на слоевете архитектурата на информационната система е еднослойна, двуслойна, трислойна, N-слойна (многослойна).

При еднослойната архитектура трите вида функции са свързани неразделно.

Еднослойната архитектура е такава архитектура, при която трите вида функции – презентационни, бизнес логика и съхранение на данни, са силно свързани и приложението задължително се изпълнява върху един компютър.

Двуслойна архитектура

При разделянето на функциите на информационната система две части – клиент и сървър се говори за двуслойна архитектура.

Двуслойната архитектура (two-tier) се характеризира с обособяване на два слоя:

- Клиентски слой – който обхваща потребителския интерфейс и бизнес логиката.

- Сървърен слой – който реализира всички функции по работа с базите данни.

Информационните системи с двуслойна архитектура изискват използване на самостоятелни системи за управление на бази данни (СУБД) като Oracle, IBM DB2, Sybase, Microsoft SQL Server.

Приложението се намира върху работната станция (персонален или мрежов компютър).

Основното предимство на двуслойната клиент-сървър архитектура е разделянето на функционалността между двата слоя. Избягва прекомерното натоварване на сървъра при обслужване на множество заявки. Основната част от обработката не се извършва от сървъра, а от клиентския слой. Разделянето на клиент и сървър позволява много потребители да бъдат обслужвани от сървъра

едновременно. Друго голямо предимство на двуслойната архитектура е намаляването на мрежовия трафик, тъй като бизнес логиката е при клиента и не е нужна непрекъснатата комуникация със сървъра за извършването на обработката. В някои случаи това може да се окаже обаче недостатък, защото натоварването на клиента с изпълнението на бизнес логика не винаги е добра практика.

Като недостатък на двуслойните приложения може да посочи сложната поддръжка на приложенията. При промяна в логиката на приложението се налага преинсталация на клиентската част, а при голям брой потребители това би означавало да се прави преинсталация на приложението на голям брой компютри, което може да доведе до значителни трудности и големи разходи. По-големият проблем на тази архитектура обаче е невъзможността на един сървър да обслужва едновременно прекалено много потребители, а това води до лоша скалируемост. За този проблем решение може да бъде кластеризацията на сървъра, но при големи системи обикновено се преминава към архитектура с повече слоеве.

Трислойна архитектура

При обособяването на трите вида функционалност в самостоятелни слоеве се говори за трислойна архитектура.

Трислойната архитектура (three-tier) се характеризира с обособяване на потребителския интерфейс, бизнес логиката и управлението на данните в самостоятелни слоеве.

В трислойната архитектура се разграничават:

- Слой на потребителския интерфейс (user interface tier), наричан също слой на представянето (presentation tier).
- Слой на бизнес логиката (business logic tier), наричан също слой на приложението (application tier).
- Слой на управление на данните (data management tier).

Всеки един от трите слоя може да бъде променен или реализиран с различни програмни средства, без това да повлияе на останалите два слоя, което придава гъвкавост и по-лесна поддръжка на самото приложение.

Логическото обособяване на бизнес логиката позволява нейното имплементиране на отделен компютър, но това не винаги е необходимо. Когато трите слоя се инсталират върху два компютъра, се говори за **трислойна клиент-сървър архитектура**.

Трябва да се отбележи най-напред, че когато се говори за “клиент” и “сървър”, се разбира логически обобщени понятия, които включват множество характеристики. Това са обикновено програмни модули, но така могат да се наричат понякога и самите компютри.

Терминът "клиент-сървър" най-често описва взаимоотношенията между две компютърни програми, от които едната програма – **клиент, прави заявка за услуга към другата програма - сървър, който изпълнява заявката.** Понятието за клиент-сървър може да се използва по отношение на програми на един компютър, но по-важна е идеята клиент-сървър, реализирана в мрежа.

В една мрежа моделът клиент-сървър предоставя удобен начин за свързване на програми, разположени на различни места.

Клиентът е една работна станция, с която обикновено работи един потребител. Клиентът управлява и извършва по-голямата част от обработките, с помощта на едно или повече приложения. Клиентът прави заявки за услуги към сървъра и изчаква отговор.

Сървърът е многопотребителска машина с няколко микропроцесора или с микропроцесори с много ядра, голям обем оперативна, дискова памет и изчислителна мощ. Сървърът чака заявки за услуги от клиента, може да работи с много клиенти едновременно, като при получаване на заявките ги обработва и отговаря.

Характеристики на клиента:

- подава заявки;
- изчаква отговор;
- свързва се до малък брой сървъри едновременно;
- взаимодейства с крайните потребители чрез графичен интерфейс.

Характеристики на сървъра:

- пасивност;
- чака заявки от клиенти;
- при получаване на заявки ги обработва и след това отговаря;
- получава заявки от голям брой клиенти;
- не контактува директно с крайния потребител.

Прилагането на клиент/сървър архитектура на приложенията в една организация дава възможност за подходящо, отговарящо на нуждите ѝ разпределение на обработка на данни между клиента и сървъра.

Механизмът, по който взаимодействат клиента и сървъра се нарича „комуникация между процеси“ InterProcess communication (IPC) и се състои от съобщения, които разменят двата компютъра.

Клиент-сървър архитектурата е в основата на общ, технологичен модел на обработка, който е независим от различните платформи. Потребителят изисква функционалността, която клиент-сървър архитектурният модел предоставя, а компютърна платформа осигурява достъп до тази функционалност. Промените в платформата и свързаните с нея технологии остават прозрачни за потребителя. Системи, изградени с прозрачност на технологията, предлагат най-висока вероятност на възвръщаемост на технологичните инвестиции.

Най-общо структурата на клиент-сървър архитектурата може да се илюстрира както е посочено на фиг. 3. Тази илюстрация ще бъде многократно допълвана и детайлизирана по-нататък.



фигура 2. Най-обща илюстрация на клиент/сървър архитектура

Разграничават се два варианта на трислойна клиент-сървър архитектура: с "дебел" клиент и с "тънък" клиент, в зависимост от това доколко е натоварен с функции и мощност клиентският компютър.

Трислойна клиент/ сървър архитектура с „тънък“ клиент

При трислойна клиент/ сървър архитектура с „тънък“ клиент, трите вида функции са обособени, но бизнес логиката и слойът на данните се намират върху общ компютър .

При трислойната клиент/сървър архитектура с „тънък“ клиент на всяка клиентската работна станция е инсталиран потребителския интерфейс, а на сървърния компютър са инсталирани бизнес логиката и СУБД.

Типичната архитектура с „тънък“ клиент, клиентът разполага с ограничена изчислителна мощност (процесор и памет), мрежова карта и обичайните входно изходни устройства (монитор, клавиатура, мишка или др.). Основната изчислителна мощност и дискова памет се намират на сървъра, където се изпълняват всички потребителски програми.

Главната идея на архитектурата „тънък“ клиент, е да се намалят разходите по компютърната техника и нейната поддръжка. Почти цялата администрация се извършва на сървъра. Ако има нужда от засилване на процесорната мощност или увеличаване на дисковото пространство, това се извършва само на сървъра.

Този модел на архитектурата оставя повече функции на сървъра и затова се използва за критични приложения, тъй като заявките се управляват по-лесно, когато по-голямата част от работата се извършва на сървъра. При тази архитектура мрежовият обмен може да се сведе до минимум.

Трислойна клиент-сървър архитектура с „дебел“ клиент

При трислойна клиент-сървър архитектура с „дебел“ клиент, трите вида функции са ясно разграничени, като на клиентските работни станции е инсталиран потребителския интерфейс и бизнес логиката.

При трислойната клиент-сървър архитектура с „дебел“ клиент на клиентската работна станция са реализирани потребителския интерфейс и бизнес логиката, а на сървърния компютър е реализиран само слойът на данните, чрез определена СУБД.

В този случай клиентската работна станция трябва да разполага с процесорна мощ, голямо дисково пространство и възможност за администриране на файловете, т.е. този тип архитектура е по-скъпа от гледна точка на цената и поддръжката.

Този модел на архитектурата оставя повече функции на клиента. Той е най-традиционна форма на клиент-сървър системи.

В реална работна среда, в едно приложение могат да съществуват едновременно както архитектура с „дебел“ клиент, така и с „тънък“ клиент.

Отделяне на бизнес логиката. Сървър на приложенията

Когато програмните модули, реализиращи нивото на бизнес логиката се инсталират върху отделен компютър, се използва сървър на приложенията (Application server).

Терминът сървър на приложенията се използва двузначно. „Сървър на приложенията“ означава:

1. *Специализирана програмна среда*, която обслужва модулите с бизнес логиката. Тази среда осъществява, от една страна, връзки между отделните модули с бизнес логика, а от друга страна, осъществява връзките на бизнес модулите със слоя на представянето (потребителския интерфейс) и със слоя на данните (СУБД).

2. *Компютър*, на който са инсталирани модулите с бизнес логиката и обслужващата ги специализирана програмна среда.

При необходимост, за да се различат двете значения на сървър на приложенията, се използват термините: „софтуерен сървър на приложенията“ и „компютър - сървър на приложенията“.

Сървърът на приложенията обслужва бизнес логиката. Бизнес логиката е мост между потребителя и данните. Този слой, въз основа на запитванията от потребителя, извлича нужната информация от базата данни, обработва я по специфичните за приложната област правила и я подава за извеждане към слоя на потребителския интерфейс. Това се извършва чрез прилагане на формални процедури и бизнес правила към извлечените релевантни данни.

На това ниво могат да се разграничат две части. Първата част включва относително стабилните бизнес правила за съответната приложна област, а втората включва множеството от често променяни процедури на обработка свързани с базите данни.

Втората част може да се инкапсулира в компоненти, които са физически отделени от модулите реализиращи основните бизнес правила.

На сървъра на приложенията могат да се обособят две поднива – на бизнес правилата (Business Rules Layer) и на достъпът до данните (Data Access Layer), които обслужват съответно потребителския интерфейс и слоя на данните.

Разработването на уеб приложения (информационни системи работещи в Интернет или Интранет) доведе до по-нататъшно развитие на логическите архитектури. Появи се необходимостта от създаване на системи с четирислойни и многослойни архитектури.

I.2.4. Клиент-сървър архитектура на уеб приложения

Интернет има някои специфични характеристики, които от своя страна водят до особености и в архитектурата на клиент-сървър уеб приложенията.

Интернет трябва да поддържа връзка между хетерогенни мрежи; да е независим от времето и разстоянията в комуникационната среда; да позволява достъп до информацията и приложенията на различни сървъри чрез единствен клиент; да предоставя възможност за свързване чрез компютърна мрежа с помощта на не много скъпо струващо устройство; клиентите трябва да имат достъп до мрежата без значение компютърната си платформа; информацията, която ще се разполага на сървърите да е достъпна посредством унифициран стандарт, независим от платформата, на която работи сървъра.

Тези специфики на интернет, придават няколко особености на клиент-сървър архитектурата на уеб приложенията.

Първо, тя включва уеб сървър, каквито са например Microsoft Internet Information Service (IIS) и Apache.

Второ, потребителският интерфейс на клиентската работна станция се реализира чрез стандартен браузер, като Microsoft Internet Explorer (IE), Google Chrome, Mozilla FireFox, Opera и др.

Трето, функционалните ограничения на браузерите налагат разделянето на слоя на потребителския интерфейс на два подслоя:

- Подслой на визуализацията на потребителския интерфейс (GUI - graphic user interface visualization) реализиран от браузера. Този подслой, освен визуализацията на потребителския интерфейс, включва минимална обработка, написана на скриптов език изпълним от браузера, като JavaScript.
- Подслой на управлението на потребителския интерфейс (GUI management), реализиран на уеб сървъра заедно с интерпретираща среда за съответния език, на който е разработен модула за потребителския интерфейс, като ASP (Active Service Pages), JSP (Java Service Pages), PHP.

1.2.5.Трислойна архитектура на уеб приложения

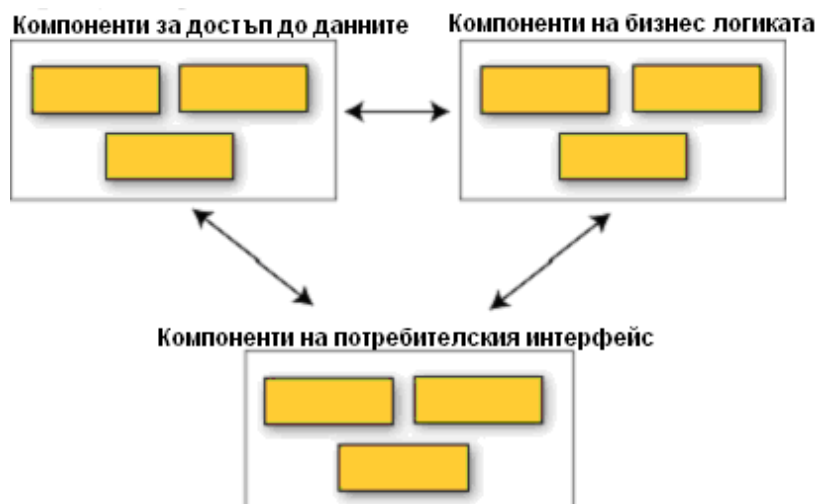
При по-прости уеб приложения, управлението на потребителския интерфейс (ПИ) не се разграничава строго от бизнес-логиката и се извършва от уеб сървър.

За да се създаде уеб базирано трислойно клиент-сървър приложение са необходими много знания и умения от страна на разработващия приложението. За да се изгради системния вход/изход например, трябва да са налице знания за такива средства за разработка, които програмират браузера, за да извърши работите по форматиране на изхода, въвеждане на данни, проверка за валидност и др. Необходимо е познаване на XHTML (HTML), CSS, DHTML и XML за представяне на структури от данни. Основният език за програмиране на браузера е JavaScript, който се използва, за да се манипулира HTML и структурите от данни.

По отношение на обработката на данни от страната на уеб сървър трябва да се програмират приложения, които ще извършат основните обработки в системата. За програмиране на сървърното приложение са необходими знания за сървърните езици за програмиране VB, C#, Java, PHP и др. От една страна, това са пълнофункционални обектно-ориентирани езици, които съдържат всички средства за програмиране на бизнес логиката и от друга страна, те могат да се извикват от вградени сървърни компоненти и да описват отделни функции на системата.

За управлението на базата данни се използват езици за програмиране като SQL (Structured Query Language) за съхранение, управление, редактиране и достъп до данните. Езиците за програмиране на базата данни се използват за създаване на командни процедури и функции за извличане, актуализиране и отпечатване на данните. Сървърните езици извикват тези SQL команди и съхранени процедури, за да изпълнят обработките в базата данни, спестявайки на разработващия системата да пише програмен код на ниско ниво. ADO.NET компонентите осигуряват интерфейса между приложенията и системата за управление на базата данни (СУБД).

Отделните компоненти на трите слоя (фиг. 9) трябва да работят строго координирано, за да се генерира и изведе съдържанието на уеб страницата върху компютърния екран.



фигура 3. Координация между отделните компоненти на трислойната архитектура

Няколко основни предимства произтичат от разделянето на програмния код в различни слоеве:

Първо, при този подход отделни личности и групи могат да разработват самостоятелно всеки от слоевете. Дизайнерът на страницата може да работи върху дизайна и външния вид на потребителския интерфейс, програмистът може да програмира бизнес логиката и специалистът по бази данни може да изгражда достъпа до базата, без тези три дейности да се изчакват.

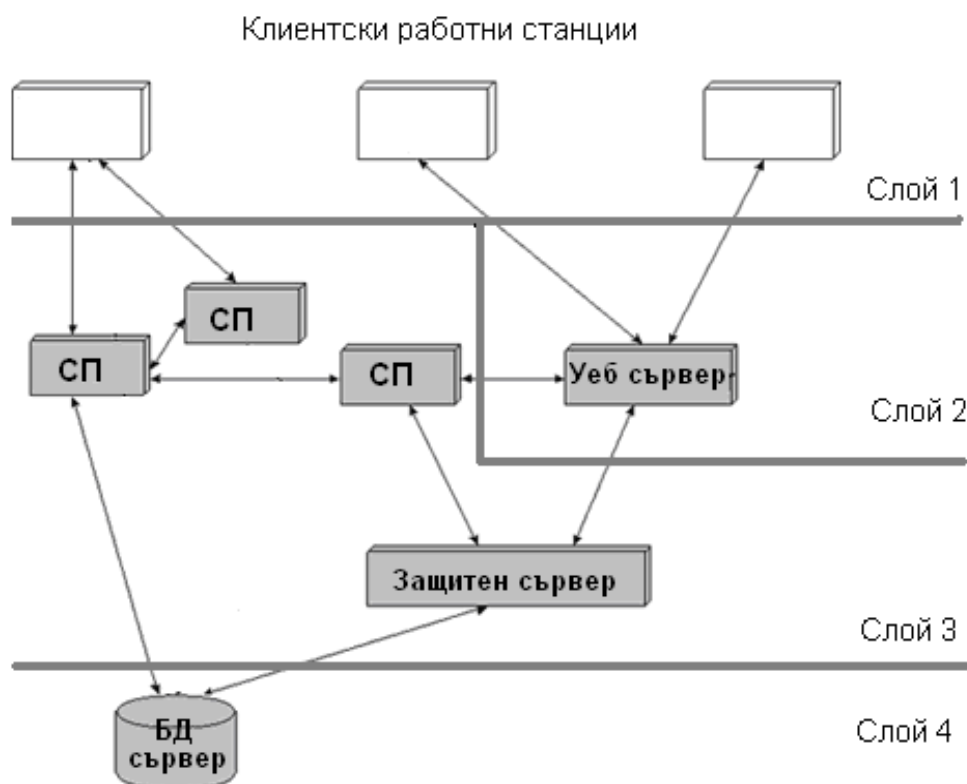
Второ, много по-лесно се поддържат такива уеб приложения. Ако например има промяна в дизайна на базата данни или местоположението ѝ, то тези промени трябва да се отразят само в компонентите за достъп до базата, а не в самата уеб страница като цяло. Компонентите са "черни кутии", които крият вътре в себе си изпълнението на детайлите. Уеб страниците извикват тези компоненти да извършат своята работа и не се интересуват от това какви детайли са променени вътре в тези компоненти.

Трето, компонентите могат да бъдат споделяни с други приложения. Техниките и методите на съвместната логика и съвместният достъп до данните, могат да бъдат използвани от всички страници, които се нуждаят от тях, без да е нужно да се пише повторение на същия код. Създава се на практика отделна библиотека от класове за обработка на уеб страниците, които могат да бъдат извикани от други страници и приложения, без да се знаят техните конкретни детайли, а само крайните резултати от изпълнението.

1.2.6. Сложна (N-слойна) архитектура

При съвременните корпоративни информационни системи, бизнес логиката често е реализирана в множество относително независими модули, които работят върху отделни компютри – сървъри на приложения, обменящи помежду си данни. На фиг. 10 е представена такава сложна архитектура.

Архитектурата на система, включваща няколко сървъри на приложения, се нарича сложна или N-слойна (N-tier), тъй като едно обръщение на потребителя се обработва от голям брой модули. От теоретична гледна точка обаче, тази архитектура се класифицира като четирислойна, при която слойът на бизнес логиката е разпределен върху няколко компютъра.



фигура 4. Сложна (N-слойна архитектура) с няколко сървъра на приложения (СП).

Най-важната особеност на системите с N-слойна архитектура е, че бизнес логиката е разделена на множество групи от бизнес процеси, като всеки от тях е реализиран в обособен модул. Това обуславя следните положителни характеристики на приложенията:

1. Модифицируемост – дава се възможност за промяна функционалността на някой от модулите, без това да влияе на другите модули.

2. Разширяемост – възможност на добавяне на нови функции, чрез включване на нови модули с бизнес логика.

3. Многократна използваемост – възможност за изграждане на нови приложения чрез интегриране на съществуващи бизнес-компоненти.

I.3.Методологически принцип ACID

Накрая, за да обобщим общовалидните характеристики на традиционната трислойна архитектура, ще използваме методологическия принцип, известен като ACID (Atomicity, Concurrency, Independence, Durability), който е в сила при системите изградени на транзакции.

- Atomicity – всяка транзакция се изпълнява до завършването ѝ, или не се изпълнява въобще. Това важи и в случаите, когато като транзакция се обозначава операция с много подоперации, но всички се изпълняват според принципа „до края или нищо“. Ако някоя от подоперациите не завърши успешно, тогава може да се върне обратно частично свършената операция до там, където всичко е успешно.

- Concurrency – транзакциите се изпълняват така, че да бъдат максимално близо до едновременното им изпълнение.

- Independence – транзакциите са създадени така, че да се изпълняват независимо една от друга. Всяка клиентска програма е написана така, че да се изпълнява така, все едно останалата част от системата бездейства. Сървърът за базата данни предпазва от едновременното изпълнение на транзакциите. Този принцип може да се нарече също така „изолация“.

- Durability – Резултатите от завършената транзакция са непроменливи.

Характерно за този принцип е, че спазването му гарантира изолиране на едновременните транзакции към базата данни. Ако две транзакции се изпълняват едновременно от две клиентски програми върху едни и същи данни, то този подход гарантира, че винаги едната транзакция ще се изпълни изцяло след другата и няма да има взаимодействие помежду им.

Тези свойства са описани за първи път от Джим Грей (Jim Gray) през 1981 година, а през 1983 г. Андреас Ройтер (Andreas Reuter) и Тео Хардер (Theo Härder) въведоха акронима ACID, за да ги опишат в едно.

I.4. Типове сървъри

I.4.1 Общи сървъри

Съществуват много различни типове сървъри. Реално колкото обработки могат да се направят и дейности да се извършат в мрежа, толкова типове сървъри съществуват. Има сървъри за електронна поща, за защита на данни, за местоположение и пр.



фигура 5. Видове общи сървъри

POP3, SMTP, IMAP пощенски сървъри

Сървърите, работещи по версия 3 на протокола POP3 (Post Office Protocol), съхраняват входящите имейли до момента, когато проверите пощата си, и тогава ги изпращат във вашия компютър. POP3 е най-често използваният тип акаунт за лични имейли. Съобщенията обикновено се изтриват от сървъра, когато проверите пощата си.

Сървърите, работещи по протокола за достъп до имейл чрез Интернет (IMAP - Internet Message Access Protocol) ви позволяват да работите с имейли, без първо да ги изтеглите във вашия компютър. Можете да визуализирате, изтривате и организирате съобщенията директно в имейл сървъра, и те се съхраняват на сървъра, докато решите да ги изтриете. IMAP обикновено се използва за бизнес имейл акаунти.

Сървърите, работещи по протокола за обикновено предаване на електронни съобщения (SMTP), управляват изпращането на вашите имейли до Интернет. SMTP сървърът управлява изходящите имейли и се използва в съчетание с входящ POP3 или IMAP имейл сървър.

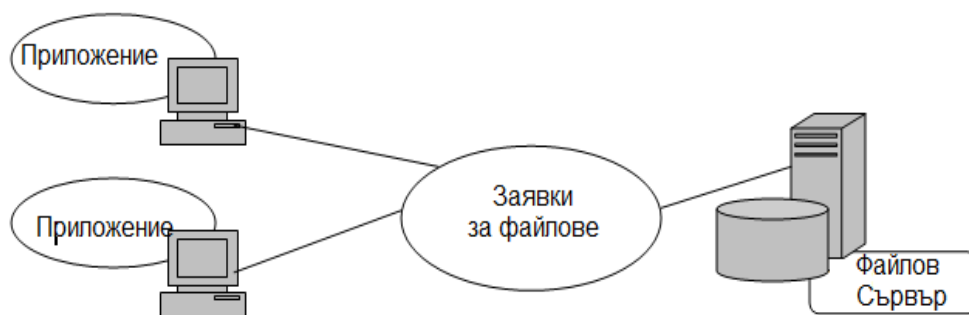
По-нататък ще разгледаме част от типовете сървъри, които имат директна връзка с клиент-сървър архитектурата на информационна система.

1.4.2. Файлов сървър

Това е най-първичният начин за осигуряване на услуги, чрез обмен на съобщения през мрежата, за да се намерят исканите файлове.

Файловият сървър е компютър, обикновено част от LAN мрежа, който има за цел да предостави общо дисково пространство за споделяне и съхранение на файлове - документи, снимки, филми, бази данни и пр. Достъпът до файловете си осъществява от работните станции, свързани към компютърната мрежа. Файловият сървър не е предназначен за извършване на изчислителни задачи. Той не извършва обработки. Предназначен е преди всичко за съхранение и извличане на данни, а обработките се извършват от работните станции.

Клиентът прави заявка за файлове през мрежата към файловия сървър. В типичния файлов сървър, софтуерът, споделените данни, базите данни и резервните копия се съхраняват на диск, или други запаметяващи устройства за съхранение на данни, които се управляват от файловия сървър. На файловия сървър може да се инсталира повече от един диск, като тогава обикновено се използва технологията RAID, при която не само се ускорява достъпът до данните, но при повреда на някой от дисковете, няма да си загубят данни, а само ще се забави скоростта.



фигура 6. Клиент-сървър с файлов сървър

Интернет файлови сървъри

Файловите сървъри в Интернет се категоризират по метода на достъп:

- Файлови сървъри достъпни чрез File Transfer Protocol
- Файлови сървъри достъпни през HTTP, но различни от уеб сървъри, които обикновено предоставят динамично уеб съдържание в допълнение към статични файлове.
- Сървъри на LAN.

Много файлови сървъри едновременно принт сървъри, тъй като те осигуряват достъп до принтери чрез мрежа.

В някои случаи до един файлов сървър може да се осъществява достъп с различни цели – той може да работи като FTP сървър, сървър за малки и средни фирми и т.н., обслужващи един и същи файлове.

Типичен пример за фирмен файлов сървър е **Windows Small Business Server** на **Microsoft**.

Windows Small Business Server на Microsoft.

Това е файлов сървър, разработен от **Microsoft**.

I.4.2.Сървъри за транзакции

The Microsoft Transaction Server (MTS) – сървър за транзакции на Майкрософт. За повече информация:

<http://searchsoa.techtarget.com/definition/Transaction-Server>

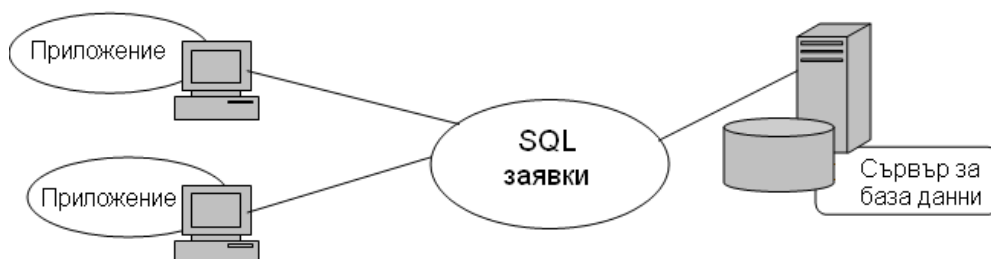
I.4.3 Сървър за бази данни

Сървърите за бази данни играят ключова роля в съхранението на данни.

Сървърът за база данни, изпълнява няколко основни задачи в клиент-сървър архитектурата, освен съхранението – това са анализ, манипулиране, архивиране на данни, както и някои други специфични дейности.

В една клиент-сървър архитектура обикновено приложението, което обработва SQL заявките и данните, се намират на една и съща машина. Сървърът използва своята процесорна мощ да направи необходимата извадка от записи и да ги върне обратно на клиента.

Приложението с потребителския интерфейс се намира на клиентския компютър, така че трябва да се пише код и за клиента и за сървъра.



фигура 7. Клиент-сървър архитектура със сървър за база данни

Има няколко основни вида сървъри на бази данни:

- Плоски (flat) файлови сървъри на база данни
- Сървъри на релационни бази данни
- Обектни (Object) сървъри на база данни
- Сървъри на обектно-релационни бази данни

Файлови сървъри на бази данни

Най-известният представител на файловите сървъри на бази данни е **FileMaker** на FileMaker Inc., която е част от Apple Inc.

Плоските файлове са текстови файлове, или двоични и текстови файлове, които съдържат информация, която може да бъде отграничена с разделители като запетая, двоеточие или точка и запетая.

Плоските файлове се използват не само като инструменти за съхранение на данни, но и като инструменти за трансфер на данни към отдалечените сървъри, това са т.нар. информационни потоци. През последните години обаче обменът на данни между различни платформи беше заменен от XML файловете, които съдържат не само данни, но и описание на данните. Плоските файлове за прехвърляне на данни се използват при големите компютри (мейнфрейм).

Определяща характеристика на FileMaker е, че машината на базата данни е интегрирана с формите, екраните, справките, използвани за достъп до нея. В тази връзка, тя е по-близо до настолните системи за бази данни като Microsoft Access. При големите релационни системи за

управление на бази данни (RDBMS) задачите им са свързани основно с организацията, съхранението и обработването на данните и са отделени от потребителския интерфейс. До версия 7 на FileMaker всяка таблица се съхраняваше като отделен файл с релационни връзки към другите файлове и всеки файл имащи собствен вграден интерфейс. Версия 7 (от 2004 г., към момента има и версия 11), въведе нов файлов формат, който поддържа размер на файловете до 8 терабайта. Отделните полета в един запис могат да съдържат двоични данни до 4 гигабайта или 2 гигабайта от 2-байтав Unicode текст. Релационният модел на FileMaker се обогатява, като предлага възможност за съхранение на множество таблици в един файл и графичен редактор, които показва и допуска манипулиране на свързани таблици.

Термините, използвани за описание на различните аспекти на базата данни и инструментариума във **FileMaker** се различават, например от MySQL, но понятията остават същите.

Сървъри на релационни бази данни

Най-голям дял имат сървърите на релационни бази данни и съответно системите за управление на релационни бази данни (RDBMS), тъй самите релационни бази данни най-често се използват като средство за описание и съхранение на данни.

Обектните бази данни, съхраняват на данни в обекти, а не в таблици. В базата данни се съхранява информация за класа, атрибутите и методите. Обектни (Object) сървъри на база данни използват Object Query Language (OQL) като стандартен език за комуникация.



Обектно-релационните бази данни са релационни бази данни, данните се съхраняват в таблици, но включват и част, която позволява използването на сложните типове описания на обекти. Ако релационната бази данни ще се използва за съхраняване на обекти, то обектът първо се декомпозира на части, всяка част се поставя в таблици. После обратно за да се използва обекта, трябва да бъде отново сглобен от частите, съхранени в таблици.

Съществуват различни фирмени сървъри за бази данни: Microsoft SQL Server, Oracle, DB2 и др.

Общото между тези сървъри (с много малки видоизменения) е езикът за заявки SQL query language.

Таблица. 1. Описание на фирмени сървъри за бази данни

Общоприет знак	Описание
	<p>Microsoft SQL Server е сървърна система, работеща върху операционната система Windows и предназначена в началото за малък и среден бизнес и едва напоследък за корпоративни системи.</p>
	<p>Популярен, безплатен, с отворен код, сървър за база данни.</p> <p>Популярността му се дължи до голяма степен на факта, че работи ефективно с по-малко ресурси. Той прави това за сметка на определени функции, като транзакции и външни ключове, което го прави по-малко подходящ за някои видове приложения и за публикуване на данни в интернет.</p> <p>Работи с Windows и Unix-подобни системи.</p>
	<p>Oracle е един от класическите сървъри на бази данни, многоплатформена сървърна система</p> <p>Най-широко използваната, но не е най-лесна за инсталиране.</p> <p>Работи с Windows и Unix-подобни системи</p>
	<p>Interbase е популярен продукт от Borland, който получава отлични оценки в потребителските проучвания.</p> <p>Основните му предимства са неговите скромни хардуерни изисквания, но с възможност за разширяване до много големи системи, както и способността му работите ефективно, без непрекъсната намеса от администратора на базата данни.</p> <p>Работи с Windows и Unix-подобни системи</p>

	<p>Firebird е безплатен с отворен код софтуер за база данни, въз основа на изходния код на Interbase 6 Borland.</p> <p>Firebird е ориентиран към Mac OS X</p>
	<p>DB2 е софтуер в сферата на "голямото желязо".</p> <p>Тя е достъпна за Windows , Linux, NT, и Unix</p>
	<p>База данни от корпоративен клас, освободена от SAP като система с отворен код.</p> <p>Достъпен за Windows и няколко Unix-подобни системи</p>

I.4.4. Сървъри на приложенията

Сървърите на приложенията (Application Servers) в клиент-сървър архитектурата на приложения, извършват съхранение и обработка на бизнес логиката. В точка 2.3.3 бяха разгледани общите характеристики на сървърите на приложения.

I.4.5.Сървъри за съвместна работа (Groupware Servers)

Сървърите за съвместна работа са предназначени да осигурят връзка между потребителите в една групова система (Groupware) независимо от местоположението им.

Те управляват неструктурирана информация, като текст, изображения, поща, списъци със съобщения. Съхраняват данните, които поддържа груповия софтуер и често ще действат като сървъри за електронна поща, календарни сървъри и файлови сървъри и пр.

Microsoft Exchange е пример за фирмен Groupware сървър. Осигурява бизнеса с електронна поща, календар и контакти чрез компютър или телефон и мрежа, така че служителите могат да останат

свързани и в синхрон, независимо къде се намират. Сървърът съдържа вградени много средства за защита срещу спам и фишинг заплахи.

Другият типичен примре на фирмен Groupware сървър е IBM Lotus Sametime.

I.4.6.Уеб сървър

Уеб сървърът приема http заявките на клиентите или уеб браузърите и ги обслужва, връщайки http отговори и допълнително обогатено съдържание. Уеб сървър е и самият компютър, който изпълнява тази функционалност.

Функции на уеб сървърните програми:

1.Приемане на http заявки от мрежата и връщане на http отговори на запитващия. Отговорът може да е http документ, текстов файл, изображение или друг вид документ.

2.Изпращане на съобщение за грешка, ако има проблеми със обслужването на заявката или заявката е подадена неправилно.

3.Водене на подробен отчет за клиентските заявки и и отговорите на сървъра.

4. Автентификация при достъп до определени ресурси.

5.Обслужване на статично и динамично съдържание.

6. Поддръжка на https (чрез SSL или TLS) за осъществяване на защитени връзки.

7.Компресия на съдържанието, за да се намали обемът на отговорите (натоварването по канала).

8.Виртуален хостинг за обслужване на множество сайтове през един единствен IP адрес.

I.5.Инструментариум за разработка на Клиент-Сървър системи

I.5.1. Разработка на Client-Server приложение във Visual Studio .NET с Visual Basic.Net

Среда за изпълнение на програмите:

NET Framework

Език за програмиране:

Visual Basic.NET

Среда за разработка на програми:

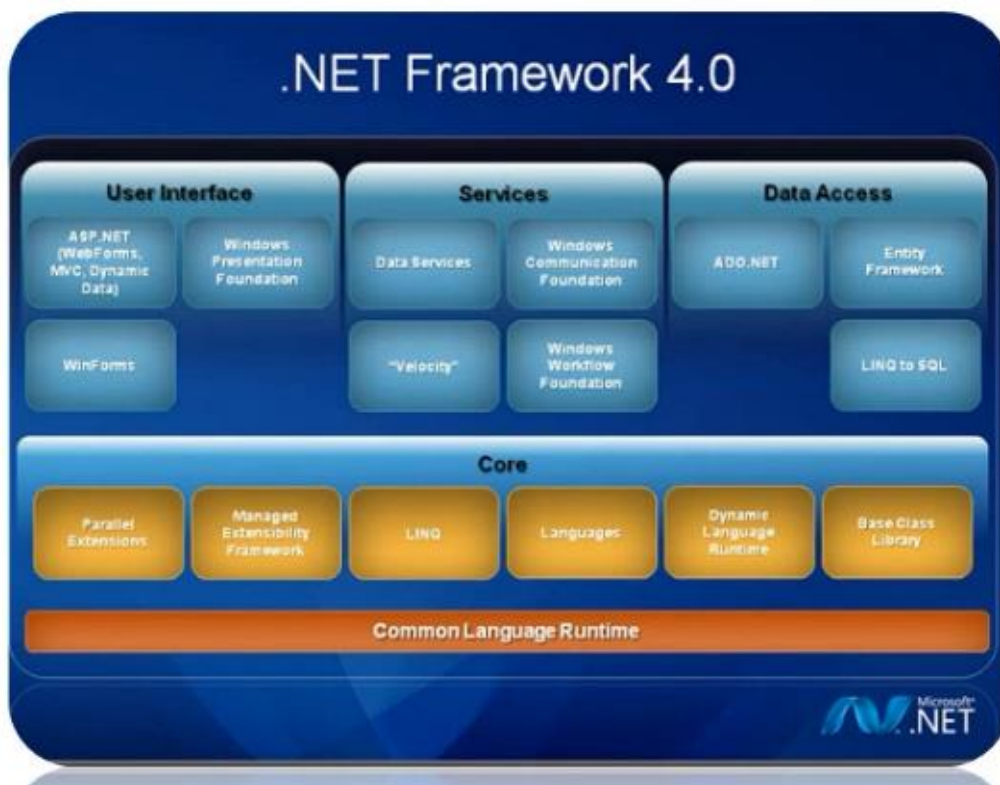
MS Visual Studio

Net Framework

Виртуална машина, която управлява изпълнението на програмите.

Поддържа се от Microsoft за Windows операционни системи, а за други операционни системи - чрез трети фирми.

Включва библиотека от класове, за решаване на основните програмни проблеми.



фигура 8. Структура на .Net Framework (източник...

Библиотеката от класове покрива огромна област от програмни нужди в различни предметни области, включва:

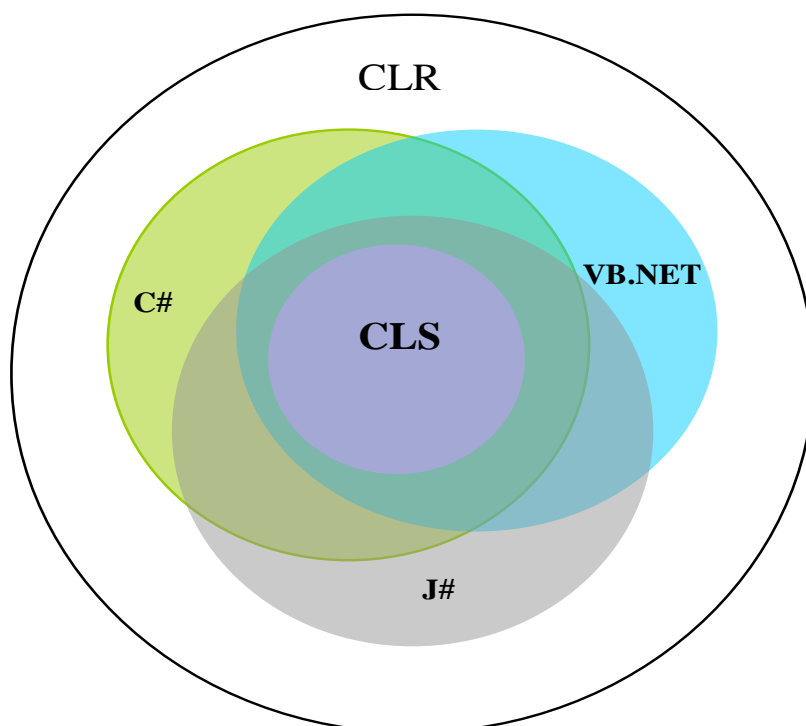
- интерфейса,
- достъпът до данните,
- връзката с базата данни,
- защитните средства,
- средства за уеб приложенията,
- цифровите алгоритми,
- връзките с мрежата и др.

Виртуалната машина се нарича Common Language Runtime (CLR), аналогично на JVM (Java Virtual Mashine).

Междинният език на .NetFramework, наречен Common Intermediate Language (CIL) е процедурно-независим набор от инструкции, дефиниращи зареждане, съхранение, инициализиране, извикване, аритметични и логически операции.

Междинният CIL код е поместен в .NET асемблита. Асемблитата се състоят от един или повече файлове, един от които трябва да съдържа манифест с метаданни за асемблита.

Езици за програмиране предлагани от Microsoft за CLR и минималните изисквания CLS (Common Language Specification) към езици от трети производители.



фигура 9. CLS -

В таблицата е обобщен пример

Таблица

VB.Net	CLS	C#
hero .Name = "SpamMan" hero .PowerLevel = 3	Ok. Стандартен код на международен език (CIL)	hero .Name = "SpamMan"; hero .PowerLevel = 3;
With hero .Name = "SpamMan" .PowerLevel = 3 End With	↑ Специфичен код на международен език (CIL) генериран от компилятора на VB.Net	

VISUAL STUDIO

Visual Studio е основната интегрирана среда за разработка (Integrated Development Environment IDE) на Microsoft.

Visual Studio може да бъде използвано за разработка на конзолни и графични потребителски интерфейси с Windows Forms приложения, Web приложения и Web услуги (сървиси).

ОСНОВНИ ЕЛЕМЕНТИ

- Редактор на код и дебъгер
- Дизайнер на Windows Form
- Дизайнер на Web Form
- Дизайнер на класове
- Дизайнер за бази данни
- Редактор за свойствата на компонентите
- Solution Explorer
- Server Explorer

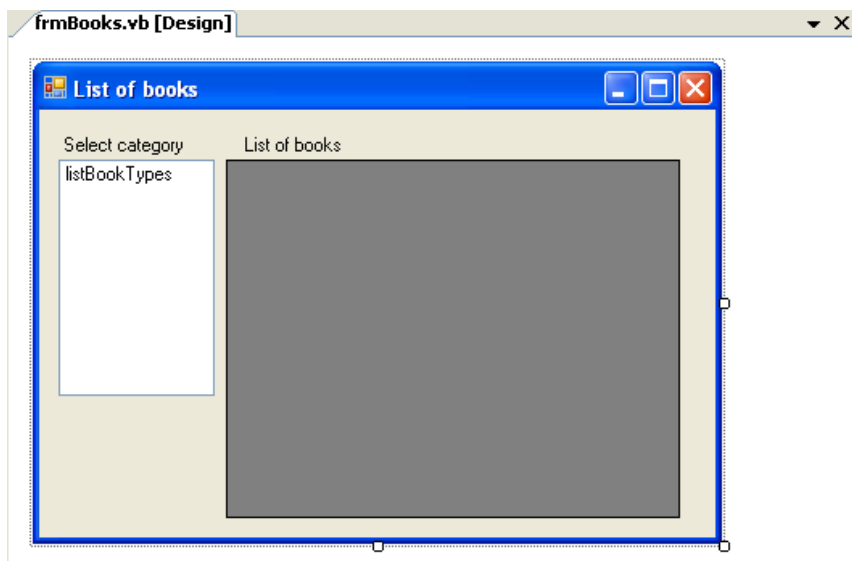
Редактор на код и дебъгер

Visual Studio включва интелигентен редактор на код - поддържа подразбиращи се подсказки и редакции на кода, оцветяване в различни цветове на класовете, променливите, кода и пр.

Вграден дебъгер - Обслужва приложения, написани на езиците, поддържани от Visual Studio.

Дизайнер на Windows Form

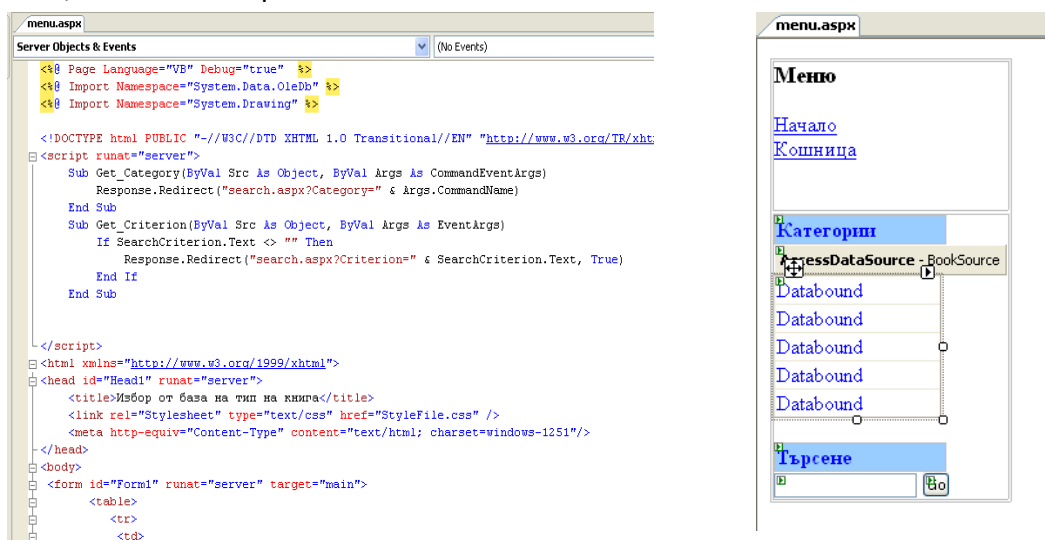
Този дизайнер е предназначен за изграждане на Windows приложения с графичен потребителски интерфейс.



фигура 10. Windows приложение

Дизайнер на Web Form

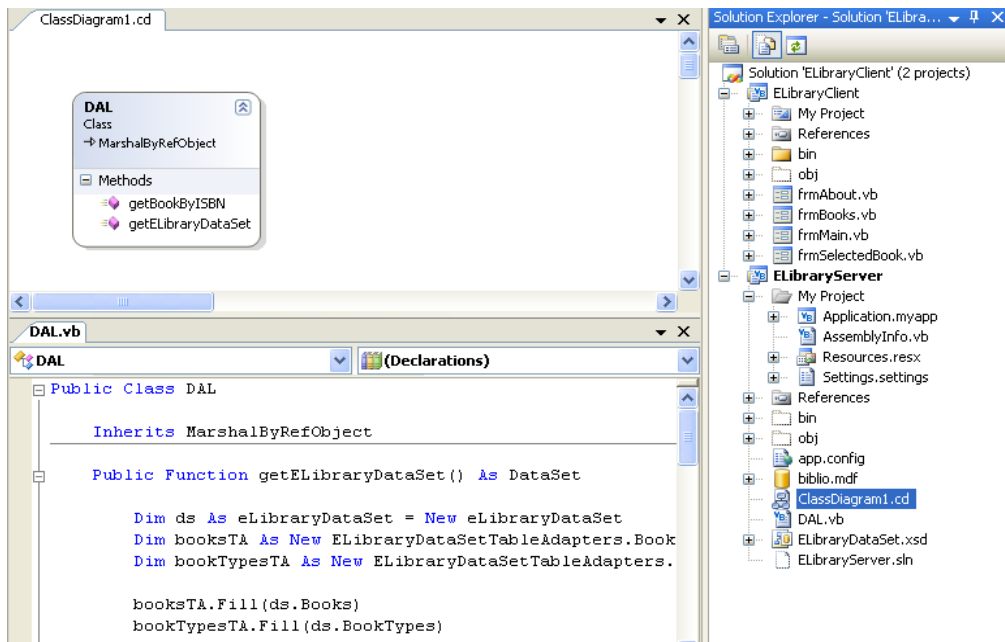
Дизайнер на Web Form е за създаване и редактиране на уеб страници. Използва се за разработка на ASP.NET приложения. Поддържа HTML, CSS и JavaScript.



фигура 11. Дизайнер на Web Form – в режим Source и Design

Дизайнер на класове (Class designer)

За създаване и редактиране на класове, включително създаване на обекти от класовете и достъп до тях. Използва UML моделирането. Генерира VB.NET и C# код за класове и методи. Генерира клас диаграми за ръчно писаните класове.

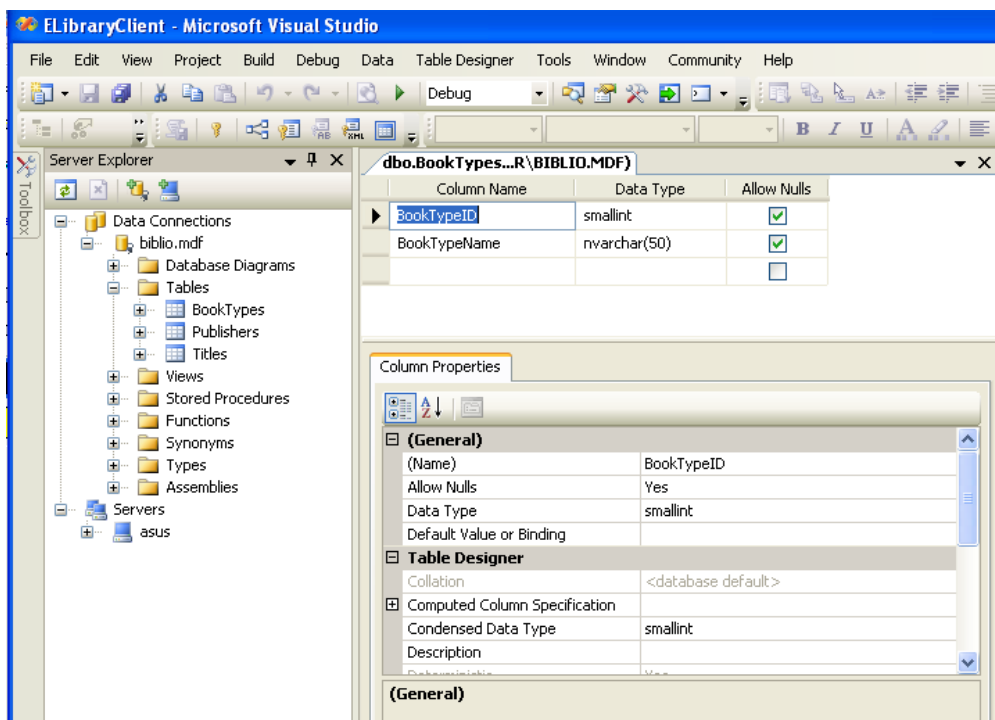


фигура 12. Дизайнер на класове с клас диаграма на един клас

Дизайнер на данни (Data designer)

Дизайнерът на данни се използва за редактиране в графичен режим на схемата на базата данни – създаване на таблици, първични и външни ключове. Използва се за създаване и редактиране на изгледи (view).

Екраните са идентични с тези на Microsoft SQL Server Management Studio



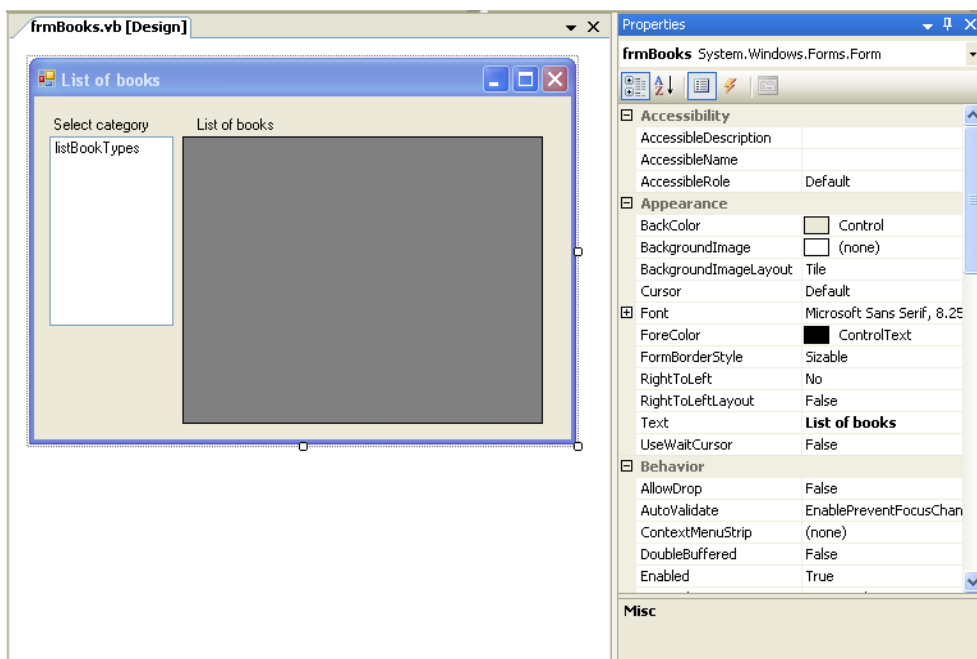
фигура 13. Изглед на екрана за данни

Редактор на свойства (Properties Editor)

Редактиране на свойства в графичен потребителски интерфейс.

Списък с достъпни свойства в режим само за четене и такива, които могат да бъдат редактирани.

Свойства достъпни за редактиране за класове, форми, уеб страници и др.



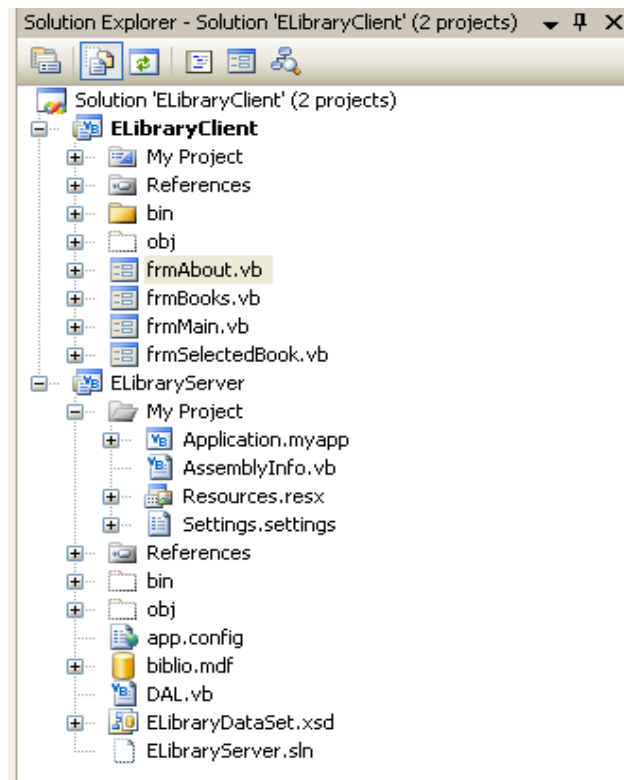
фигура 14. Изглед на редактора на свойства

Solution Explorer

Solution Explorer се използва да управлява и показва файловете в solution – *приложение, което може да включва няколко проекта.*

Списък от файлове с код и други ресурси, които се използват за изграждане на приложение

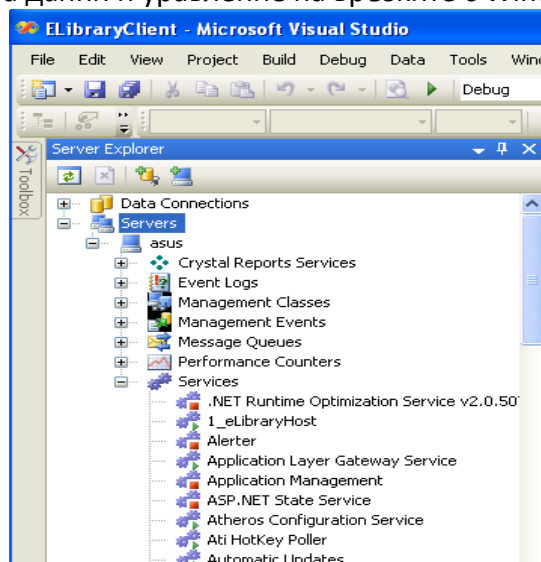
Файловете в solution са подредени йерархично.



фигура 15. Изглед на Solution Explorer

Server Explorer

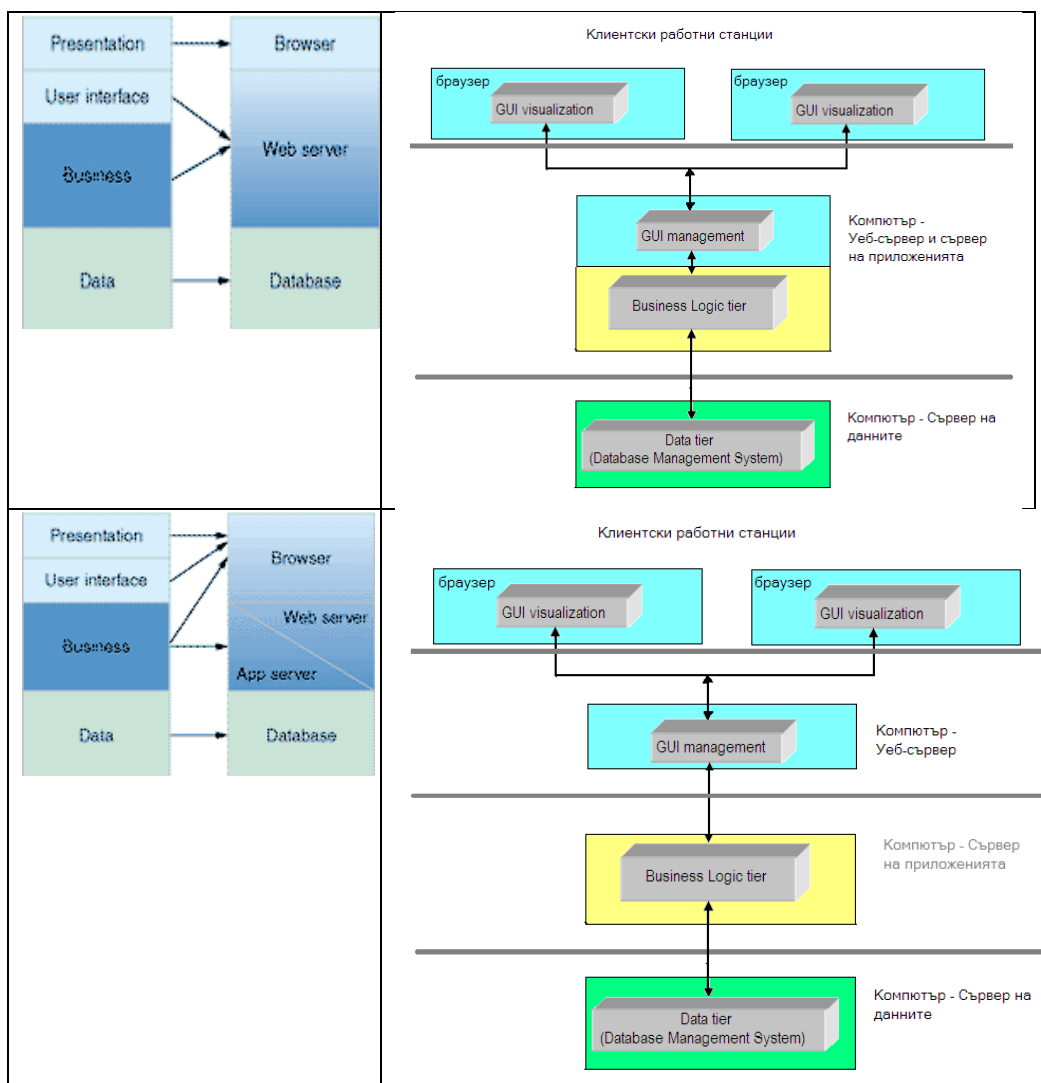
Server Explorer е средство, което се използва за управление на връзките с базата данни и управление на връзките с Windows Services.



фигура 16. Server Explorer – връзка с Windows Services

I.5.2.Инструментариум за разработване на уеб базирани клиент-сървър системи

Нека да разгледаме трислойната уеб базирана клиент-сървър архитектура, за да видим как хардуерните, софтуерните компоненти и обработващите функции се отнасят една към друга от гледна точка на процеса на обработката на информация фиг. 13.



фиг. 13. Трислойна уеб архитектура

Всички функции по вход/изход се извършват главно на клиентската машина чрез уеб браузър. Действията на потребителския

интерфейс като въвеждане на данни, проверка на данни, контрол на обработката и форматиране на изхода се извършват на компютъра от първия слой, често дори без да взаимодействат с останалите системни компоненти.

Основната обработка на информацията се пада на уеб сървъра, и по-точно на тази част от него, която се нарича сървър на приложенията - Web application server. Тук аритметичните и логически процедури са програмирани за да осъществят основните задачи по обработка на информацията - логиката на работа или бизнес логиката на системата.

Съхранението на данните се осъществява на сървър за данни или database server. От тук се извличат данните необходими за обработката. Чрез този сървър се осъществява поддръжката на актуалността на данните посредством добавянето, промяната и изтриването им.

Системните функции са отделени, въпреки че за крайния потребител те остават скрити. Той взаимодейства само с потребителския интерфейс. Основната идея на това отделяне е, че специализираните компоненти извършват специализирана, конкретна работа, за която те са най-добре измислени. Разграничават се специфичните дейности за всеки един слой и тези специфични дейности могат да се възложат на различни специалисти. Разбира се всички дейности могат да се извършат на едно единствено работно място, върху един компютър, но и в този случай използваните средства и технологии ще са еднакви, което прави разработката на уеб приложения рутинна работа.

Необходими умения за създаване на уеб базирано трислойно клиент/сървър приложение

Ще посочим какви умения са необходими за създаване на уеб базирано трислойно клиент-сървър приложение с трислойната клиент-сървър архитектура.



фигура 17. Необходими умения за създаване на уеб базирано трислойно клиент-сървър приложение

За системния вход/изход

Потребителят взаимодейства със системата посредством функциите на потребителския интерфейс чрез уеб браузера на клиентския компютър. Трябва да са налице знания за такива средства за разработка, които програмират браузера, за да извърши работите по форматиране на изхода, въвеждане на данни, проверка за валидност и др. Необходимо е познаване на XHTML (HTML), CSS, DHTML и XML (eXtensible Markup Language) за представяне на структури от данни. Основният език за програмиране на браузера е JavaScript, който се използва за да се манипулира HTML и структурите от данни.

CSS

Използване на CSS

Формат

Селектор – HTML tag, class - клас (има и Pseudo-Classes) или ID – идентификатор

Отваряща фигурална скоба

Декларация

- Свойство
- Стойност

Затваряща фигурална скоба

Типове CSS

- Inline: This affects only the single instance of an element.
<p style="color: red">Red Text </p>
- Document level: This is the default when “(classes)” is selected in the Page Editor Options and is sometimes referred to as an embedded stylesheet.

```
<style type="text/css">  
p {color: red;}  
</style>
```

- External: This is where your style definitions are contained in a separate CSS page that is linked to one or more pages on your website.
p {color: red;}

Red—Represents a CSS ID.

Green—Represents a CSS class.

Blue—Represents a selector.

Yellow—Represents an inline style.

Свързване към HTML-файла и вграждане

```
<link href="basic.css" rel="stylesheet" type="text/css" />  
<style type="text/css"> @import url("../site.css");</style>
```

Каскада на типовете

1. Inline style
2. Document-level style, including embedded external stylesheets
3. External style using link relative
4. Browser defaults

Class

Дефинициите на класовете започват с точка (.), могат да се използват многократно в страницата и могат да се прилагат към HTML тагове, като се използва атрибута class

ID

Идентификаторите винаги започват с решетка (#) . Могат да се използват само веднъж на дадена страница от сайта. Могат да се прилагат към HTML тагове, като се използва атрибута ID.

<div> и

Тези два HTML тага са много важни при CSS. Когато се използва <div>, CSS (стила) се прилага към специфична секция от страницата, чрез контекстуалните селектори (тези, които започват с #) .

А когато се използва , стилът се прилага към избрания текст вътре в HTML тага.

Обработка на данни

От страната на уеб-сървър трябва да се пишат приложения, които ще извършат основните обработки в системата. За програмиране на сървърното приложение са необходими знания за сървърните езици за програмиране VB и C#. От една страна, това са пълнофункционални обектно-ориентирани езици, които съдържат всички средства за програмиране на бизнес логиката и от друга страна, те могат да извикват

вградени сървърни компоненти и да описват отделни функции на системата.

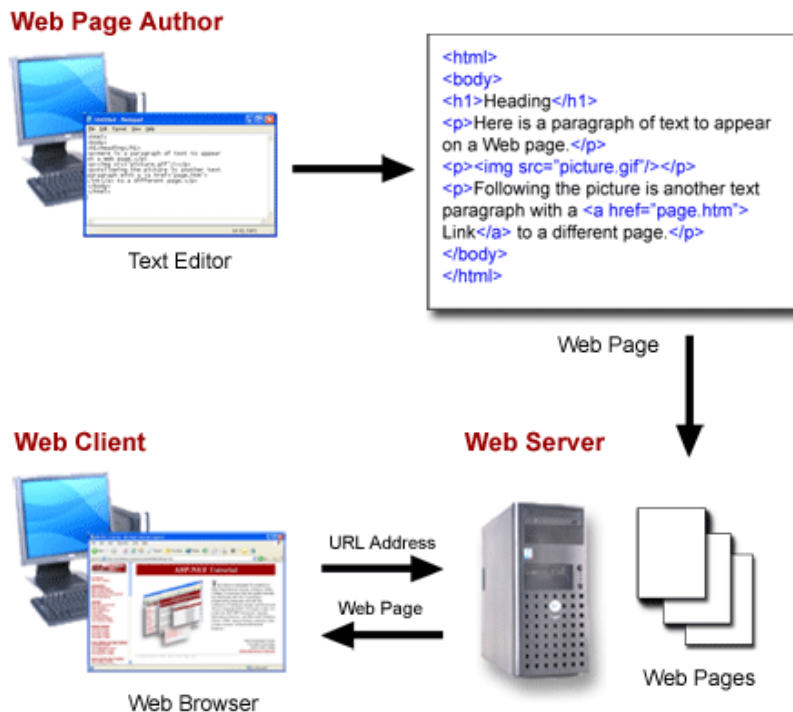
Управление на базата данни

От страна на database сървъра се използват езици като **SQL (Structured Query Language)** за съхранение, управление, редактиране и достъп до данните. Езиците за програмиране на базата данни се използват за създаване на командни процедури и функции за извличане, актуализиране и отпечатване на данните. Езиците за обработка на данни извикват тези SQL команди и съхранени процедури за да изпълнят обработките в базата данни, спестявайки на разработващия системата да пише програмен код на ниско ниво. **ADO (Active-X Data Objects).NET** компонентите осигуряват интерфейса между скриптовите езици и тези дейности по обработката на данните в базата данни.

Уеб програмиране

Като част повече на обработката, отколкото на доставката на информация клиентските и сървърните компоненти на уеб-базираните системи имат разширена роля. PC клиента, Web сървъра, и база данни сървъра работят строго координирано, за да генерират и изведат съдържанието на уеб страницата върху компютърния екран.

При традиционното изграждане на страници (фиг.15) разработващия създава уеб-документа, използвайки текстов или HTML редактор, зарежда този документ в уеб-достъпна папка на някой сървър, получавайки URL –адрес. Потребителите въвеждат този адрес в браузерите си и получават съдържанието на страницата.



Фиг. 15. Традиционно изграждане на уеб-страници.

От тази гледна точка на уеб сървъра не му остава кой знае колко работа. Той просто съхранява документите в достъпна папка и ги предоставя на потребителя.

При съвременния модел на обработка на информацията обаче, ситуацията коренно се е променила. Създаването на уеб страниците се извършва динамично – тяхното съдържание се променя в зависимост от действията на потребителя. Страниците съдържат VB код за обработка на аритметическите и логическите операции, SQL директиви и обръщения към процедури за извличане, визуализиране, актуализиране и съхранение на данните в базата данни. Страниците могат да съдържат също JavaScript, чрез който потребителския интерфейс обработва някои ситуации, когато страницата се връща на браузера.

Изобщо като цяло уеб страницата се превръща в списък от команди насочени към хардуерните и софтуерните компоненти на системата, за да се извърши координиран набор от дейности за генериране, обработка, съхранение и доставка на информация до клиента.

Web System Developer



Text Editor

```
<SCRIPT runat="server">
-- Server-side Script in VB --
</SCRIPT>
<html>
<body>
<h1>Heading</h1>
<p>Here is a paragraph of text to appear
on a Web page.</p>
<p>Text with embedded <% Processing
results%></p>
</body>
</html>
```

Web Page

Web Client



Web Browser

Web Server



Web Pages

URL Address

Web Page

.NET Compiler

Finished
HTML
Page

Compiled
Web
Page

.aspx Page

Database Server



Фиг. 16. Съвременна разработка на уеб-страници

Уеб сървърът координира всички дейности по обработката на информацията. Когато URL адресът на една страница за първи път се получи от сървъра, тя не се доставя директно на потребителя, който я е поискал. Страницата се преобразува чрез ASP.NET процесор (aspnet_isapi.dll) в **софтуерен клас**. Това е в действителност компилиран софтуерен обект, компютърна програма за изпълнение на входни, обработващи, изходни и съхраняващи функции. На уеб сървъра се осъществява обработка на всякакви процедури, които от сървъра на базата данни извличат информация. Всички браузерни скриптове в страницата се игнорират от уеб сървъра и се препращат към клиентския компютър за активиране, когато страницата се върне. На практика, уеб сървърът изгражда страницата на базата на вградените програми и функции, връщайки обогатено съдържание към браузера.

HTML

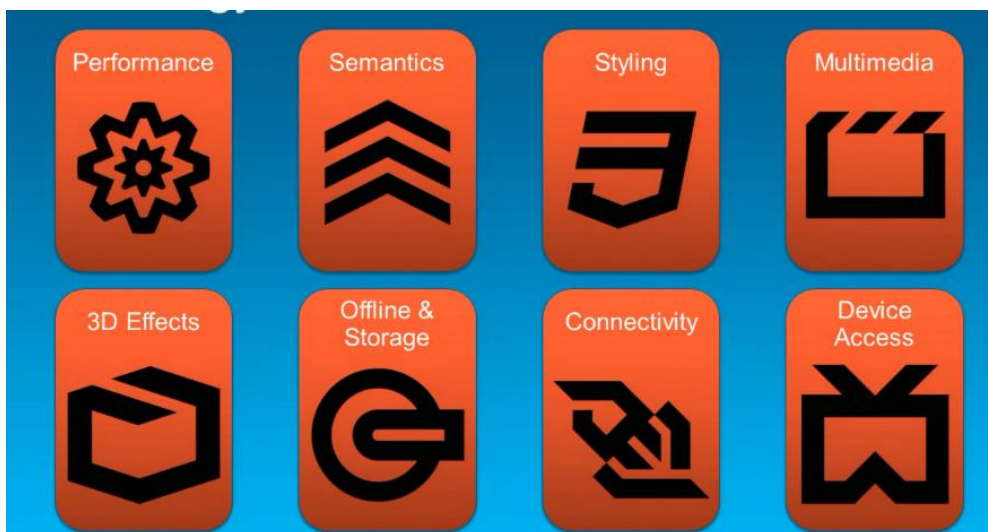
HTML (Hypertext Markup Language) е формален език, който реализира една по-обща спецификация разработена от международната организация по стандартизация ISO; (тази по-обща спецификация се нарича **SGML** - Standard Generalized Markup Language).

Под динамичен HTML (DHML) се разбира съвкупността от HTML + CSS + JavaScript.

През 2011 година беше представена новата спецификация на HTML, която включва някои нови технологични елементи.

Технологични области, разработени в новата спецификация HTML 5

Според разработчиците на спецификацията, основните технологии застъпени в HTML 5 са следните осем [2] :



фигура 18. Технологии застъпени в HTML 5

- Изпълнение
- Семантика
- Силове
- Мултимедия
- 3D ефекти
- Офлайн работа и съхранение на данни
- Свързаност
- Достъп до устройства

Нови тагове в HTML 5

До този момент са създадени 28 нови тага [3] , несъществуващи до този момент в HTML 4. В таблицата по-долу са посочени основните от тях.

Таблица 2. Нови тагове в HTML 5

Tar	Описание на действието
<article>	Дефинира статия
<aside>	Съдържание извън основното на страницата
<audio>	Аудио съдържание
<command>	Команден бутон
<datagrid>	Дефинира данни в дърво

<datalist>	Списък от типа „dropdown“
<datatemplate>	Макет (template) на данни
<details>	Детайли за елемент
<dialog>	Дефинира диалог
<embed>	Външно интерактивно съдържание или софтуерен компонент добавящ специфични атрибути към приложение (plugin)
<eventsource>	Дефинира цел (target) за събитие изпратено от сървър
<figure>	Дефинира групирано съдържание
<figcaption>	Заглавие на групирано съдържание
<footer>	Дефинира последният текстов ред от кода (footer) на секция или страница
<header>	Дефинира първият текстов ред от кода (header) на секция или страница
<mark>	Маркиран текст
<meter>	Дефинира мерната единици в зададен диапазон
<nav>	Дефинира линковете за навигация
<nest>	Точка на вмъкване в макета на данни (datatemplate)
<output>	Дефинира някои типове за изход
<progress>	Развитие на стартираната задача
<rule>	Дефинира правила за актуализиране на макет (template)
<section>	Секция
<source>	Източник
<time>	Дата/време
<video>	Видеосъдържание

Нови атрибути в HTML 5

Новите атрибути са дадени в таблица 3 [3]. Те не са съществували до този момент в HTML 4, но в съзвучие с новите тагове са добавени към спецификацията.

Таблица 3. Нови атрибути в HTML 5

Атрибут	Стойност	Описание на действието
contenteditable	true false	Посочва дали потребителят има право да редактира съдържанието
draggable	true false auto	Посочва дали потребителят има право да издърпва съдържанието
irrelevant	true false	Посочва дали елементът е ирелевантен. Ирелевантните елементи не се извеждат на екран.
ref	URL / id	Референция към друг документ или част от документ
registrationmark	reg_mark	Поставя знак за регистрация на елемента - (®)
template	URL / id	Референция към друг документ или част от документ който трябва да бъде приложен към елемента

Някои от новите тагове в HTML 5 са пределно ясни по смисъл и значение – например <audio>, <video>. Докато преди трябваше да се използват различни плъгини като Silverlight и Flash, в новата спецификация могат да се използват директно двата тага. В <audio>, трябва да се посочи източникът и атрибутът **controls**, за да се даде възможност на потребителя да управлява изпълнението. Управлението се визуализира по различен начин в различните браузери. Могат да се използват и други атрибути като: autoplay, loop и пр. Кодът изглежда примерно така:

```
<audio src="../../Sleep Away.mp3" controls="controls">
  Your browser does not support the audio element.
</audio>
```

По подобен начин се задава и тагът <video>.

Интересен е също така новият таг <canvas>, който осигурява едно бяло платно (канава) за рисуване. Необходимо е обаче да се познава JavaScript, за да може да се чертае по канавата. Предишният стандарт за рисуване в прозореца на браузера е Scalable Vector Graphics (SVG).

Нови семантични елементи в HTML5

От изброените по-горе тагове, има няколко с ясно предназначение, тъй като обозначават конкретни части от сайта или действия в него. Например:

- article
- aside
- figure
- footer
- header
- mark
- nav
- section
- time

Без да са необходими допълнителни знания (освен стандартния английски език), се отгатва какво върши всеки един нов таг.

Може да се изгради сайт, който да използва новите елементи посочени в двете таблици и поради тяхното семантично богатство, ще е ясна структурата на сайта и действието му. Ето графичен пример:



фигура 19. Изграждане на сайт с новите семантични тагове в HTML 5

Значението на **header** и **footer** е ясно – те дефинират съответно първият и последният текстов ред от кода на страницата. Елементът **nav** ще съдържа линковете за навигацията из страниците на сайта. В **section** ще бъдат разположени линковете на менюто. А елементът **aside** ще съдържа всички статии, линкове, рекламни банери, които нямат връзка с основния сайт. Те ще бъдат оформени чрез отделни елементи **article**. Тагът **article** е особено актуален в момента – неговата необходимост

възникна във връзка с бързото развитие на блоговете – там всяка отделна статия трябва да се зададе самостоятелно.

Не може да има никакво съмнение и в предназначението на тага **time** - дефиниране на датата и времето, или **mark** - за маркиране на части от текст.

Новите тагове в HTML 5 са толкова семантично ясни, че в първия момент на запознаване с тях, трудно се възприемат като нови. Тези тагове обаче наистина не съществуват в старата спецификация на HTML 4. Като думи и понятия те се използват, но или като имена на атрибути (например `id="footer"`) в `<div>` тага, който вече не съществува в HTML 5, или като част от различни, добавящи специфика, софтуерни компоненти.

XML

XML (Extensible Markup Language) е прост формален език за представяне на йерархично организирани данни. Данните се представят като вложени един в друг тагове. XML не е ориентиран към представяне на някакъв конкретен вид данни.

Езикът се нарича "Extensible", - *разширяем*, защото не е фиксирано множеството на използваните тагове.

Най-важните задължителни изисквания, за да бъде правилно построен един XML файл, са:

- да има един единствен таг на най-високо ниво, в който се влагат всички останали тагове;
- влагането на таговете е без пресичане (аналогично на правилото за влагане на цикли в програмирането)
- Стойностите на атрибутите на таговете са задължително са поставени в кавички

`<човек име="Иван" > </човек >`

XHTML: XML + HTML

Формалният език XHTML 1.0 е конкретна XML спецификация, в която се реализират таговете на HTML 4.

По-нататък в настоящия учебник ще разгледаме по-подробно XML.

Компоненти на кода на уеб страницата

Кодът на уеб страницата (файл с разширение **aspx**) включва:

- HTML тагове, и
- Програмен код на някой от езиците за .NET – VisualBasic.Net, C# или друг

Този код се изпълнява от сървъра, и се генерира HTML код.

Уеб страницата е съчетание от потребителски и съвършен код, финалната страница, която се връща на браузъра, съдържа само чист HTML документ. Това е ключовият момент. Съвършените скриптове и средства за обработка на данните, които се появяват в оригиналната страница, не са част от върнатата на браузъра страница. Само крайните резултати и HTML кода са видими. Съвърът е изпълнил скриптовете и вмъкнал резултатите в HTML кода. Програмните кодове са заменени от техните изходни данни. Крайната страница не е толкова различна от тази, която бихме написали използвайки само чист HTML. Разликата е в това, че съвърът е написал кода вместо нас.

Тъй като крайната (върнатата) страница представлява чист HTML код, всеки браузер би могъл да визуализира резултатите. Не е нужен специален софтуер на клиентския компютър. Това означава също, че потребителят не може да види оригиналния код. В този смисъл съвърът осигурява защита на програмите – потребителят не може да види кода, обработващ бизнес логиката и бизнес правилата, вградени в него.

Разбира се за всичко това е необходима практика. С практически упражнения, всеки ще бъде способен да интегрира HTML код, CSS стилове, JavaScript, съвърно-базиран (VB, C#) код, SQL команди и .NET компоненти в едно подредено цяло, насочвайки всички хардуерни, софтуерни компоненти и данни на системата за изграждане на веб страница, като част от цялостната веб-базирана информационна система.

ASP.NET обкръжение

Разработката на веб приложения ще осъществяваме въз основа на ASP.NET технологията. ASP.NET е програмна структура (framework) на Microsoft, която позволява разработката на веб приложения и веб сървиси (услуги).

ASP.NET е най-бързо разрастващата се среда за веб разработки днес. Тя привежда в действие много професионални високо специализирани търговски сайтове като Home Shopping Network, Dell Computer, Merrill Lynch, the London Stock Exchange, USA Today, Bank One, Century 21. Съществуват и хиляди други сайтове на по-малки фирми базирани на ASP.NET.

Програмно обкръжение

ASP.NET има богат набор от софтуерни елементи за работа с обектно-ориентирани езици. Много голяма функционалност има скрита и работи фоново (като background). Такива са например вградените, но

скрити функции за следене на сесията, за състоянието на страницата и др.

Програмното обкръжение на ASP.NET би могло да се нарече по-скоро декларативно програмиране, отколкото конвенционалното логическо програмиране. Това са програми или скриптове, които описват по-скоро **какви** обработки да се извършват, отколкото **как** да ги извършват. Традиционното програмиране изисква наличието на детайлни инструкции описани в програмни стъпки, които компютъра да изпълни последователно за да генерира изхода на системата. Под управлението на ASP.NET тази последователност от стъпки в голямата си част е капсулирана в предварително създадените и достъпни за използване софтуерни обекти. Задачата на програмирането е да подбере подходящия обект, да поиска от неговите свойства и методи да върнат очаквания резултат. Малко внимание се обръща на детайлите относно това как се извършват процесите вътре в капсулирания обект.

Програмистът работи с набор от инструкции на високо ниво. Детайлното програмиране на логиката е изместено от заявки за извършване на услуги. Програмистът просто декларира от каква обработка се нуждае; ASP.NET връща резултата. Тежката работа по програмирането е пакетизирана вътре в извикваните компоненти.

.NET Framework предлага 4500 софтуерни класове, в които е капсулирана богата функционалност като XML, достъп до данни, качване на файлове, генериране на изображения, транзакции, SMTP поща и още много. Програмното обкръжение включва повече от 25 .NET езика, включително вградената поддръжка за VB.NET, C#, JScript.NET и др.

I.6.Методика за разработка на настолни клиент-сървър приложения

Описанието на методиката за разработка на клиент-сървър приложения използва разработения пример ELibrary.

I.6.1. Разработване на компонентите на ниво потребителски интерфейс с *Windows Forms*

При разработката на компонентите на ниво потребителски интерфейс се използват форми (Windows Forms) и контроли във формите. Ще разгледаме първо най-често използваните типове контроли





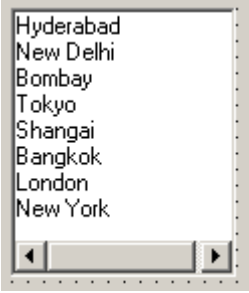
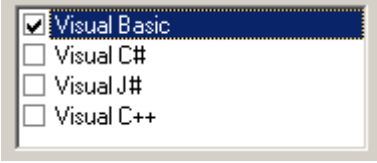
и свойствата им. След това ще се спрем на основните форми, използвани при създаването на примера Elibrary.

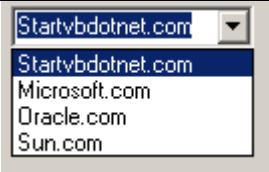
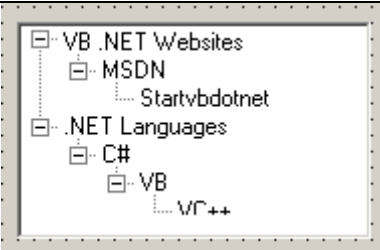
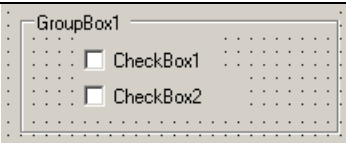

Типове контроли

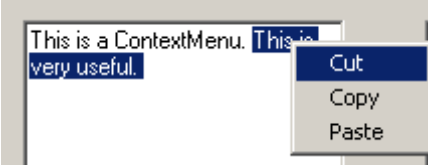
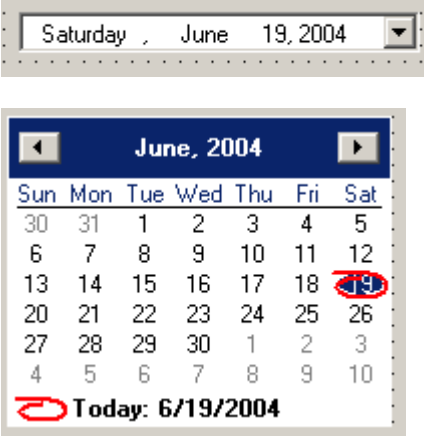

Control е базовият клас за контролите използвани в Windows формите. Класът **Control** е в именното пространство [System.Windows.Forms](#) (пълното име е `System.Windows.Forms.Controls`).

По-важните типове контроли са посочени в таблица.

таблица 1. Основни типове контроли

Контрол	Визуализация
TextBox	 A rectangular text input field with a dotted border and the text 'TextBox1' inside.
Label	 A rectangular label with a dotted border and the text 'Label1' inside.
CheckBox	 Two checkbox controls side-by-side. The first is labeled 'CheckBox1' and is unchecked. The second is labeled 'CheckBox2' and is checked.
RadioButton	 Two radio button controls side-by-side. The first is labeled 'RadioButton1' and is unselected. The second is labeled 'RadioButton2' and is selected.
ListBox	 A vertical list box with a scroll bar at the bottom. It contains the following items: Hyderabad, New Delhi, Bombay, Tokyo, Shangai, Bangkok, London, and New York.
CheckedListBox	 A vertical list box with checkboxes next to each item. The items are: Visual Basic (checked), Visual C#, Visual J#, and Visual C++.

ComboBox	
TreeView	
GroupBox	
Panel Panels cannot display caption; Panels can have scrollbars where as GroupBoxes can't. Default value of the AutoScroll property is set to False. Set it to True if you want a scrollbar with the Panel.	
MainMenu is the container for the <i>menus</i> ; Menus	

<p>Context menus are <i>menus</i> that appear when an item is right-clicked.</p>	
<p>DateTimePicker</p> <p>In Date TimePicker when we want to make a selection we click on the drop-down arrow and select the date from the MonthCalendar which is displayed.</p>	
<p>StatusBar</p>	
<p>Източник: http://www.startvbdotnet.com/controls/default.aspx</p>	

Свойства на контролите

По-важните свойства на обектите от класа **Control**, наследени във всички типове контроли, са посочени в таблица.

таблица 2. Свойства на обектите от клас **Control** и описанието им

Свойство	Описание
BackColor	Gets/Sets the background color for the control

BackgroundImage	Gets/Sets the background image in the control
Bottom	Gets the distance between the bottom of the control and the top of its container client area
CanFocus	Returns a value specifying if the control can receive focus
CanSelect	Returns a value specifying if the control can be selected
ContainsFocus	Returns a value specifying if the control has the input focus
ContextMenu	Gets/Sets the shortcut menu for the control
Controls	Gets/Sets the collection of controls contained within the control
Cursor	Gets/Sets the cursor to be displayed when the user moves the mouse over the form
DataBindings	Gets the data bindings for the control
Enabled	Gets/Sets a value indicating if the control is enabled
Focused	Returns a value specifying if the control has input focus
Font	Gets/Sets the font for the control
ForeColor	Gets/Sets the foreground color of the control

HasChildren	Returns a value specifying if the control contains child controls
Height	Gets/Sets the height of the control
Left	Gets/Sets the x-coordinates of a control's left edge in pixels
Location	Gets/Sets the co-ordinates of the upper-left corner of the control
Name	Gets/Sets name for the control
Parent	Gets/Sets the control's parent container
Right	Returns the distance between the right edge of the control and the left edge of it's container
Size	Gets/Sets size of the control in pixels
TabIndex	Gets/Sets the tab order of this control in its container
TabStop	Gets/Sets a value specifying if the user can tab to this control with the tab key
Tag	Gets/Sets an object that contains data about the control
Text	Gets/Sets the text for this control
Top	Gets/Sets the top coordinates of the control

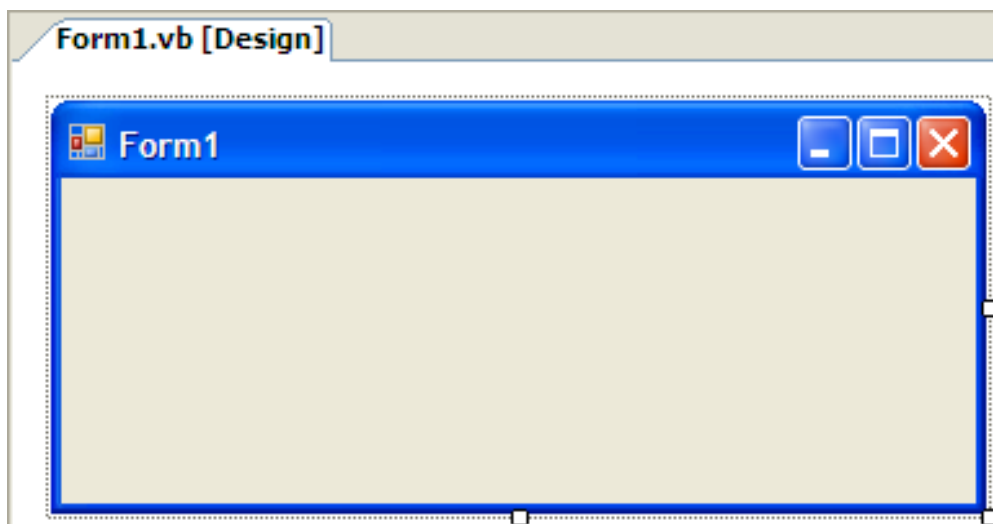
Visible	Gets/Sets a value specifying if the control is visible
Width	Gets/Sets the width of the control

Форми използвани в примера Elibrary

За разработване на компонентите на нивото на потребителския интерфейс в примера Elibrary се използват три основни вида форми:

- Форма от общ вид
- Диалогова форма
- MDI форма (Multi-Document Interface MDI Parent Form)

Форма от общ вид



фигура 20. Форма от общ вид

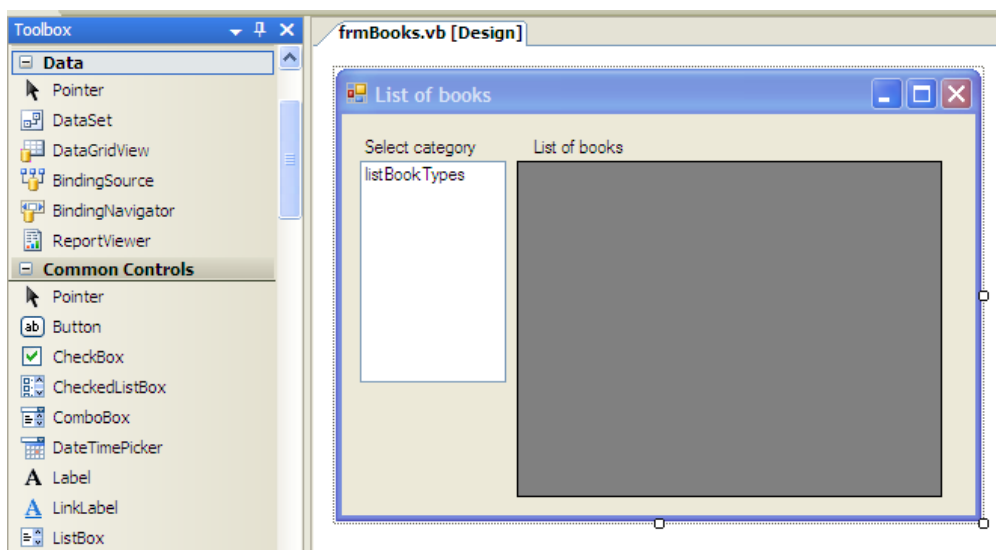
Формата показана на горната фигура е създадена от макета от общ вид (Windows Form) и има свойство `controlBox=true`. Активирани трите контролни бутона (control box) в заглавната лента на формата:

- за минимизиране `MinimizeBox=true`
- за максимизиране `MaximizeBox=true`
- за затваряне (той е винаги активиран при `controlBox=true`).

Във формата могат да се включват компоненти чрез издърпване с мишката (drag-and-drop) от Toolbox.

В примера ELibrary, формата frmBooks е от основен вид.

Включени са два етикета (Label), един грид (DataGridView) и един списъчен контрол (ListBox).



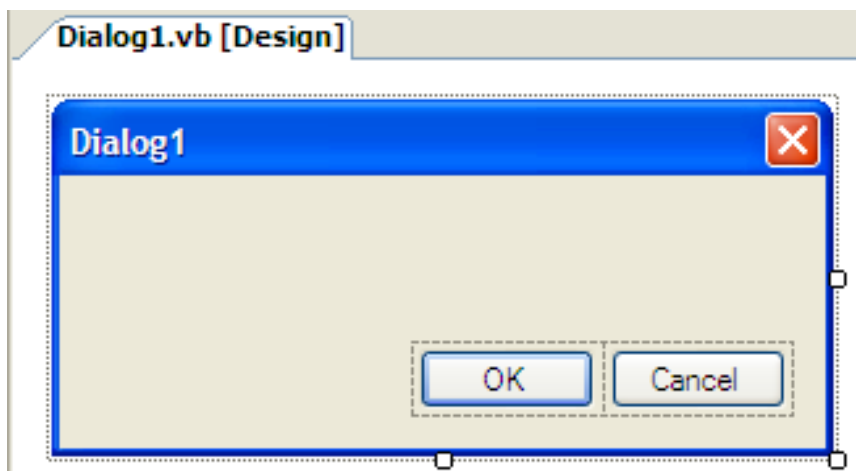
фигура 21. Основна форма и част от наличните в Toolbox компоненти

Диалогова форма

Диалоговата форма е създадена с използването на макет **Dialog** има зададени следните свойства:

- **FormBorderStyle=FixedDialog**
- **controlBox=true**, но е активен само бутонът за затваряне, а другите два бутона са деактивирани с **MinimizeBox=false**

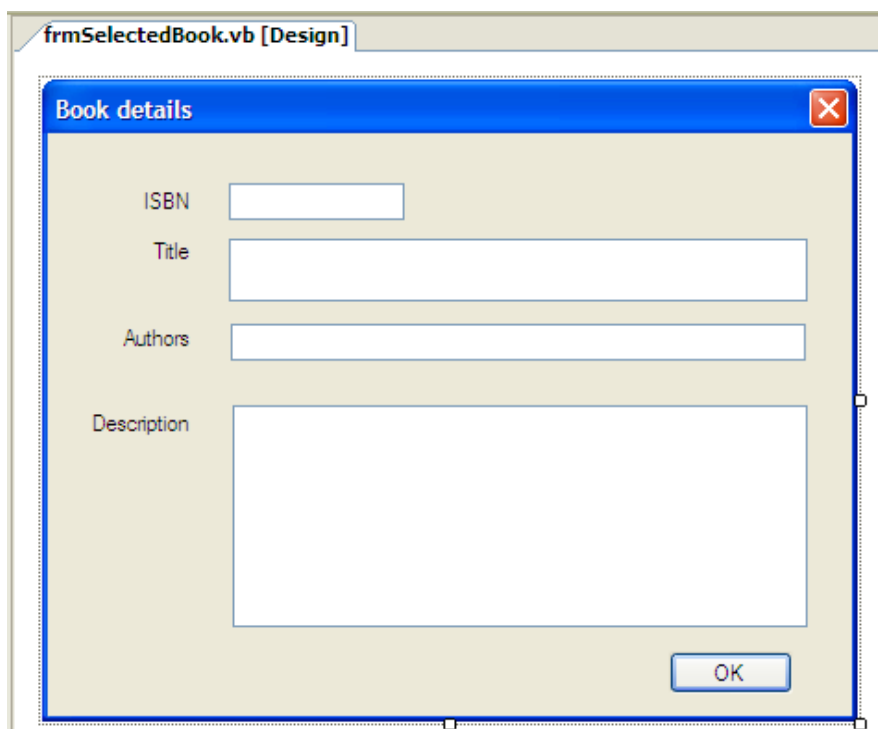
Включени са и два бутона [OK] и [Cancel].



фигура 22. Диалогова форма

В примера ELibrary, формата `frmSelectedBook` е от диалогов вид. Отстранен е бутонът [Cancel]. Добавени са нови полета.

Най-често формите от диалогов тип се активират модално, т.е. до затваряне на формата, само тя е активна от формите на приложението.

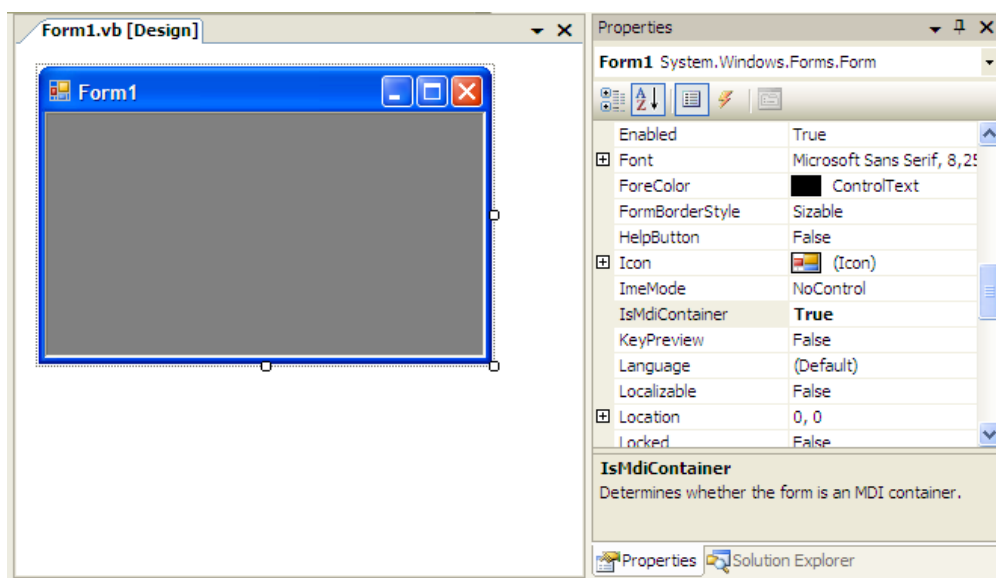


фигура 23. Диалогова форма за избор на книга

MDI форма (Multi-Document Interface MDI Parent Form)

Една форма от общ вид се преобразува във MDI форма като се задава свойството

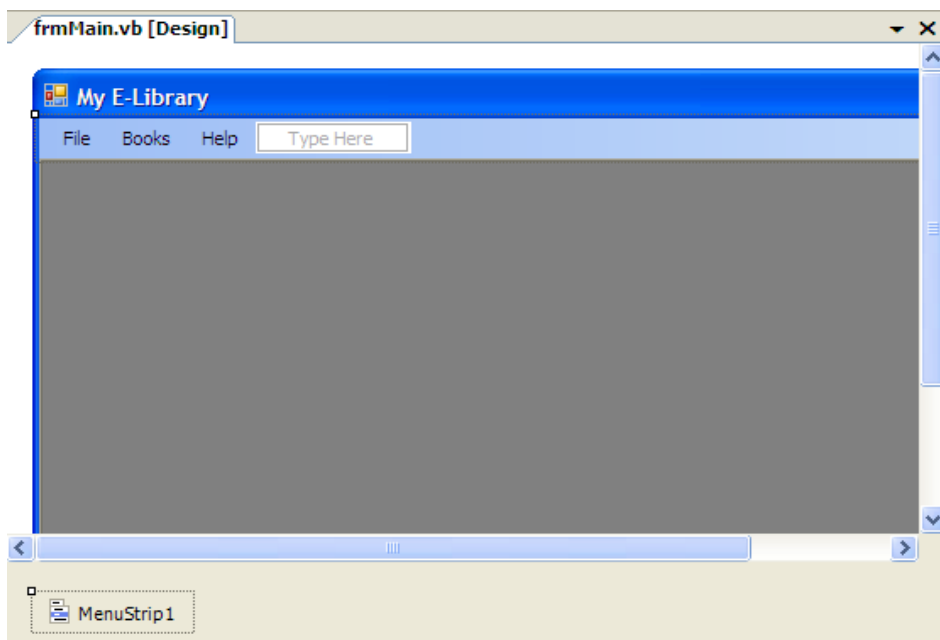
isMdiContainer=True



фигура 24. Общ вид на MDI форма

В примера ELibrary, главната форма frmMain е MDI форма, с добавено меню MenuStrip1.

В MDI форма по принцип могат да се влагат компоненти (етикети, бутони и т.н.), но те ще се скриват от дъщерните форми.



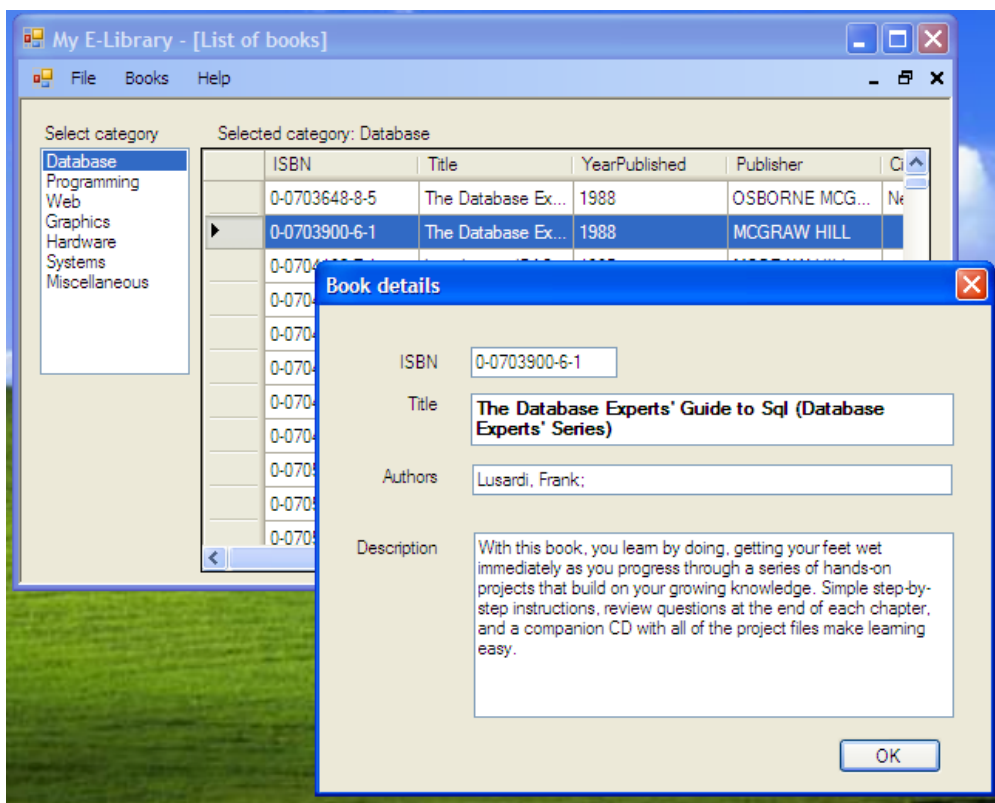
фигура 25. MDI форма с вложени компоненти

Дъщерни форми

Когато една форма се активира като дъщерна, то тя се отваря само в рамките на родителската MDI форма.

В примера Elibrary формата `frmBooks` е дъщерна спрямо `frmMain` и при активиране се показва в нейната рамка.

Формата `frmSelectedBook` не е дъщерна на `frmMain`. Нейното разположение върху екрана е независимо от `frmMain`.



фигура 26. Дъщерна и недъщерна форма

При дъщерни форми съществува следното ограничение – те не могат да се активират със `.ShowDialog()`, т.е. не могат да се активират модално.

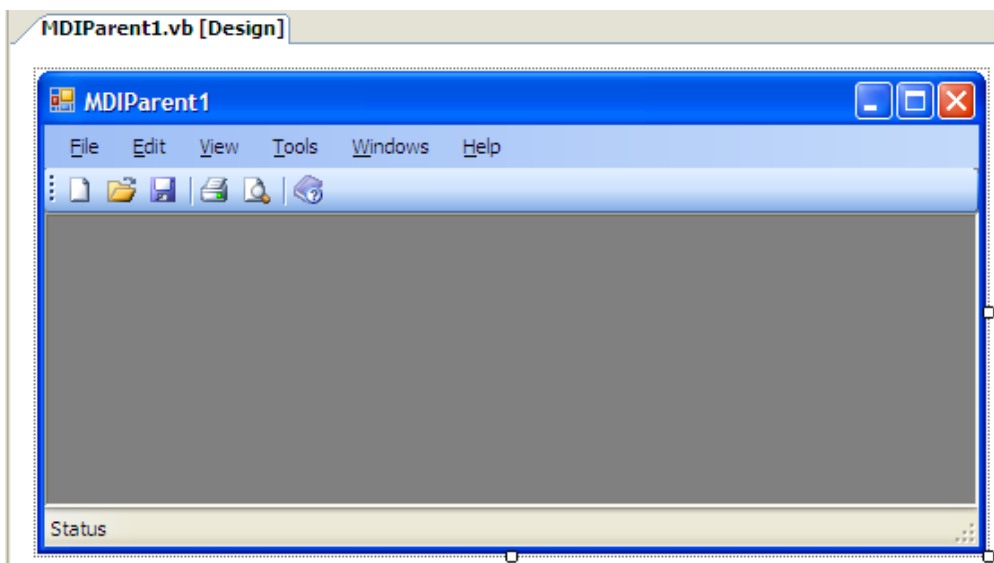
Ако трябва да се направи формата `frmSelectedBook` дъщерна, то трябва да се премахне модалното активиране.

```
frmSelectedBook.FormBorderStyle=FixedDialog  'ВИД РАМКА
frmSelectedBook.MdiParent = frmMain  'става дъщерна
frmSelectedBook.Show()  'НОРМАЛНО ПОКАЗВАНЕ
frmSelectedBook.ShowDialog()  'МОДАЛНО ПОКАЗВАНЕ
```

листинг 1.

В разработения пример всички диалогови форми (с `FormBorderStyle=FixedDialog`), се активират модално – със свойството `.ShowDialog()`.

Най-често използваните MDI форми се създават с макета MDI Parent Form, включват меню и инструментална лента с набор от стандартни функции. Формата изглежда по следния начин:



фигура 27. MDI Parent Form с меню и инструментална лента

Стандартните функции, които не се използват в приложението, трябва да се изтрият. Трябва да се има предвид, че дизайнерът на Visual Studio не изтрива функциите, при изтриване на съответния компонент от менюто. Функциите трябва да се изтрият ръчно от кода на формата.

II. РАЗПРЕДЕЛЕНИ СИСТЕМИ

II.1. Дефиниране на разпределена система

Дотук разгледахме архитектурата на клиент-сървър приложенията, при които обработката на данни се извършва паралелно. Паралелната обработка и клиент-сървър приложенията, които използват този тип, са само част от съществуващите програмни и хардуерни модели. Съществуват разпределени (distributed) системи.

Разпределената система се състои от съвкупност от автономни компютри, свързани с мрежа и middleware софтуер, които дават възможност на компютрите да координират дейността си и да споделят ресурсите на системата, така че потребителите да възприемат системата като единно, интегрирано изчислително средство.

Най-простата дефиниция на разпределената система за обработка на данни според Кенет Бритман е: „съвкупност от компютърни програми, изпълнявани на един или повече компютри, координирайки действията помежду си чрез размяна на съобщения“.

Компютърната мрежа е съвкупност от компютри свързани хардуерно, които поддържат преноса на съобщения по маршрутизиращите протоколи, по такъв начин, че съобщенията да достигнат с голяма вероятност до тяхното местоназначение.

Друга дефиниция: Разпределените системи се дефинират като софтуер който гарантира на своите потребители, че съвкупност от независими компютри, ще се изглеждат като единна и съгласувана система.

Разпределените системи използват компютърната мрежа за пренос на съобщения, но може да се изгради система, в която се пренасят и цели обекти, вместо съобщения.

Дотук използвахме понятието клиент-сървър за ситуациите, при които един компютър изпраща заявка и изисква отговор от друг. Клиентският компютър може да бъде десктоп компютър, лаптоп или дори мобилен телефон, изобщо всяко устройство, което може да изпълни приложната програма, с която потребителят отправя заявките. Сървърът отговаря на заявките, като извлича, обработва и връща

необходимите данни от базата. Ако трябва най-общо да посочим характеристиките на централизираната система, те са:

- Един компонент с неавтономни части.
- Компонентите се споделят от потребителите през цялото време.
- Всички ресурси са достъпни.
- Софтуерът работи чрез изпълнението на един процес.
- Една контролна точка.
- Една точка на прекъсване.

Потребителските заявки, например от типа на търсене на определена книга, софтуерен компонент или някаква друга стока за бита могат да се окажат много сложни за изпълнение. Този тип заявки изискват извличане на данни от много сървъри, коопериране на работата между тях. Сървърите може да не се намират на едно и също място и да не се контролират от една компания. Още повече, голяма част от заявките, които потребителите инициират, изискват намиране на точният сървър (или съвкупност от сървъри) сред множеството сървърни системи, който да съответства най-точно на въпроса и да попълни и филтрира най-точните данни.

Характеристиките на разпределена система, в съпоставка с традиционната клиент-сървър система, дадени най-общо са:

- Множество автономни компоненти
- Компонентите не се споделят от всички потребители
- Ресурсите могат да не бъдат достъпни
- Софтуерът работи чрез изпълнението на едновременни процеси на различни процесори
- Множество контролни точки
- Множество точки на прекъсване.

Какво се опитваме да постигнем при изграждането на разпределена система? Как да се прецени дали дадена система е разпределена. Трябва да се проследят следните елементи:

- Споделяне на ресурси
- Откритост
- Едновременност
- Скалируемост
- Отказоустойчивост
- Прозрачност

Споделяне на ресурси

Облачните системи (cloud computing) са типичен пример за разпределени системи.

Много важна част на облачните системи (cloud computing) е маршрутизирането на всяка заявка до правилния сървър, който има възможност да отговори точно и коректно.

II.2. Среден слой (Middleware)

При разпределената системи възникват няколко допълнителни въпроса, които изискват решение:

1. Как единното приложение се разделя на клиентска и сървърна част?

2. Кои функции отиват при клиента и кои при сървъра?

3. Може ли клиент-сървър моделът да посрещане изискванията на всеки бизнес, независимо дали е голям или малък?

4. Може ли да се създаде индивидуална клиент-сървър система, която да работи например от дома?

5. Как новите устройства (лаптопи, таблети) могат да се включат в една клиент-сървър система?

Отговорът на всички тези въпроси се крие в трите градивни блока на разпределената клиент-сървър система, които се създават така, че да посрещнат широк спектър от изисквания – от малкото предприятие до огромната корпорация и се разбират добре със всички съвременни устройства.

Трите градивни блока на клиент-сървър приложенията с разпределена обработка са:

1. Клиент

2. Сървър

3. Среден слой (Middleware)

Терминът *middleware* (среден слой) в интернет приложенията се използва за описване на отделни софтуерни продукти, които служат за връзка между две приложения и предават данни между тях [8].

Използва се като в контекста на разпределените системи като софтуер, който предоставя услуги извън тези, предвидени от операционната система, за да се даде възможност на различните

компоненти на разпределената система да комуникират помежду си и за управление на данни.

Middleware софтуерът, свързва базите данни с уеб сървър. Този междинен софтуер позволява на потребителите да изискват данни от базата данни с помощта на форми, визуализирани чрез уеб браузър и дава възможност на уеб сървър да връща динамични уеб страници, променящи се според заявките на потребителя.

Middleware печели популярност в 1980-те години като решение на проблема как да се свържат новите приложения за по-старите системи, макар че този термин е бил в употреба от 1968 г. насам.

Той също така улеснява разпределената обработка, свързва няколко приложения за създаване на по-голямо приложение, обикновено през мрежата.

В сравнение с операционните системи и мрежовите услуги, средният слой осигурява допълнителна функционалност чрез приложни програмни интерфейси, за да осигури на приложенията повече възможности за:

- осигуряване чрез мрежата на взаимодействие на една с друга услуга или приложение;
- филтриране на данни, с цел да се направят използвани за публични приложения, като същевременно останат достатъчно защитени;
- осигуряване на независимост спрямо мрежовите услуги;
- надеждност и винаги на разположение;
- добавяне на допълнителни атрибути като семантика.

Middleware предлага някои уникални технологични предимства за бизнеса и индустрията. Традиционните системи за бази данни обикновено са разположени в затворени среди, където потребителите да получават достъп до системата само чрез ограничена мрежа или Интранет. С растежа на World Wide Web, потребителите могат да получат достъп до почти всяка база данни, за които те имат подходящи права за достъп от всяка точка на света. Middleware разглежда проблема на различни нива на оперативна съвместимост (interoperability) между различните структури на бази данни. Middleware софтуерът прави прозрачен достъпът до системите за управление на бази данни старите (СУБД) или приложенията чрез уеб сървър, без оглед на специфичните характеристики на базата данни.

Бизнесът използва приложенията от средния слой, за да свърже информацията от различните бази данни в предприятието, като

например заплатите, продажбите и счетоводството, или базите данни, намиращи се в няколко географски местоположения.

Разработчиците за безжични мрежи използват middleware софтуера, за да отговорят на предизвикателствата, свързани с безжичната сензорна мрежа (WSN) и WSN технологиите. Прилагането на middleware софтуер позволява на WSN разработчиците да интегрират операционни системи и хардуер с голямо разнообразие от различни приложения, които в момента са на разположение.

Електронната търговия използва middleware за подпомагане в обработката на бързи и сигурни транзакции между много различни компютърни среди.

Middleware софтуерът е елемент използван в широк спектър от предметни области, тъй като обединява ресурсите през различни мрежи или компютърни платформи.

II.3 Типове middleware софтуер

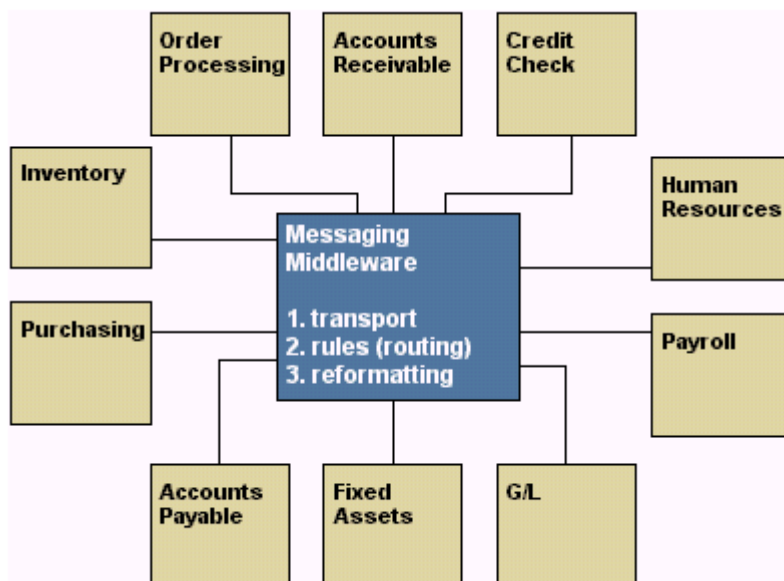
Middleware софтуерът се разработва от много десетилетия, като потребителски софтуер, но днес има разнообразна група от продукти, които предлагат в един пакет цялостни middleware решения.

II.3.1. Среден слой чрез съобщения (Messaging Middleware)

Messaging middleware софтуерът осигурява общ интерфейс и транспорт между приложенията. Ако машината, която получава съобщението е претоварена, съобщенията се организират в опашка, докато тя стане достъпна.

Messaging middleware софтуерът може да съдържа бизнес логика, която да маршрутизира съобщенията до съответните дестинации и също така да преформатира на данните, ако е необходимо.

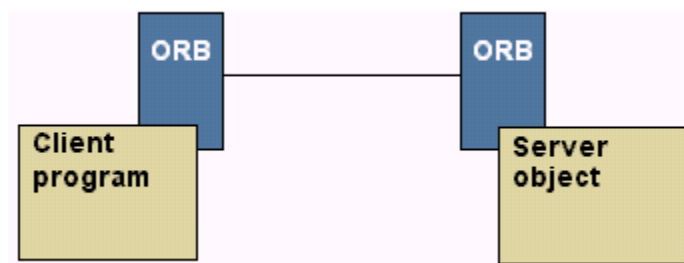
Messaging middleware софтуерът е подобен на системата за електронна поща, с изключение на това, че се използва за изпращане на данни между различните приложения.



фигура 28. Среден слой чрез съобщения (източник Computer Desktop Encyclopedia [7])

II.3.2. Среден слой чрез Object Request Broker

В обектно-ориентирани системи, чрез използването на Object Request Broker middleware, приложенията могат изпращат обекти и заявки за услуги.



фигура 29. Среден слой чрез Object Request Broker [източник <http://encyclopedia2.thefreedictionary.com/middleware>]

Object Request Broker е технология, която използват разпределените системи и действа като един вид телефонна централа.

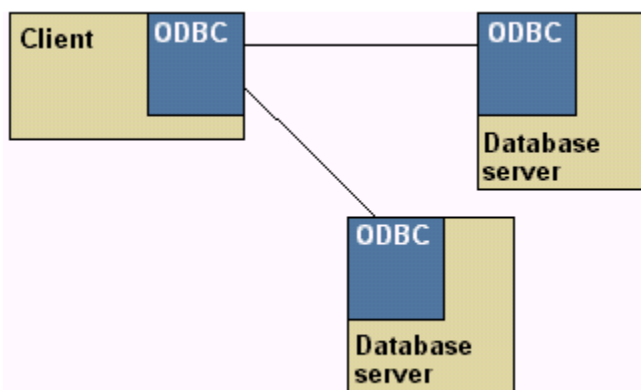
Object Request Broker дават възможност на потребителите да изградят системи, съгласявайки обекти от различни доставчици.

Системите, които използват технологията Object Request Broker са CORBA, DCOM и други.

II.3.3. Middleware софтуер за бази данни

Middleware софтуерът за бази данни позволява директен достъп до структури от данни и осигурява взаимодействието директно с бази данни, чрез разнообразни възможности за свързване. В тази категория са включени пакетите ETL (Extract, Transform, and Load).

Общите програмни интерфейси клиентите и базите данни също са в тази категория. Характерен пример е Open Database Connectivity (ODBC), който позволява на приложенията да правят стандартни обръщения към всички бази данни, които поддържат този интерфейс.



фигура 30. Общ програмен интерфейс между базите данни и клиента [източник <http://encyclopedia2.thefreedictionary.com/middleware>]

II.3.4. Middleware софтуер за транзакции

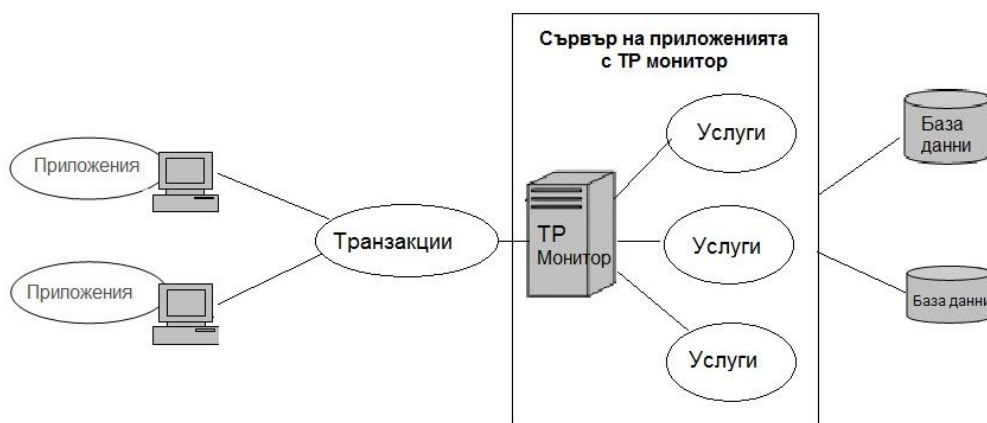
В тази категория се включва софтуерът на традиционния монитор за обработка на транзакции (TPM), който обикновено е част от сървъра на приложенията. Това е първият тип софтуер, който е наречен Middleware.

Транзакцията е единица работа, която се изпълнява точно един път и води до трайни резултати.

Обработката на транзакции е предназначена да поддържа базата данни (и някои модерни файлови системи) в консистентно състояние,

като се гарантира, че всички независими операции, извършени в процеса на обработка, ще завършат успешно или ще бъдат отменени успешно, без да нарушат целостта на системата.

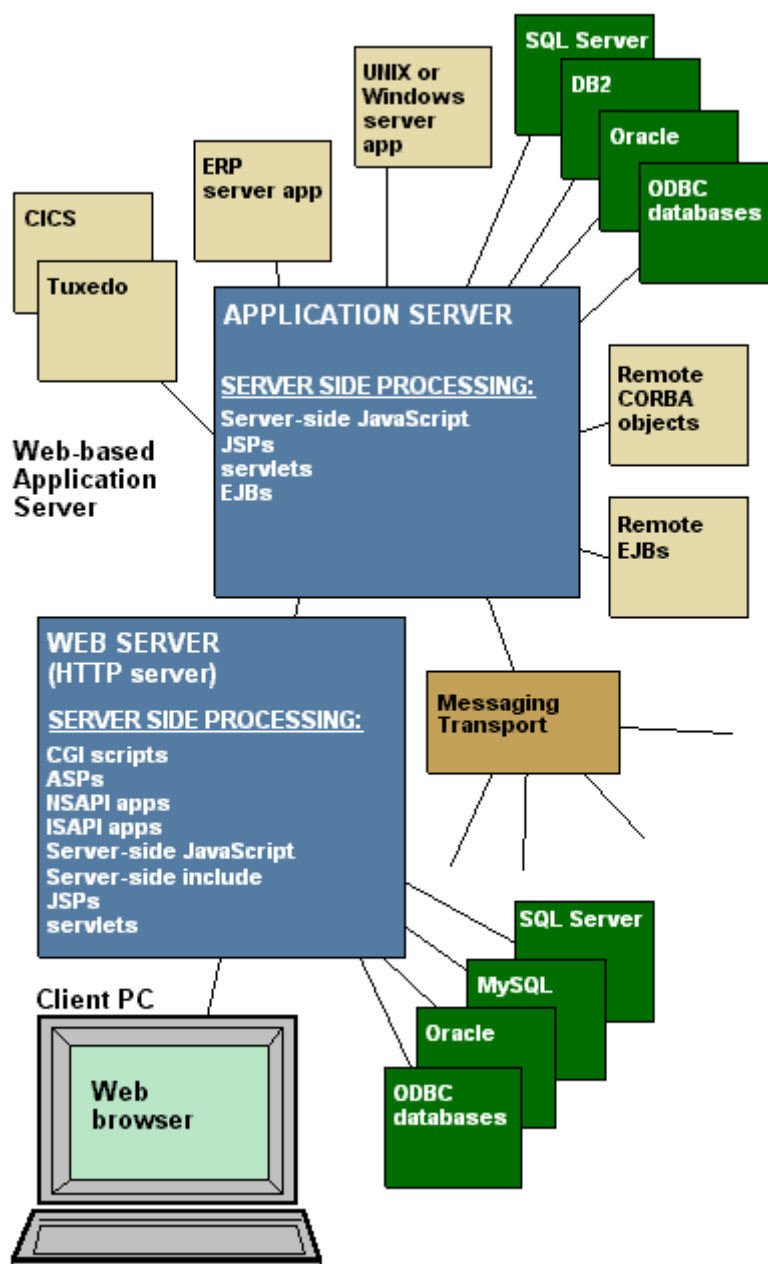
В разпределените клиент-сървър системи, TP мониторът осигурява интегритет, като се гарантира цялостта на транзакциите. Може да бъде реализиран на отделна машина и да се използва за балансиране на натоварването между клиенти и различни сървъри на приложения и сървъри на бази данни. TP монитор гарантира, че всички бази данни се актуализират от една транзакция. Работа на TP монитора е също така да осигури адекватна реакция, в случай, че транзакцията не бъде изпълнена по някаква причина – или издава съобщение за грешка, или препраща транзакцията към сървър за обработка на неизпълнени транзакции или извършва друго подходящо действие.



фигура 31. Общо представяне на middleware софтуер за транзакции

II.3.5. Middleware Сървър на приложенията

Сървърът на приложенията, когато работи като среден слой, осигурява интерфейса между браузъра и широк спектър от приложения. Браузърът може да се използва в настолните компютри или лаптопите и другите съвременни устройства като таблети и мобилни устройства.



фигура 32. Middleware Сървър на приложенията [източник <http://encyclopedia2.thefreedictionary.com/middleware>]

II.3.6. Network Logon Middleware

Middleware софтуерът за мрежи включва общ подход за идентифициране на потребителите и мрежовите ресурси, оторизация и автентификация на потребителите и създаване на стандартизирани схеми тип директория. Използването на middleware софтуер разрешава проблемите, които възникват, когато приложенията са отговорни за тези задачи или пък възниква несъвместимост поради различни версии на софтуерните компоненти.



фигура 33. Network Logon Middleware

Използвани литературни източници:

1. Стандарт ANSI/IEEE Std 1471-2000 "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems".
2. Satrom, Brandon (2011, 05 17). Application Development with HTML5 . USA. Retrieved from:
<http://channel9.msdn.com/Events/TechEd/NorthAmerica/2011/DEV343>
3. The elements of HTML. HTML Standart:
<http://www.whatwg.org/specs/web-apps/current-work/multipage/>
4. XHTML 1.0 The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0 (2002)
<http://www.w3.org/TR/xhtml1/#diffs>
5. Херш Базин. Microsoft ASP.NET – професионални проекти. 2003.
6. <http://uanet.eu/ebooks/Operating%20Systems/linux/linuxnag-bg.pdf>
7. Computer DesktopEnciclopedia
<http://www.computerlanguage.com/>
8. McGraw-Hill Dictionary of Scientific & Technical Terms, 6E, 2003. McGraw-Hill Companies, Inc.
9. Briman, Kenneth. Guide to Reliable Distributed Systems. Springer. 2012