Contents

[1. Трислойни модели Model-View-Controller и Model-View-ViewModel. 1](#_Toc130752352)

[2. Рефакториранe 3](#_Toc130752353)

[3. Видове тествания на кода (компонентен, интеграционен, регресионен и системен). 3](#_Toc130752354)

[4. Писане на SQL заявки чрез език за програмиране. 3](#_Toc130752355)

[5. Използване на системи за обектно-релационно свързване (ORM frameworks) – разлики в подходите Code First и Database First. 4](#_Toc130752356)

[6. Пакети, пакетни мениджъри и тяхната употреба 4](#_Toc130752357)

**Изпитна тема № 7: Разработка на софтуер**

Трислойни модели Model-View-Controller и Model-View-ViewModel – графично представяне, начин на изпълнение. Рефакториране на код. Видове тествания на кода (компонентен, интеграционен, регресионен и системен). Писане на SQL заявки чрез език за програмиране. Използване на системи за обектно-релационно свързване (ORM frameworks) – разлики в подходите Code First и Database First. Разработка на приложение с ORM framework. Принципи за правилно разделяне на компонентите на приложението по слоеве. Пакети, пакетни мениджъри и тяхната употреба.

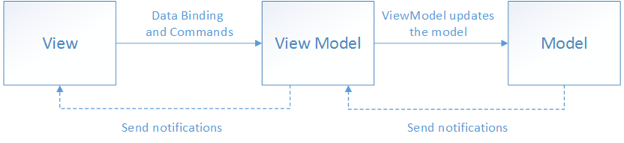
# Трислойни модели Model-View-Controller и Model-View-ViewModel.

* **Tрислойния модел -** Разпределя приложението на слоеве​. Всеки слой има строго определена задача​.
* **Model-View-Controller -** е щаблон за дизайн (Design Pattern) с три свързани един с друг компонента по следната схема:



* Модел (Model) – управлява данните, изпълнява заявки и команди върху данните и не позволява данните да се манипулират от презентационния слой
* Изглед (View) презентационен слой – представя данните от модела на потребителя и предоставя възможност за въвеждане на данни от потребителите
* Контролер (Controller) – преобразува входните данни в команди и ги изпраща към Изгледа или към модела​. Т.е отговаря за обработка на данни, достъпване на слоя за данни и подготовка на данни за презентационния слой
* **Model-View-View-Model –** Модел за изглед на модел, е добре подреден и повторно използван начин за организиране на кода. Основните компоненти на MVVM са модел, изглед и модел. На първо място, моделът държи данните. Второ, view model работи като връзка или връзка между модела и изгледа. Той преобразува обектите за данни от модела по такъв начин, че обектите лесно се управляват и представят. И накрая, изгледът съдържа потребителските интерфейси.

Схема на модела:



# Рефакториранe

Рефакториране (нарицателно) – промяна във вътрешната структура на софтуера, въведена с цел да се улесни разбирането на кода и да се направи по-лесно въвеждането на нови промени, но без да се променя наблюдаваното поведение

* Запазва се работеща версия на проекта
* Проверява се текущото поведение ( по възможност с автоматизирани тестове)
* Рефакториране
* Проверява се запазено ли е първоначалното поведение

# Видове тествания на кода (компонентен, интеграционен, регресионен и системен).

**Софтуерното тестване** е процес на изследване и проучване на софтуер с цел получаване на информация за качеството на продукта и услугата, която се изпитва.

* Компонентното тестване е процес в програмирането, чрез който се тестват отделни единици (компоненти) от сорс код – един или повече програмни модула и техните контролни данни, процедури за използване, оперативни процедури с цел да се установи дали работят правилно.
* Тестовете на ниво интеграция имат за цел да проверя взаимовръзките и съвместната работа на отделните компоненти, когато са поставени в една среда. Компонентите може да се интегрират на вълни или накуп.
* Регресионното тестване означава да се тества отново вече тествана част от системата след промяна по някоя нейна функционалност.
* Тестовете на системно ниво проверяват дали системата като цяло отговаря на изискванията.

# Писане на SQL заявки чрез език за програмиране.

При свързания модел за достъп до данни се използват обекти от следните класове:

**SqlConnection** - Създава връзката към БД на SQL Server. Изрично отваряне и затваряне на връзката с **Open()** и **Close()** методи

**SqlCommand** - Изпълнява SQL команди върху SQL Server чрез създадената връзка. Може да приема параметри (SQLParameter), който съсдржа заявката към БД. По-важни методи

* + **ExecuteScalar() -** Връща единствена стойност (стойността на първата колона на първия ред от множеството от резултати) (като **System.Object**)
  + **ExecuteReader() -** Връща **SqlDataReader.** Това е курсор върху върнатите записи (резултатно множество - result set)
  + **ExecuteNonQuery()**- Връща броя на засегнатите редове (**int**)

**SqlDataReader** - Извлича данни (множество от записи) от SQL Server като резултат от изпълнението на SQL заявка. Важни свойства и методи:

* + **Read()** – премества курсора напред, като връща false, в случай че няма следващ запис
  + **Indexer[]** – връща стойността на текущия запис по дадено име на колона или индекс
  + **Close()** – затваря курсора и освобождава ресурсите

# Използване на системи за обектно-релационно свързване (ORM frameworks) – разлики в подходите Code First и Database First.

ORM - Обектно-релационно съответствие /Object-Relational Mapping/ позволява да манипулираме бази данни използвайки общи класове и обекти

Таблици в базата данни è C#/Java/etc. класове

Има следните функционалности:

* + **Автоматично генерира SQL за извършване на операции върху данните**
  + **Създаване на обектен модел по схемата на БД (DB First модел)**
  + **Създаване на схемата на БД от обектен модел (Code First модел)**
  + **Извличане на информация обектно-ориентирано API (например LINQ заявки)**

Подходът **Database First** моделира класовете с oбекти след като базата данни е създадена. При този подход Entity Data Model (EDM) се създава от основната база данни. Ръчните промени в базата данни са възможни лесно и винаги можете да актуализирате EDM, ако е необходимо (например, ако схемата на основната база данни се промени).

Подходът **Code First** създава съответните таблици в БД следвайки тяхно описание чрез класове. Този подходът дава повече контрол над кода - вече не е необходимо да се работи с автоматично генериран код.

# Пакети, пакетни мениджъри и тяхната употреба

**Пакети** са готови комплекти от програми, които са безплатни пакети и с отворен код.

**NuGet пакетния мениджър** е част от Visual Studio и предоставя функционалност за:

* Лесно инсталиране и обновяване на библиотеки и инструменти с отворен код
* Конфигуруеми източници на пакети
* Лесно като добавянето на референция
* Инсталатор с графичен интерфейс
* Налична конзола за управление на пакетите