Тема 7

1. MVC и MVVM са две трислойни архитектурни модели, които се използват за разработка на софтуерни приложения.

MVC стой за “Model-View-Controller":

Model - представя данните и функционалността, свързана с тях. Той може да включва различни видове данни като информация за потребители, конфигурационни данни, данни за продукти и други.

View - отговаря за визуализацията на данните, с които работи потребителят. Изгледът може да бъде визуален елемент като екранна форма, таблица, списък или друг.

Controller - свързва модела и изгледа, управлява данните, извлича информация от модела и я предоставя на изгледа за визуализация. Контролерът също така реагира на действията на потребителя и извършва нужните операции върху модела.

Графично представено ще е нещо като

M <->V<->C

Изпълнението на MVC модела е такова, че всеки от компонентите е изолиран от другите два. Това позволява лесно поддръжка и разширение на системата, защото промени в единия компонент не влияят върху другите два.

MVVM(Model-View-ViewModel) е самият модел:

Model - представя данните и функционалността, свързана с тях.

View - отговаря за визуализацията на данните, с които работи потребителят.

View Model - слой между модела и изгледа, който съдържа логиката за визуализацията и обработката на данните, която не би трябвало да е част от модела или изглед

1. Когато работим с готов примерен код, може да има различни причини да го обясним или да го рефакторираме. Например, може да искаме да разберем как работи даден модул или да добавим нова функционалност към системата. Рефакторирането на кода може да бъде извършено, за да се подобри качеството на кода, да се направи по-четим, по-лесен за поддръжка, да се намалят грешките или да се добавят нови функционалности.

Когато обясняваме готов примерен код, е важно да обърнем внимание на следните елементи:

**Целта на кода** - трябва да разберем за какво служи кодът и каква е целта му. Това може да ни помогне да разберем как да го използваме правилно и как да го модифицираме адекватно.

**Архитектурата** - трябва да разберем как е проектиран кодът и как е организиран в модули или компоненти. Това може да ни помогне да разберем лесно как са свързани различните части на кода и да ги модифицираме без да счупим логиката на приложението.

**Използвани технологии и библиотеки** - трябва да разберем какви технологии и библиотеки са използвани в кода и как да ги използваме правилно. Това може да ни помогне да разберем как да интегрираме кода в нашата система или как да го подобрим като използваме същите технологии и библиотеки.

Когато рефакторираме готов примерен код, можем да използваме различни подходи, за да подобрим качеството на кода. Ето някои от тях:

**Премахване на дублирания код** - можем да намалим сложността на кода, като премахнем дублиращия се код и го заменим с повтарящи се функции.

**Използване на подходящи дизайн патерни** - можем да използваме добре познати дизайн патерни, за да подобрим архитектурата на системата.

1. Тестванията са важна част от процеса на софтуерната разработка, като имат за цел да проверят функционалността и качеството на кода. Видовете тествания могат да се разделят в няколко категории:

**Unit тествания** - това са тествания на отделни компоненти или модули на кода, за да се уверим, че те работят правилно. Unit тестванията са обикновено автоматизирани и се изпълняват на локалното устройство на разработчика.

**Интеграционни тествания** - това са тествания на интеграцията между различни компоненти или модули на кода, за да се уверим, че те работят правилно, когато са свързани. Интеграционните тествания обикновено се изпълняват на цялата система и могат да изискват инсталацията на специален софтуер.

**Приемателни тествания** - това са тествания на цялата система, които се изпълняват от крайните потребители или потребителски групи, за да се уверят, че приложението отговаря на изискванията им и работи правилно.

**Функционални тествания** - това са тествания, които се фокусират върху функционалността на системата и проверяват дали приложението работи правилно спрямо дадени спецификации.

**Тествания на производителността** - това са тествания, които се изпълняват за да се установи какво е максималното натоварване на системата и колко време отнема за приложението да извърши дадени операции при определени обстоятелства.

**Тествания на сигурността** - това са тествания, които се изпълняват, за да се убедим, че системата е сигурна и не може да бъде проникната от злонамерени хакери или зловреден софтуер.

**Тествания на издръжливостта** - това са тествания, които се изпълняват, за да се установи как системата ще се справи с грешки или повреди в хардуера или мрежата.

1. **Code-first** и **Database-first** са два различни подхода за генериране на модели на данни и техните мапинги към релационни бази от данни в .NET приложения.

**Code-first** е подход, при който програмистът първо дефинира моделите на данните в кода си, използвайки класове и атрибути, след което Entity Framework автоматично генерира базата данни и мапингите към нея. Този подход осигурява по-голяма гъвкавост и удобство, тъй като програмистът може да създава моделите на данните в своето предпочитано програмно средство (IDE), като използва .NET синтаксис, и след това да избере метод за генериране на базата данни. Това може да бъде полезно, особено ако проектът има нужда от често обновяване на базата данни или се изисква лесна преносимост на моделите на данните между различни бази данни.

**Database-first** е подход, при който базата данни се генерира първо и след това Entity Framework автоматично генерира моделите на данните и техните мапинги към базата данни. Този подход осигурява по-бързо създаване на моделите на данните и мапингите им, тъй като програмистът може да използва инструменти за визуално моделиране на бази данни, като например SQL Server Management Studio, за да създаде базата данни и след това да използва Entity Framework, за да генерира моделите на данните и техните мапинги. Това може да бъде полезно, особено ако проектът използва вече съществуваща база данни или има сложни взаимодействия с други системи.

Общо казано, **Code-first** е по-гъвкав и позволява по-голяма свобода при създаване на модели на данните, докато **Database-first** е по-бърз и подходящ за проекти, които използват вече съществуваща база данни.

1. <https://github.com/stankopk/InsuranceManager>
2. **ORM (Object-Relational Mapping)** е техника за свързване на обекти в програмен код с релационни бази данни. ORM инструментите позволяват на програмистите да работят с бази данни като с обекти в програмен код, като избягват необходимостта да пишат SQL заявки ръчно. Възможно е да има множество видове ORM инструменти, но ето няколко от тях:

**ActiveRecord ORM**

**Data Mapper ORM**

**Code First ORM**

**Entity Framework ORM**

Основната разлика между ORM подхода и SQL заявките е, че ORM позволява на програмистите да работят с базите данни като с обекти в програмен код, докато SQL заявките изискват програмистите да пишат SQL заявки ръчно. ORM позволява на програмистите да се фокусират върху модела на данните, като предоставя абстракция между модела на данните и базата данни, като автоматично генерира SQL заявки. Освен това ORM позволява на програмистите да избягват определени грешки, свързани с писането на SQL заявки ръчно, като например SQL инжекции.

1. В контекста на програмирането, понятията "пакет" и "пакетен мениджър" се използват за да се опише начинът на организиране и управление на програмен код и библиотеки, които са необходими за разработка на софтуерни приложения.

**"Пакет"** е колекция от свързани файлове и код, които обикновено представляват библиотеки или модули, съдържащи функционалност, която може да бъде използвана от други програмисти. Пакетите обикновено имат име и версия, която позволява на разработчиците да указват точната версия на пакета, която да се използва в тяхното приложение.

**"Пакетен мениджър"** е инструмент за управление на пакетите. Той обикновено позволява на разработчиците да инсталират, актуализират и премахват пакети от тяхната система. Пакетните мениджъри често също така имат функционалност за управление на зависимости, което позволява на програмистите да инсталират нужните пакети за тяхното приложение и да се уверят, че те са съвместими един с друг.

Разликата между пакетите и пакетните мениджъри е, че пакетите са просто колекции от код и библиотеки, докато пакетните мениджъри са инструменти за управление на тези пакети. Това означава, че пакетите могат да бъдат използвани без да се използва пакетен мениджър, но пакетният мениджър улеснява процеса на управление на пакетите и ги прави по-лесно достъпни за други разработчици.

Съществуват множество пакетни мениджъри в зависимост от конкретните нужди и технологиите, които се използват в различните програмни езици и операционни системи. Ето някои от най-популярните пакетни мениджъри и техните основни характеристики:

**npm (Node Package Manager)** - пакетен мениджър за Node.js. Позволява инсталиране на библиотеки и модули от обществен репозиториум или от локален файлов систем.

**NuGet** - пакетен мениджър за .NET платформата. Позволява инсталиране на библиотеки и модули от обществен репозиториум или от локален файлов систем.

**Composer** - пакетен мениджър за PHP. Позволява инсталиране на библиотеки и модули от обществен репозиториум или от локален файлов систем.

**pip** - пакетен мениджър за Python. Позволява инсталиране на библиотеки и модули от обществен репозиториум или от локален файлов систем.

**Homebrew** - пакетен мениджър за macOS. Позволява инсталиране на софтуер от обществен репозиториум или от локален файлов систем.

**Chocolatey** - пакетен мениджър за Windows. Позволява инсталиране на софтуер от обществен репозиториум или от локален файлов систем.

**apt-get** - пакетен мениджър за Linux системи, базирани на Debian. Позволява инсталиране на софтуер от обществен репозиториум или от локален файлов систем.

Тези пакетни мениджъри обикновено имат много подобни функционалности, но също така имат специфични функции за конкретния език или операционна система, за които са предназначени. Важно е да се избере правилния пакетен мениджър, който ще отговаря на нуждите на конкретния проект и ще улесни управлението на пакетите и зависимостите.

За пример на инсталация на пакетен мениджър ще използваме NuGet Package Manager. Обикновено при инсталацията на Visual Studio той е автоматично инсталиран, но в случай че не, ще трябва ръчно да го инсталирате.

**Отваряме Visual Studio Installer**

**Във Visual Studio Installer избираме версията на Visual Studio, която използваме и цъкаме на бутона “Modify”**

**От там отиваме на таб “Individual Components” и търсим**

**“NuGet Package Manager” и го удряме с тикче за да се инсталира.**

**Когато обаче става дума за инсталиране на пакети става по следния начин:**

**Кликаме с дясно копче на проекта в Solution Explorer;**

**След това на dropdown менюто избираме “Manage NuGet packages”;**

**В търсачката пишем името на пакета, който искаме да изтеглим и го инсталираме.**