**Тема 7**

1. MVC и MVV

M са две различни трислойни архитектури, използвани в софтуерното инженерство за разделяне на кода на софтуерното приложение на три отделни компонента: модел, изглед и контролер (за MVC) или модел, изглед и изглед-модел (за MVVM).

Моделът представлява данните и логиката на приложението. Изгледът представлява визуалното представяне на тези данни. Контролерът (за MVC) или изглед-моделът (за MVVM) служат за свързване на модела и изгледа.

Дефиниране на трислойни модели (MVC и MVVM). Представя ги графично и

Различава начините на изпълнението им

MVC и MVVM са две различни трислойни архитектури, използвани в софтуерното инженерство за разделяне на кода на софтуерното приложение на три отделни компонента: модел, изглед и контролер (за MVC) или модел, изглед и изглед-модел (за MVVM).

Моделът представлява данните и логиката на приложението. Изгледът представлява визуалното представяне на тези данни. Контролерът (за MVC) или изглед-моделът (за MVVM) служат за свързване на модела и изгледа.

Изпълнението на MVC и MVVM е различно. В MVC, контролерът служи за обработка на входни данни от потребителя и променя модела спрямо тези данни. Изгледът се обновява автоматично, когато моделът се промени.

В MVVM, изглед-моделът служи за извличане на данни от модела и трансформирането им в такъв вид, който може да бъде показан в изгледа. Потребителският вход отново се обработва от изглед-модела. Изгледът е свързан към изглед-модела и автоматично се обновява при промяна в изглед-модела.

Така че, в крайна сметка, каква трислойна модел да използвате зависи от конкретните нужди на вашия проект.

Дефиниране на трислойни модели (MVC и MVVM). Представя ги графично и

Различава начините на изпълнението им

MVC и MVVM са две различни трислойни архитектури, използвани в софтуерното инженерство за разделяне на кода на софтуерното приложение на три отделни компонента: модел, изглед и контролер (за MVC) или модел, изглед и изглед-модел (за MVVM).

Моделът представлява данните и логиката на приложението. Изгледът представлява визуалното представяне на тези данни. Контролерът (за MVC) или изглед-моделът (за MVVM) служат за свързване на модела и изгледа.

Ето графичното представяне на тези две трислойни модели:

Изпълнението на MVC и MVVM е различно. В MVC, контролерът служи за обработка на входни данни от потребителя и променя модела спрямо тези данни. Изгледът се обновява автоматично, когато моделът се промени.

В MVVM, изглед-моделът служи за извличане на данни от модела и трансформирането им в такъв вид, който може да бъде показан в изгледа. Потребителският вход отново се обработва от изглед-модела. Изгледът е свързан към изглед-модела и автоматично се обновява при промяна в изглед-модела.

Така че, в крайна сметка, каква трислойна модел да използвате зависи от конкретните нужди на вашия проект.

Можем да разгледаме по-подробно всяка от трислойните модели:

MVC (Model-View-Controller) е архитектурен модел, който се използва за разделяне на приложението на три отделни компонента.

Моделът (Model) представлява данните и логиката на приложението. Това може да бъде база данни, XML файлове или дори класове, които съхраняват информация и извършват операции с нея.

Изгледът (View) е визуалното представяне на тези данни. Това може да бъде графичен интерфейс, HTML страница или дори текстов файл, който съдържа форматирана информация.

Контролерът (Controller) свързва модела и изгледа. Той приема входни данни от потребителя и ги превръща в команди за модела или изгледа. Той също така се грижи за обновяването на изгледа, когато моделът се промени.

MVC е особено полезен за големи приложения, където е важно да се поддържа ясно разделение между различните компоненти и да се осигури лесна поддръжка на кода.

MVVM (Model-View-ViewModel) е архитектурен модел, който също се използва за разделяне на приложението на три отделни компонента.

Моделът (Model) отново представлява данните и логиката на приложението.

Изгледът (View) е визуалното представяне на тези данни, но този път изгледът е свързан директно с изглед-модела (ViewModel).

Изглед-моделът (ViewModel) служи за извличане на данни от модела и трансформирането им в такъв вид, който може да бъде показан в изгледа. Потребителският вход се обработва от изглед-модела и се прехвърля към модела за обработка.

2.

За да рефакторирате кода, първо трябва да го прегледате и да идентифицирате проблемните области, които бихте искали да подобрите. Например, може да има дублиран код, сложни и непотребни конструкции или грешки в имплементацията.

След като сте идентифицирали проблемите, можете да започнете да рефакторирате кода, като използвате подходящи техники, като например извличане на методи, обединяване на дублиращ се код или използване на по-чист и разбираем синтаксис.

Преди да започнете да рефакторирате кода, е важно да направите резервно копие на оригиналния файл, за да можете да се върнете към него, ако се появят проблеми. Също така, трябва да тествате новия код, за да се уверите, че работи правилно и не се появяват нови проблеми.

За да рефакторирате кода, първо трябва да го прегледате и да идентифицирате проблемните области, които бихте искали да подобрите. Например, може да има дублиран код, сложни и непотребни конструкции или грешки в имплементацията.

След като сте идентифицирали проблемите, можете да започнете да рефакторирате кода, като използвате подходящи техники, като например извличане на методи, обединяване на дублиращ се код или използване на по-чист и разбираем синтаксис.

Преди да започнете да рефакторирате кода, е важно да направите резервно копие на оригиналния файл, за да можете да се върнете към него, ако се появят проблеми. Също така, трябва да тествате новия код, за да се уверите, че работи правилно и не се появяват нови проблеми.

Моля, изпратете ми кода и кажете ми какво бихте искали да постигнете чрез рефакторирането, за да мога да ви предложа конкретни съвети и примери за рефакториране.

3.В общи линии, тестванията могат да се разделят на две главни категории: функционални и нефункционални тествания.

Функционалните тествания проверяват дали софтуерът изпълнява очакваното поведение. Те могат да бъдат разделени на следните подкатегории:

Unit тестове – тествания на отделните компоненти или функции на софтуерното приложение, за да се уверим, че те работят коректно.

Integration тестове – тествания на различните компоненти на софтуерното приложение, за да се уверим, че те работят заедно коректно.

System тестове – тествания на цялото софтуерно приложение, за да се уверим, че то изпълнява всички изисквания и функционалности, които са му поставени.

Acceptance тестове – тествания, които се извършват от потребителите на софтуерното приложение, за да се уверят, че то отговаря на техните нужди и изисквания.

Нефункционалните тествания проверяват качествата на софтуерното приложение, като изпълнение, сигурност, надеждност и други. Те могат да бъдат разделени на следните подкатегории:

Performance тестове – тествания на производителността на софтуерното приложение, за да се уверим, че то може да обработва големи обеми от данни и заявки при зададени условия.

Security тестове – тествания на сигурността на софтуерното приложение, за да се уверим, че то е защитено от възможни атаки и злоупотреби.

Usability тестове – тествания на удобството и лесното използване на софтуерното приложение от потребителите.

Compatibility тестове – тествания на съвместимостта на софтуерното приложение с различни операционни системи, хардуерни компоненти и софтуерни инструменти.

Regression тестове – тествания на софтуерното приложение, след като са направени промени, за да се уверим, че няма неблагоприятни ефекти върху вече тестваните функционалности.

4. Comparison of Code First and Database First

Сравнява Code First и Database First подхода и прави изводи за разликите

Между тях. Описва процеса на разработка, следвайки подходите.

Code First и Database First са два различни подхода за разработка на софтуерни приложения, които изискват своеобразни начини за работа с базата данни. В този отговор ще сравним двете подхода и ще дадем изводи за разликите между тях.

Code First е подход за разработка, който се фокусира върху създаването на модели на данни в програмен код, като програмистът първо дефинира моделите на данни и след това се генерират базите данни, които да съответстват на тези модели. С други думи, програмистът пише код за моделите на данни, а след това инструментите за Code First генерират базите данни.

Database First е подход за разработка, който се фокусира върху генерирането на модели на данни от базата данни. С други думи, програмистът първо генерира моделите на данни от базата данни, а след това тези модели се използват за създаване на програмния код.

Процесът на разработка при Code First подхода обикновено изглежда така:

Дефиниране на класовете и свойствата, които ще бъдат използвани като модели на данни.

Конфигуриране на моделите на данни с помощта на анотации или конфигурационни файлове.

Генериране на базата данни от моделите на данни.

Писане на код, който използва генерираната база данни.

Процесът на разработка при Database First подхода обикновено изглежда така:

Генериране на моделите на данни от базата данни.

Писане на код, който използва генерираните модели на данни.

Разликите между двата подхода могат да бъдат сведени до това, кой от тях е най-подходящ в даден контекст. Code First подходът е по-подходящ, когато има нужда от гъвкавост и бързина в разработката. Той позволява програмистът да създава модели на данни по свой вкус, което прави процеса на разработка по-гъвкав и ефективен.

5.

Когато създавате приложение, което работи с база от данни, има два основни начина за начина на проектиране на базата – Code First и Database First.

Code First означава, че първо създавате моделите на данните във вашето приложение и след това базата от данни се генерира автоматично, когато приложението се стартира за първи път. Този начин на проектиране на базата от данни е по-модерен и по-гъвкав, тъй като позволява бързо и лесно да се променят моделите на данните, като базата от данни се обновява автоматично.

Database First означава, че първо проектирате базата от данни, а след това генерирате моделите на данните във вашето приложение. Този начин на проектиране на базата от данни е по-традиционен и е подходящ, когато имате вече съществуваща база от данни или когато искате да използвате съществуваща база от данни.

И двата начина могат да бъдат използвани за разработка на част от приложение, което да работи с база от данни. При използване на Code First, трябва да създадете моделите на данните във вашето приложение и да ги мапнете към базата от данни. Това може да стане посредством Entity Framework, който позволява да се изградят модели на данни и да се мапнат към база от данни. При използване на Database First, трябва да създадете базата от данни и да я мапнете към моделите на данните във вашето приложение. Това може да стане с помощта на Entity Framework или с други инструменти за ORM (Object-Relational Mapping).

Изборът между Code First и Database First зависи от вашите нужди и предпочитания, като и двата начина имат своите предимства и недостатъци. Преди да започнете да разработвате приложението си, трябва да проучите какъв подход е най-подходящ за вас и вашите нужди.

6.ORM (Object-Relational Mapping) е технология, която позволява на програмистите да работят с релационни бази данни, използвайки обекти от програмния им код вместо да пишат ръчно SQL заявки. Това улеснява разработката и поддръжката на софтуерните приложения, като намалява възможността за грешки, свързани с изпълнението на SQL заявки.

Една от основните разлики между ORM и SQL заявките е начинът на изразяване на заявките. В SQL, програмистът трябва да изрази заявката във вид на низ, който да се изпълни от базата данни. В ORM, програмистът работи с обекти и методи, които са част от ORM библиотеката. Тези методи се използват за извличане на данни от базата данни и преобразуването им в обекти, с които да се работи в програмния код.ORM подходът има няколко предимства пред SQL заявките. Първо, кодът е по-четим и по-лесен за поддръжка. Вместо да пишете дълги SQL заявки, програмистът може да използва методи на ORM библиотеката, които са добре документирани и лесни за разбиране. Второ, ORM позволява на програмистите да избегнат много от проблемите, свързани със съставянето на SQL заявки, като SQL injection атаки и конфликти.

ORM за работа с SQL.

Демонстрира работа с ORM технология и диференцира SQL заявките от ORM

Подхода

ORM (Object-Relational Mapping) е технология, която позволява на програмистите да работят с релационни бази данни, използвайки обекти от програмния им код вместо да пишат ръчно SQL заявки. Това улеснява разработката и поддръжката на софтуерните приложения, като намалява възможността за грешки, свързани с изпълнението на SQL заявки.

Една от основните разлики между ORM и SQL заявките е начинът на изразяване на заявките. В SQL, програмистът трябва да изрази заявката във вид на низ, който да се изпълни от базата данни. В ORM, програмистът работи с обекти и методи, които са част от ORM библиотеката. Тези методи се използват за извличане на данни от базата данни и преобразуването им в обекти, с които да се работи в програмния код.

Например, в SQL заявките, програмистът може да напише следната заявка за извличане на всички потребители, които имат име „John“:

ORM подходът има няколко предимства пред SQL заявките. Първо, кодът е по-четим и по-лесен за поддръжка. Вместо да пишете дълги SQL заявки, програмистът може да използва методи на ORM библиотеката, които са добре документирани и лесни за разбиране.

ORM технологиите предлагат редица други функционалности, които могат да бъдат полезни за програмистите при работа с бази данни. Например, ORM библиотеките позволяват на програмистите да дефинират модели за таблиците в базата данни, като тези модели отразяват структурата на данните, с които работят. Това улеснява създаването на нови записи в базата данни и търсенето на съществуващи записи, като гарантира, че програмистите използват правилните полета и типове данни.

ORM библиотеките също така могат да предоставят вградени механизми за валидация на данните, които се записват в базата данни. Това може да включва проверка на входните данни за невалидни стойности, като например празни стойности или стойности извън определен диапазон. Тези механизми за валидация могат да помогнат на програмистите да избегнат много от проблемите, свързани с невалидни данни в базата данни, като например сбъркани стойности или конфликти между различни записи в базата данни.

Въпреки това, ORM технологиите не са подходящи за всички видове задачи, свързани с работа с бази данни. Например, при работа с големи и сложни бази данни, може да бъде по-ефективно да се използват направо SQL заявки, за да се избегне излишното забавяне, свързано с използването на ORM. Освен това, при работа с бази данни, които не са релационни (например NoSQL бази данни), ORM технологиите могат да бъдат по-малко ефективни или дори невъзможни за използване.