بخش 1

Uml

.1 تعریف:

UML یک زبان مدل‌سازی استاندارد است که برای طراحی و نمایش سیستم‌های نرم‌افزاری استفاده می‌شود. این زبان، مجموعه‌ای از نمادها و نمودارها را ارائه می‌دهد که به توصیف ساختار، رفتار و معماری سیستم‌های پیچیده می‌پردازند. UML توسط توسعه‌دهندگان، تحلیل‌گران و تیم‌های طراحی برای ایجاد یک دیدگاه مشترک در مورد سیستم به کار می‌رود.

.2 فهرست نمودارها:

UML شامل نمودارهای متنوعی است که به دو دسته‌ی اصلی تقسیم می‌شوند: نمودارهای ساختاری (Static):

Class Diagram: توصیف کلاس‌ها، ویژگی‌ها و روابط آن‌ها.

Object Diagram: نمایش نمونه‌های خاص از کلاس‌ها و روابط در یک زمان مشخص.

Component Diagram: نمایش اجزای فیزیکی سیستم.

Deployment Diagram: نمایش استقرار اجزا روی زیرساخت‌های فیزیکی.

نمودارهای رفتاری (Dynamic):

Use Case Diagram: نمایش تعاملات کاربران با سیستم.

Sequence Diagram: نشان‌دهنده‌ی ترتیب پیام‌ها بین اشیاء.

Activity Diagram: نمایش جریان کارها و فعالیت‌ها در سیستم.

State Diagram: توصیف وضعیت‌های مختلف اشیاء در طول زمان.

Communication Diagram: نمایش تعاملات بین اشیاء.

.3 کاربرد:

UML برای تحلیل، طراحی و مستندسازی سیستم‌های نرم‌افزاری و همچنین ارتباط بین اعضای تیم توسعه به کار می‌رود. کاربردهای اصلی شامل موارد زیر است:

طراحی معماری نرم‌افزار.

شناسایی تعاملات و نیازهای کاربران.

پیش‌بینی مشکلات و بهینه‌سازی سیستم.

ایجاد مستندات جامع برای استفاده در مراحل توسعه و نگهداری.

.4 مزایا:

استانداردسازی: UML به‌عنوان یک زبان جهانی، امکان ارتباط بهتر بین تیم‌های مختلف را فراهم می‌کند.

وضوح: ارائه دیدگاه شفاف و قابل‌درک از سیستم.

انعطاف‌پذیری: قابل استفاده برای سیستم‌های کوچک و بزرگ.

پشتیبانی ابزارها: اکثر نرم‌افزارهای طراحی مهندسی از UML پشتیبانی می‌کنند.

.5 معایب:

یادگیری دشوار: برای تازه‌کاران ممکن است پیچیده باشد.

زمان‌بر بودن: طراحی نمودارهای UML در پروژه‌های کوچک ممکن است زمان اضافی نیاز داشته باشد.

محدودیت مدل‌سازی: برخی جنبه‌های خاص سیستم ممکن است نیازمند ابزارهای اضافی باشند.

.6 جنبه‌های ایستا و پویا:

ایستا: نمودارهایی مانند Class Diagram و Component Diagram ساختار ثابت سیستم را نمایش می‌دهند.

پویا: نمودارهایی مثل Sequence Diagram یا Activity Diagram رفتار و جریان فعالیت‌ها را در

مودار های زبان UML را همراه با تعریف و ذکر مثال به همراه توضیحات

:

**1. Use Case Diagram**

Use Case Diagram برای نشان دادن تعاملات بین کاربران (Actors) و موارد استفاده (Use Cases) در سیستم استفاده می‌شود. هدف آن ارائه‌ی دید کلی از عملکرد سیستم است.

* **جزئیات:**
  + Actors می‌توانند انسان‌ها یا سیستم‌های دیگر باشند.
  + Use Cases نشان‌دهنده‌ی وظایف یا عملکردهایی هستند که سیستم باید ارائه دهد.
  + روابط شامل ارتباطات و وابستگی‌ها بین Actors و Use Cases هستند.
* **کاربرد:** در تحلیل نیازمندی‌های کاربر و توصیف عملکرد سیستم.

**2. Activity Diagram**

Activity Diagram جریان کارها و ترتیب فعالیت‌ها در سیستم را نشان می‌دهد. این نمودار می‌تواند شامل شرط‌ها، حلقه‌ها و تصمیم‌گیری باشد.

* **جزئیات:**
  + فعالیت‌ها با مستطیل نشان داده می‌شوند.
  + جریان‌ها بین فعالیت‌ها با فلش‌ها مشخص می‌شوند.
  + تصمیم‌گیری‌ها با علامت لوزی نمایش داده می‌شوند.
* **کاربرد:** توصیف منطق کسب‌وکار یا مدل‌سازی الگوریتم‌ها.

**3. State Diagram**

State Diagram وضعیت‌های مختلف یک شیء و تغییرات آن در طول زمان را نشان می‌دهد. این نمودار معمولاً شامل رویدادهایی است که باعث تغییر وضعیت می‌شوند.

* **جزئیات:**
  + وضعیت‌ها با مستطیل‌های گرد نشان داده می‌شوند.
  + رویدادها باعث انتقال از یک وضعیت به وضعیت دیگر می‌شوند.
* **کاربرد:** طراحی سیستم‌هایی با وضعیت‌های چندگانه (مثلاً ماشین حالات).

**4. Communication Diagram**

Communication Diagram تعاملات بین اشیاء را با تمرکز بر پیام‌های ارسال‌شده نشان می‌دهد. این نمودار ترتیب پیام‌ها را نشان نمی‌دهد، بلکه تمرکز آن بر روابط اشیاء است.

* **جزئیات:**
  + اشیاء با مستطیل نمایش داده می‌شوند.
  + پیام‌ها با خطوط بین اشیاء و شماره‌گذاری مشخص می‌شوند.
* **کاربرد:** طراحی و مستندسازی ارتباطات در سیستم.

**5. Sequence Diagram**

Sequence Diagram ترتیب پیام‌ها و تعاملات بین اشیاء را به‌صورت زمانی نشان می‌دهد. این نمودار جزئیات بیشتری در مورد ترتیب و مدت‌زمان تعاملات ارائه می‌دهد.

* **جزئیات:**
  + خطوط عمودی نمایانگر طول عمر اشیاء هستند.
  + پیام‌ها با فلش‌ها و ترتیب زمانی مشخص می‌شوند.
* **کاربرد:** تحلیل جریان‌های زمانی در سیستم و طراحی توالی عملیات.

**6. Timing Diagram**

Timing Diagram نحوه تغییرات وضعیت‌ها یا تعاملات در طول زمان را نشان می‌دهد. این نمودار برای مدل‌سازی تغییرات وابسته به زمان مناسب است.

* **جزئیات:**
  + محور افقی زمان و محور عمودی وضعیت‌ها را نشان می‌دهد.
  + وضعیت‌های مختلف و انتقالات زمانی مشخص می‌شوند.
* **کاربرد:** طراحی سیستم‌هایی که وابستگی زمانی دارند.

**7. Interaction Overview Diagram**

Interaction Overview Diagram ترکیبی از Activity Diagram و Sequence Diagram است. این نمودار دید کلی از تعاملات سیستم را ارائه می‌دهد.

* **جزئیات:**
  + شامل فعالیت‌ها و توالی تعاملات است.
  + اجزای مختلف به شکل بلوک‌های مرتبط نشان داده می‌شوند.
* **کاربرد:** نمایش ساده و کلی از تعاملات پیچیده.

**8. Profile Diagram**

Profile Diagram برای سفارشی‌سازی و تنظیم ویژگی‌های سیستم استفاده می‌شود. این نمودار معمولاً در توسعه نرم‌افزارهای پیشرفته به کار می‌رود.

* **جزئیات:**
  + ویژگی‌ها و محدودیت‌های مرتبط با اجزای سیستم نمایش داده می‌شوند.
  + تنظیمات خاص سیستم مشخص می‌شوند.
* **کاربرد:** طراحی پروفایل‌های قابل تنظیم برای سیستم‌های پیچیده.

**9. Composite Structure Diagram**

Composite Structure Diagram ساختار داخلی یک کلاس یا جزء را نمایش می‌دهد، شامل اجزای داخلی و نحوه ارتباط آن‌ها.

* **جزئیات:**
  + اجزا به‌صورت مستطیل‌های کوچک‌تر داخل کلاس نمایش داده می‌شوند.
  + ارتباطات بین اجزا به شکل خطوط مشخص می‌شوند.
* **کاربرد:** طراحی اجزا و ارتباطات داخلی سیستم.

**10. Class Diagram**

Class Diagram روابط بین کلاس‌ها، ویژگی‌ها و متدها را نشان می‌دهد. این نمودار به‌ویژه در طراحی ساختار نرم‌افزار بسیار مهم است.

* **جزئیات:**
  + کلاس‌ها با مستطیل شامل نام، ویژگی‌ها و متدها نمایش داده می‌شوند.
  + روابط شامل وابستگی، ارث‌بری و ترکیب مشخص می‌شوند.
* **کاربرد:** طراحی معماری نرم‌افزار و شناسایی روابط بین اجزا.

**11. Package Diagram**

Package Diagram گروه‌بندی کلاس‌ها یا ماژول‌ها را در بسته‌ها (Packages) نمایش می‌دهد و نحوه ارتباط بسته‌ها را مشخص می‌کند.

* **جزئیات:**
  + بسته‌ها با مستطیل‌هایی که علامت فایل دارند نمایش داده می‌شوند.
  + ارتباطات بین بسته‌ها با خطوط مشخص می‌شوند.
* **کاربرد:** سازماندهی و مدیریت کلاس‌ها در سیستم.

**12. Object Diagram**

Object Diagram نمونه‌های خاص از کلاس‌ها و روابط آن‌ها را در یک زمان مشخص نمایش می‌دهد. این نمودار برای بررسی وضعیت واقعی سیستم مفید است.

* **جزئیات:**
  + اشیاء با مستطیل‌های ساده‌تر نمایش داده می‌شوند.
  + روابط بین اشیاء مشخص می‌شوند.
* **کاربرد:** تحلیل رفتار سیستم در زمان مشخص.

**13. Components Diagram**

Components Diagram اجزای فیزیکی سیستم و نحوه تعامل آن‌ها را نمایش می‌دهد. این نمودار معمولاً برای مدل‌سازی معماری نرم‌افزارهای پیچیده استفاده می‌شود.

* **جزئیات:**
  + اجزا به شکل مستطیل‌های مشخص‌شده نمایش داده می‌شوند.
  + وابستگی‌ها و ارتباطات فیزیکی مشخص می‌شوند.
* **کاربرد:** طراحی معماری نرم‌افزار و تحلیل اجزای فیزیکی.

ASP.NET Core : بخش دوم

**1. ASP (Active Server Pages):**

این فناوری اولین تلاش مایکروسافت برای ایجاد صفحات وب پویا بود. ASP اجازه می‌داد با استفاده از کدهای سمت سرور (مانند VBScript) داده‌ها پردازش و صفحات HTML تولید شوند. بعداً ASP.NET جایگزین آن شد و توانست برنامه‌های پیچیده‌تر با امنیت بیشتر ارائه دهد.

**2. dotNET (.NET):**

چارچوب **.NET** ابزار جامعی برای توسعه برنامه‌های مختلف مانند وب‌سایت‌ها، اپلیکیشن‌های دسکتاپ، و سرویس‌های ابری است. نسخه **.NET Core** متن‌باز و چندپلتفرمی است و توسعه‌دهندگان می‌توانند با استفاده از آن برنامه‌هایی که روی ویندوز، لینوکس و macOS اجرا می‌شوند، ایجاد کنند.

**3. ASP.NET Core:**

این فناوری جدیدترین نسخه **ASP.NET** است که سریع‌تر، سبک‌تر، و انعطاف‌پذیرتر است. این پلتفرم قابلیت‌هایی مانند مدیریت درخواست‌های HTTP، مسیریابی پیشرفته، و میان‌افزارها (Middleware) را به صورت پیش‌فرض ارائه می‌دهد.

**4. Middleware:**

در ASP.NET Core، **Middleware** ماژول‌هایی هستند که هر درخواست HTTP باید از آن‌ها عبور کند. این ماژول‌ها می‌توانند برای لاگ‌گیری، احراز هویت، یا مدیریت خطاها استفاده شوند. توسعه‌دهندگان می‌توانند Middleware‌های سفارشی خود را نیز بسازند.

**5. Web Routing:**

سیستم **Routing** وظیفه مسیریابی درخواست‌ها (URLها) به کنترلرها و اکشن‌های صحیح را بر عهده دارد. در ASP.NET Core، می‌توانید مسیرها را به صورت انعطاف‌پذیر تعریف و پیکربندی کنید.

**6. Data Annotation:**

**Data Annotations** مجموعه‌ای از ویژگی‌ها هستند که برای تعریف اعتبارسنجی (Validation) و رفتار داده‌ها به‌کار می‌روند. این ویژگی‌ها باعث ساده‌تر شدن کدنویسی در اعتبارسنجی ورودی‌ها و تعریف روابط پایگاه داده می‌شوند.

**7. DbContext:**

**DbContext** یک کلاس مهم در **Entity Framework Core** است که به توسعه‌دهندگان اجازه می‌دهد با پایگاه داده کار کنند. این کلاس رابطی برای ایجاد و مدیریت داده‌ها در جداول پایگاه داده محسوب می‌شود.

**8. Connection String:**

**Connection String** شامل اطلاعاتی است که برای اتصال برنامه به پایگاه داده نیاز است. این اطلاعات شامل آدرس سرور، نام پایگاه داده، نوع احراز هویت، و تنظیمات اتصال است.

**9. Dependency Injection (DI):**

در ASP.NET Core، **Dependency Injection** یک الگوی طراحی است که وابستگی‌های برنامه را مدیریت می‌کند. این قابلیت باعث می‌شود اجزای مختلف برنامه بدون نیاز به ایجاد اشیاء اضافی به‌صورت خودکار به یکدیگر تزریق شوند.

**10. Controller:**

در معماری **MVC**، **Controller** بخشی است که منطق برنامه و پردازش درخواست‌های کاربر را مدیریت می‌کند. Controller‌ها درخواست‌های HTTP را دریافت کرده، اطلاعات لازم را از مدل‌ها بازیابی می‌کنند، و پاسخ مناسب را به کاربر ارسال می‌کنند.

**11. ORM (Object-Relational Mapping):**

**ORM** یک تکنیک است که به توسعه‌دهندگان این امکان را می‌دهد بدون نیاز به نوشتن کوئری‌های SQL به پایگاه داده دسترسی داشته باشند. ORM مانند Entity Framework داده‌های جداول پایگاه داده را به اشیاء برنامه تبدیل می‌کند.

**12. Entity Framework:**

**Entity Framework (EF)** یک ORM محبوب در **.NET** است که مدیریت پایگاه داده را بسیار ساده می‌کند. EF به توسعه‌دهندگان امکان می‌دهد عملیات CRUD (ایجاد، خواندن، به‌روزرسانی و حذف) را بدون نوشتن کدهای SQL انجام دهند.

**13. Razor:**

**Razor** یک موتور نمایش برای تولید صفحات وب پویا است که ترکیبی از کدهای C# و HTML را فراهم می‌کند. این موتور برای ایجاد صفحات سریع و کارآمد بسیار مناسب است.

**14. Validation:**

اعتبارسنجی (**Validation**) در ASP.NET Core برای بررسی و تضمین صحت ورودی‌های کاربران انجام می‌شود. از ابزارهایی مانند Data Annotation یا اعتبارسنجی سمت سرور استفاده می‌شود تا مطمئن شویم داده‌ها با معیارهای مورد نظر مطابقت دارند.