**נוהל לכתיבת קוד**

תוכן עניינים

[מטרת הנוהל 1](#_Toc491159964)

[פעולות ותנאים מקדימים 1](#_Toc491159967)

[דוגמת קוד / דוגמת שימוש 2](#_Toc491159970)

[לתמיכה והבהרות נוספות 2](#_Toc491159971)

# מטרת הנוהל

כתיבת קוד בצורה נכונה בסטנדרטים הנהוגים בעולם.  
כתיבת קוד ל-Web API ב-.NET מחייבת שמירה על כמה עקרונות ונהלים כדי להבטיח איכות גבוהה, קריאות, ותחזוקה קלה של הקוד. כאן תמצא כמה נהלים חשובים!

# פעולות ותנאים מקדימים

חובה לקרוא על Clean Architecture.  
**- חובה לקרוא על SOLID ולהבין מה כל דבר אומר לפני כתיבת קוד!**

1. **Single Responsibility Principle (SRP)**: כל מחלקה צריכה להיות אחראית על חלק אחד ויחיד מהמטודות של התוכנה, וכל מחלקה צריכה לשנות מסיבה אחת בלבד.
2. **Open/Closed Principle (OCP)**: תוכנה צריכה להיות פתוחה להרחבה, אך סגורה לשינוי. כלומר, צריך להיות ניתן להוסיף פונקציונליות חדשה ללא שינוי של קוד קיים.
3. **Liskov Substitution Principle (LSP)**: עצמים בתוכנה יכולים להיות מוחלפים בעצמים מתת-סוגים שלהם מבלי לשנות את נכונות התוכנה.
4. **Interface Segregation Principle (ISP)**: ממשקים צריכים להיות ספציפיים ולא כלליים. כלומר, לא צריך לאלץ את המשתמש להתממשק עם מטודות שהוא לא ישתמש בהן.
5. **Dependency Inversion Principle (DIP)**: כיתות גבוהות יותר בהיררכיה צריכות להיות תלויות בממשקים ולא מחלקות נמוכות יותר. זה מפחית את התלות בפרטי היישום.

בהקשר של פיתוח Web API , שימוש ב-async tasks הוא מרכיב חיוני לכתיבת קוד נכון ויעיל. הנה מספר נקודות מרכזיות לשיקול בעת כתיבת Web API אסינכרוני:

1. **תגובתיות של השרת**: בשימוש ב-**async**, השרת יכול לעבד בקשות מרובות במקביל, מה שמגדיל את יכולת השרת להתמודד עם עומסים גבוהים ולהגיב במהירות לבקשות.
2. **גישה למשאבים חיצוניים**: כאשר ה-API שלך צריך לגשת למשאבים חיצוניים, כמו מסדי נתונים, שירותים אחרים, או קבצים, שימוש ב-**async** יכול למנוע חסימה של התהליכים ולשפר את התגובתיות.
3. **טיפול בעיבודים ארוכי טווח**: עבור פעולות שדורשות זמן רב יחסית, כגון עיבודים מורכבים או גישה למשאבים איטיים, שימוש ב-**async** מאפשר לשרת לטפל בבקשות אחרות במקביל.
4. **שימוש בספריות מודרניות**: רבות מהספריות החדשות והנפוצות ב-.NET תומכות בפונקציות אסינכרוניות, ולכן שימוש ב-**async** מאפשר לנצל את היתרונות של טכנולוגיות אלו.
5. **תחזוקת הקוד וקריאות**: קוד אסינכרוני נכתב בדרך כלל בצורה נקייה וברורה יותר מאשר קוד סינכרוני שמתמודד עם פעולות ארוכות משך, מה שמקל על התחזוקה וההבנה של הקוד.
6. **מניעת חסימת Threads**: באמצעות **async**, ניתן למנוע את הצורך לחסום threads בעת טעינת משאבים או פעולות אחרות שדורשות זמן, ובכך לשמור על יכולת המערכת לט

נוהל לכתיבת קוד נקי:

1. **תכנון מבנה האפליקציה:**
   * חלק את האפליקציה לשכבות מודל, ניהול נתונים, עסקיים, API
   * שמור על עקביות במבנה הפרויקט ובשמות הקבצים.
2. **ניהול תלויות:**
   * השתמש ב-DI (Dependency Injection) לניהול תלויות.
   * העדף בניית קומפוננטים עצמאיים ומבודדים.
3. **כתיבת קוד:**
   * השתמש במוסכמות קריאות לשמות משתנים, מחלקות ומתודות.
   * כתוב תיעוד והערות בקוד כשזה נחוץ.
   * עקוב אחרי עקרונותSOLID לתכנון תוכנה.
4. **טיפול בשגיאות:**
   * טפל בשגיאות באופן יעיל ומדויק.
   * החזר מענה מתאים ללקוח עם מידע מוגבל על השגיאה.
5. **אבטחת ה-API:**
   * שמור על רמת אבטחה גבוהה, כולל חיבורים מוצפנים ואימות משתמשים.
   * הגן על ה-API מפני התקפות נפוצות כמו SQL Injection אוCross-Site Scripting.
6. **בדיקות יחידה ואינטגרציה:**
   * כתוב בדיקות יחידה ואינטגרציה לכל חלק קריטי באפליקציה.
   * השתמש בכלים לאוטומציה של הבדיקות.
7. **תיעוד ו-API Documentation:**
   * כתוב תיעוד מפורט ל-API.
   * השתמש בכלים כמו Swaggerליצירת ממשק משתמש ידידותי ל-API.
8. **תחזוקת קוד:**
   * בצע סקירות קוד קבועות.
   * תחזוק את הקוד מעודכן ונקי מ"קוד מת".
9. **עקוב אחרי שינויים ועדכונים ב-.NET:**
   * עדכן את הספריות והכלים באופן תדיר.
   * עקוב אחרי המלצות ונהלים חדשים מ-Microsoft והקהילה.

**זכור שהמפתח לפיתוח יעיל ואיכותי הוא להישאר מעודכן, לשמור על קוד נקי ומאורגן, ולהיות מודע לצרכים ולבעיות האפשריות שעלולות להתעורר במהלך הפיתוח.**

# דוגמת קוד / דוגמת שימוש

1. כתיבת Class:

|  |
| --- |
| // אמור להיות עם נקודה פסיק בסוף  namespace Project10.Domain.Entites;  // אות גדולה בהתחלה - PascalCase  public class MySample  {  // אות גדולה בהתחלה - PascalCase  public int Id { get; set; }  // אות גדולה בהתחלה - PascalCase  public int Value1 { get; set; }  // אות גדולה בהתחלה - PascalCase  public int Value2 { get; set; }  // אות גדולה בהתחלה - PascalCase  public required string HiddenProps { get; set; }  } |

1. כתיבת שירות ומתודות ( פונקציות ) – תקף גם לקוד בכל מקום בפרויקט, לדברים – קוד בודד:

|  |
| --- |
| // אמור להיות עם נקודה פסיק בסוף  namespace Project10.Application.Services;  // אות גדולה בהתחלה - PascalCase  public class SampleService : BaseService<SampleService>, ISampleService  {  // חובה להשים region  #region Private members  // \_ - מסמן שירות גלובלי לכל הפונקציות  // חייב להיות עם אות קטנה בהתחלה  // readonly חייב להיות - אנחנו מביאים לו השמה בקונסטרקטור  private readonly ISampleRepository \_sampleRepository;  // \_ - מסמן שירות גלובלי לכל הפונקציות  // חייב להיות עם אות קטנה בהתחלה  // readonly חייב להיות - אנחנו מביאים לו השמה בקונסטרקטור  private readonly IRedisService \_redisService;  // דוגמא איך לכתוב קונסטנטה  private const string REDIS\_KEY\_SAMPLES = Constants.REDIS\_KEY\_SAMPLES;  #endregion  // חובה region - בקונסטרקטור  #region Ctor  // כל הפרמטרים חייבים להית עם אותיות קטנות ואז גדולות אם יש שני מילים  public SampleService(  ILogger<SampleService> logger,  IMapper mapper,  ISampleRepository sampleRepository,  IRedisService redisService  ) : base(logger, mapper)  {  // \_ = משנתנה גלובאלי מצעד שמאל  // פרמטר של הפוקציה מצעד ימין  \_sampleRepository = sampleRepository;  // \_ = משנתנה גלובאלי מצעד שמאל  // פרמטר של הפוקציה מצעד ימין  \_redisService = redisService;  }  #endregion  // כל הפונקציות חיובות region  #region Public method  // שם של פונקציה חייב להיות PacalCase  // כל הפרמטרים חייבים להית עם אותיות קטנות ואז גדולות אם יש שני מילים  public async Task<SampleResponse> GetSampleByIdAsync(int id, string someParam)  {  try  {  // פרמטר גלובאלי  \_logger.LogInformation($"{nameof(GetSampleByIdAsync)} Try to get sample by id: {id}");  // שם של משתנה  // var sample = new Sample(); - דוגמה שלא קיימת בקוד  var sample = await \_sampleRepository.GetByIdAsync(id)  ?? throw new BadRequestException($"Sample with id: {id} not found");  var response = \_mapper.Map<SampleResponse>(sample);  return response;  }  catch (Exception ex)  {  var serializeRequest = JsonConvert.SerializeObject(id);  \_logger.LogError(ex, $"Error Occurred, Request: {serializeRequest}.");  throw;  }  }  #endregion  } |

1. כתיבת Record:

|  |
| --- |
| // אמור להיות עם נקודה פסיק בסוף  namespace Project10.Application.Request;  // שם הקלאס - PascalCase  // משאתם רואים זה לא באמת םרמטרים  // רק ברקורד הפרמטרים הופכים להיות משתנים  // רק במקרה הזה אנחנו נכתוב פרמטרים ב PascalCase  public record AddSampleRequest(  int Value1,  int Value2,  string HiddenProps); |

1. מתודות (מפונקציות):

|  |
| --- |
| #region Public method  // שם של פונקציה חייב להיות PacalCase  // כל הפרמטרים חייבים להית עם אותיות קטנות ואז גדולות אם יש שני מילים camelCase  public async Task<SampleResponse> GetSampleByIdAsync(int id, string someParam)  {  try  {  // פרמטר גלובאלי  \_logger.LogInformation($"{nameof(GetSampleByIdAsync)} Try to get sample by id: {id}");  // שם של משתנה  // var sample = new Sample(); - דוגמה שלא קיימת בקוד  var sample = await \_sampleRepository.GetByIdAsync(id)  ?? throw new BadRequestException($"Sample with id: {id} not found");  var response = \_mapper.Map<SampleResponse>(sample);  return response;  }  catch (Exception ex)  {  var serializeRequest = JsonConvert.SerializeObject(id);  \_logger.LogError(ex, $"Error Occurred, Request: {serializeRequest}.");  throw;  }  } |

1. **כאשר אתם כותבים קוד אסינכרוני – להשתמש ב Async Task Await  
   ולא ב .Result!**

# 

# לתמיכה והבהרות נוספות

לתמיכה והבהרות נוספות באפשרותכם לפנות ל- \_\_\_\_\_\_\_\_\_ - צוות תשתיות פיתוח

(טלפון ........)