Tipps & Tricks

Allgemeine Themen in der Software Entwicklung

Bild abdunkeln wenn es zu hell ist

Ja, das geht ganz einfach mit reinem .NET-Framework und C#. Du brauchst im Wesentlichen zwei Schritte:

1. **Mittlere Helligkeit messen**
2. **Bild um einen errechneten Faktor abdunkeln**

---

1. Mittlere Helligkeit messen

Wir laden das JPG in ein `Bitmap`, iterieren über alle Pixel (oder eine Unter-Stichprobe, um es schneller zu machen) und berechnen die durchschnittliche Luminanz mit der üblichen Formel

Y=0,299 R+0,587 G+0,114 BY = 0{,}299\,R + 0{,}587\,G + 0{,}114\,B

```
private static float GetAverageBrightness(Bitmap bmp)
{
    long sum = 0;
    int width = bmp.Width;
    int height = bmp.Height;
    int total = width * height;

    // Optional: für sehr große Bilder nur jedes N-te Pixel prüfen
    int step = (total > 1_000_000) ? 10 : 1;

    for (int y = 0; y < height; y += step)
    for (int x = 0; x < width;  x += step)
    {
        Color c = bmp.GetPixel(x, y);
        // Luminanz-Berechnung
        int lum = (int)(0.299 * c.R + 0.587 * c.G + 0.114 * c.B);
        sum += lum;
    }
```

```
    // Durch Schritt-Vergrößerung wieder korrigieren
    int count = (width / step) * (height / step);
    return sum / (float)count / 255f;  // Normiert auf [0,1]
}
```

Ruft ihr `GetAverageBrightness(yourBitmap)` auf, erhaltet ihr eine Zahl zwischen etwa 0 (komplett schwarz) und 1 (komplett weiß). Werte über ~0.6–0.7 sind in der Regel zu hell für Passfotos.

---

2. Bild abdunkeln

Wenn der gemessene Wert bb zu hoch ist, könnt ihr das Bild um einen Faktor (target/b)(\mathrm{target} / b) abdunkeln. Am einfachsten geht das mit einem **ColorMatrix** und `Graphics.DrawImage`:

```
private static Bitmap AdjustBrightness(Bitmap original, float brightnessFactor)
{
    // ColorMatrix: Diagonale bleibt bei 1, und das Offset in der letzten Spalte skaliert RGB.
    var cm = new ColorMatrix(new float[][]
    {
        new float[] { brightnessFactor, 0,              0,              0, 0 },
        new float[] { 0,              brightnessFactor, 0,              0, 0 },
        new float[] { 0,              0,              brightnessFactor, 0, 0 },
        new float[] { 0,              0,              0,              1, 0 },
        new float[] { 0,              0,              0,              0, 1 }
    });

    var ret = new Bitmap(original.Width, original.Height);
    using (Graphics g = Graphics.FromImage(ret))
    using (var attrs = new ImageAttributes())
    {
        attrs.SetColorMatrix(cm);
        g.DrawImage(
            original,
            new Rectangle(0, 0, ret.Width, ret.Height),
            0, 0, original.Width, original.Height,
            GraphicsUnit.Pixel,
            attrs);
```

```
    }
    return ret;
}
```

---

3. Gesamtroutine

Kombinieren wir beides in einer Methode:

```
public static Bitmap AutoDarken(Bitmap bmp, float targetBrightness = 0.5f)
{
    // 1. aktuelle Helligkeit
    float current = GetAverageBrightness(bmp);

    // 2. gewünschten Zielwert nicht überschreiten
    if (current <= targetBrightness)
        return bmp;  // ist dunkel genug

    // 3. Faktor berechnen und anwenden
    float factor = targetBrightness / current;
    return AdjustBrightness(bmp, factor);
}
```

Beispiel-Aufruf

```
var img = (Bitmap)Image.FromFile("passfoto.jpg");
var darkened = AutoDarken(img, 0.5f);
darkened.Save("passfoto_dunkler.jpg", ImageFormat.Jpeg);
```

- `targetBrightness` ist hier auf 0.5 gesetzt (also mittleres Grau). Je kleiner der Wert, desto dunkler das Ergebnis.
- Siehst du, dass manche Bilder immer noch zu hell sind, kannst du den Default-Wert weiter absenken (z. B. 0.45).

Performance-Tipp

- Für große Bilder lohnt sich `Bitmap.LockBits` + Pointer-Zugriff statt `GetPixel`/`SetPixel`.

- Du könntest auch nur jeden 5. oder 10. Pixel zur Helligkeitsmessung heranziehen (wie im Beispiel), das reicht für eine grobe Schätzung meistens aus.

Mit dieser Methode kannst du automatisiert durch einen ganzen Ordner von Passfotos iterieren und alle zu hellen Bilder automatisch abdunkeln, bevor du sie auf die Plastikkarte druckst.

LiteDB Kommandos für die Abfrage mit LiteDB Studio

Hier ein kompaktes, druckfertiges Cheatsheet zur LiteDB SQL-ähnlichen Abfragesprache. Einfach kopieren, als PDF/Print ausgeben – und schon hast du es offline zur Hand.

Übersicht der SQL-Befehle

Befehl	Syntax	Kurzbeschreibung
SELECT	`SELECT [TOP n] <Felder> FROM <Collection> [WHERE <Filter>] [GROUP BY <Felder>]`	
`[HAVING] [ORDER BY [ASC	DESC]] [LIMIT n] [OFFSET n]`	Datenabfrage mit Projektion, Filter, Sortierung, Gruppierung und Pagination
INSERT	`INSERT INTO <Collection> (<Feld1>,...) VALUES (<Wert1>,...)` `INSERT INTO <Collection> (<BsonDocument>)`	Neue Dokumente hinzufügen
UPDATE	`UPDATE <Collection> SET <Feld>=<Wert>[, ...] [WHERE <Filter>]`	Felder bestehender Dokumente ändern
DELETE	`DELETE FROM <Collection> [WHERE <Filter>]`	Dokumente löschen (ohne `WHERE`: alle)
RENAME	`RENAME COLLECTION <alt> TO <neu>` `RENAME INDEX <alt> TO <neu> ON <Collection>`	Umbenennen von Collections oder Indizes
DROP	`DROP INDEX <IndexName> ON <Collection>`	Index löschen
EXPLAIN	`EXPLAIN <SELECT	INSERT

Filter- und Ausdruckssyntax (BsonExpression)

- **Vergleichsoperatoren**
  `=` `!=` `>` `<` `>=` `<=`
- **Logische Verkettung**
  `AND`, `OR`, `NOT`
- **Pfadangaben**
  - Dot-Notation: `Address.City`
  - Array-Zugriff: `Tags[0]`, `[Scores].any(x=>x>50)`
- **Array-/Kollektions-Operatoren**
  - `any`, `all`, `contains`
- **Regex**
  - `/{pattern}/i` (z. B. `/^A.*/i`)

Aggregationen & Funktionen

Funktion	Beschreibung	Beispiel
`COUNT(<Feld	*>)`	Anzahl Elemente/Gruppengröße
`SUM(<Feld>)`	Summe numerischer Werte	`SUM(Price)`
`MIN/MAX(<Feld>)`	Kleinster/Größter Wert	`MAX(Date)`
`AVG(<Feld>)`	Durchschnitt	`AVG(Rating)`
`TOLOWER/TOUPPER()`	Text in Klein-/Großbuchstaben	`TOUPPER(Name)`
`TRIM/LENGTH/SUBSTR`	Zeichenketten-Funktionen	`TRIM(Title)`, `LENGTH(Text)`
`DATETIME(x)`	Wandelt Timestamp/String → DateTime	`DATETIME(CreatedAt)`

> **Hinweis:** Zusätzlich unterstützen alle BsonExpression-Funktionen (docs) u. a. `IIF`, `ISNULL`, `ROUND`, `CEIL`, `FLOOR`.

Beispiele

```
-- 1) Einfache Abfrage mit Filter und Sort
SELECT Name, Age
FROM users
WHERE Age >= 18 AND City = 'Berlin'
```

```
ORDER BY Name ASC
LIMIT 10 OFFSET 0;

-- 2) Gruppierung und Having
SELECT City, COUNT(*) AS Einwohner
FROM users
GROUP BY City
HAVING Einwohner > 1000
ORDER BY Einwohner DESC;

-- 3) Einfügen
INSERT INTO products (Name, Price, Tags)
VALUES ('Kaffeemaschine', 79.99, ['küche', 'elektronik']);

-- 4) Update mit Filter
UPDATE products
SET Price = Price * 0.9
WHERE Tags.contains('elektronik');

-- 5) Löschen
DELETE FROM sessions
WHERE LastAccess < DATETIME('2025-01-01');

-- 6) Collection umbenennen
RENAME COLLECTION oldUsers TO customers;
```

Schnellreferenz Schlüsselwörter

```
SELECT, INSERT, UPDATE, DELETE,
FROM, INTO, VALUES,
WHERE, GROUP BY, HAVING,
ORDER BY, LIMIT, OFFSET,
RENAME COLLECTION, RENAME INDEX,
DROP INDEX, EXPLAIN
```

> **Tipp:** Bei sehr vielen Feldern oder dynamisch konfigurierten Sets lieber die `WHEN ... CONTAINS`-Variante in Switch-Statements nutzen, oder direkt `BsonExpression`-Filter anlegen. Damit bleibst du flexibel und wartbar.

Viel Erfolg beim Ausdrucken und Entwickeln!

OpenApi Doc zu Proxy

Kurzantwort

- NSwag bekommst du als Windows-GUI (NSwagStudio), als CLI und als NuGet/MSBuild-Integration. Download/Installationswege siehe unten.
- „Lists statt Arrays“ stellst du über die Generator-Settings ein (arrayType/arrayInstanceType/responseArrayType/parameterArrayType). Beispiel-nswag.json ist weiter unten.

Wo bekomme ich NSwag her?

- NSwagStudio (GUI + CLI inklusive): auf GitHub unter “Releases/Downloads” – MSI oder ZIP (xcopy). (github.com)
- NSwag.MSBuild (für automatische Generierung beim Build): NuGet-Paket NSwag.MSBuild in dein Projekt einbinden. (nuget.org, github.com)
- Alternativ: CLI als .NET-Tool/ZIP; allgemeine Befehle/Kommandos siehe Doku. (github.com)
- Offizielle Einstiegshilfe von Microsoft (Codegen-Optionen, NSwagStudio, NuGet, CLI): (learn.microsoft.com)

Beispiel: C#-Client mit List<T> statt T[] Variante A: nswag.json (empfohlen, reproduzierbar)

- Lege neben dein Projekt eine nswag.json und rufe sie via NSwagStudio („Generate Outputs“) oder per CLI/MSBuild auf.

Beispieldatei (wichtige Stellen markiert)

```
{
  "documentGenerator": {
    "fromDocument": {
      "url": "https://example.local/swagger/v1/swagger.json",
      "output": null,
      "newLineBehavior": "Auto"
    }
  },
  "codeGenerators": {
    "openApiToCSharpClient": {
      "namespace": "MyCompany.MyApiClient",
```

```
      "className": "{controller}Client",
      "generateClientClasses": true,
      "generateClientInterfaces": true,
      "injectHttpClient": true,
      "disposeHttpClient": false,
      "useBaseUrl": false,

      // WICHTIG: Lists statt Arrays
      "arrayType": "System.Collections.Generic.List",
      "arrayInstanceType": "System.Collections.Generic.List",
      "responseArrayType": "System.Collections.Generic.List",
      "parameterArrayType": "System.Collections.Generic.IEnumerable",

      // weitere sinnvolle Defaults für .NET Framework 4.8
      "jsonLibrary": "NewtonsoftJson",
      "classStyle": "Poco",
      "generateDefaultValues": true,
      "generateDataAnnotations": true,

      "output": "Generated\\MyApiClient.g.cs",
      "newLineBehavior": "Auto"
    }
  }
}
```

- Die Settings arrayType/arrayInstanceType/responseArrayType/parameterArrayType sind offiziell Teil der NSwag/NJsonSchema-Generator-Konfiguration. In der „NSwag Configuration Document“-Doku findest du diese Felder (dort z. T. mit anderen Beispieltpyen); du setzt sie einfach auf List bzw. IEnumerable wie oben. (github.com, stackoverflow.com)
- Tipp: Für Methodenparameter ist IEnumerable<T> flexibel, Rückgaben/DTO-Eigenschaften als List<T> erfüllen „List statt Array“.

Aufruf

- Per CLI (aus Projektordner):
  - nswag run .\nswag.json
- Per MSBuild automatisch (z. B. nach dem Build):

```
<!-- In der .csproj des .NET Framework 4.8-Projekts -->
<ItemGroup>
```

```
  <PackageReference Include="NSwag.MSBuild" Version="14.4.0">
    <PrivateAssets>all</PrivateAssets>
  </PackageReference>
</ItemGroup>

<Target Name="NSwag" AfterTargets="Build">
  <Exec Command="$(NSwagExe) run nswag.json" />
</Target>
```

- NSwag.MSBuild stellt die Variable $(NSwagExe) bereit, die das passende nswag-Tool aufruft. (github.com)

Variante B: Direkt über die CLI ohne nswag.json

- Du kannst die wichtigsten Optionen auch als Schalter setzen, z. B.:

```
nswag openapi2csclient ^
  /input:https://example.local/swagger/v1/swagger.json ^
  /output:Generated\Client.g.cs ^
  /namespace:MyCompany.MyApiClient ^
  /ArrayType:"System.Collections.Generic.List" ^
  /ArrayInstanceType:"System.Collections.Generic.List" ^
  /ResponseArrayType:"System.Collections.Generic.List" ^
  /ParameterArrayType:"System.Collections.Generic.IEnumerable"
```

- Die verfügbaren Kommandos/Parameter sind in der CommandLine-Doku beschrieben; die oben gezeigten Typ-Optionen entsprechen den nswag.json-Feldern aus der Config-Doku. (github.com)

Variante C: NSwagStudio (GUI)

- Swagger-URL eintragen, Tab „CSharp Client“ öffnen und bei „Array Type“/„Array Instance Type“ List eintragen; generieren. NSwagStudio kann die Konfiguration als nswag.json speichern. (github.com)

Server-Stubs (falls du „API“ im Sinne von Controller-Gerüst meintest)

- NSwag kann auch Controller aus einem OpenAPI-Dokument erzeugen (ASP.NET Web API 2 für .NET Framework). In der Config nutzt du statt openApiToCSharpClient den Generator openApiToCSharpController, und kannst arrayType etc. identisch setzen; für klassisches Web API ist das aspNetNamespace typischerweise "System.Web.Http". (github.com)

Kompatibilität .NET Framework 4.8

- Die Kernpakete (NSwag.Core/NSwag.Generation/NSwag.AspNetCore) und NSwag.MSBuild unterstützen .NET Framework ≥4.6.2 – damit funktioniert 4.8 problemlos. (www-1.nuget.org, feed.nuget.org, packages.nuget.org)

Zusätzliche Hinweise

- Serializer: Für .NET Framework ist Newtonsoft.Json die pragmatische Wahl. Du kannst über jsonLibrary zwischen NewtonsoftJson und System.Text.Json wechseln (siehe Microsoft-Blogbeitrag mit Beispiel). (devblogs.microsoft.com)
- Best Practices: Auch wenn du List<T> erzwingst, sind für Parameter IEnumerable<T> und für öffentliche APIs häufig IReadOnlyList<T>/ICollection<T> sinnvoller. Wenn du aber explizit „List überall“ willst, setze responseArrayType ebenfalls auf List wie oben.

Wenn du mir deine Swagger/OpenAPI-URL gibst, kann ich dir eine passende nswag.json mit genau den richtigen Settings für dein Projekt (.NET 4.8, Namespace, Dateinamen, usw.) vorbereiten.