C# OO Programmeren

LES 3

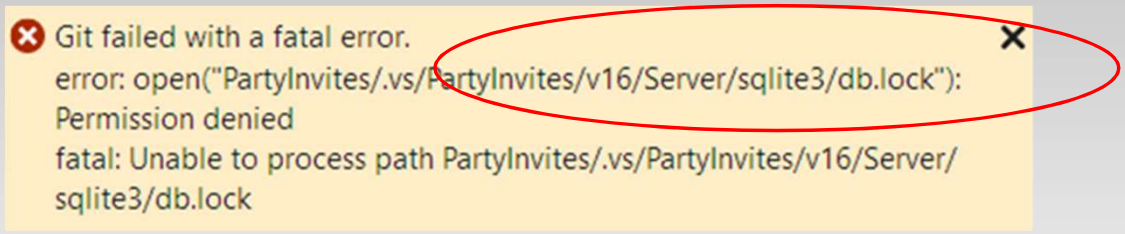
JOHAN DONNÉ

Dienstmededelingen:

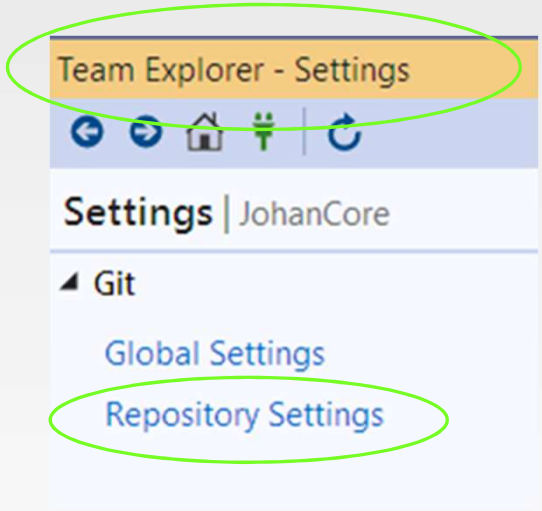
Winform toepassing in .net core

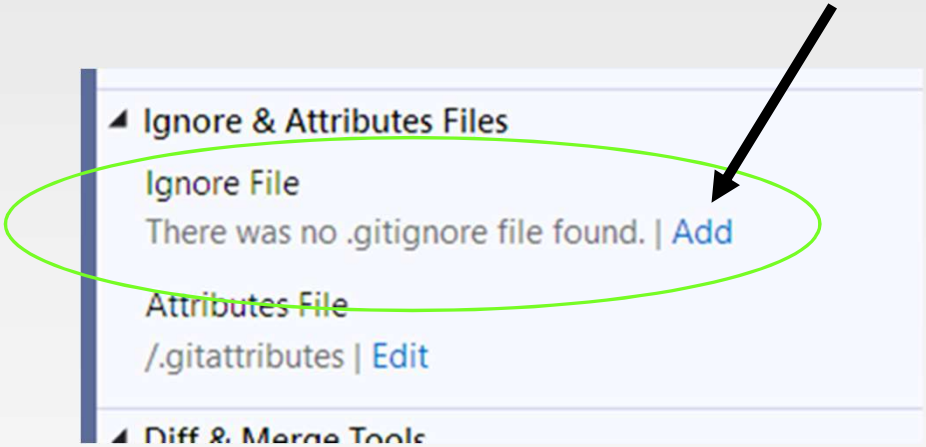
- Kan sinds .net core 3.0
- Nog geen designer ==> niet optimaal
- Dus voor WinForms: beter .Net framework (4.7.2 of 4.8)

Dienstmededelingen:



Oplossing:





Tip:

‘Clean Code’ (Robert C. Martin aka ‘Uncle Bob’)

Boy scout rule:
“Always leave the code you’re editing a little better than you found it”

Vermijd verder te werken met code waarvan je weet dat ze qua stijl niet optimaal is. Refactor zodra je beseft dat iets niet optimaal is. Vaak beter incrementeel dan een complete ‘rewrite’.

Naamgeving, layout, SRP, organisatie, code duplication,

Tip:

"Do or Do Not. There is No Try" ~ Yoda

97 Journey Every Programmer should Accomplish

Overzicht

- Multilayer model
- Praktische organisatie

Multilayer model

Problematiek

- nieuwe toepassing: hoe aanpakken?
- onderhoudbaarheid
  - ⇨ Fouten zoeken en oplossen
  - ⇨ Kleine aanpassingen
- flexibiliteit
  - ⇨ Grote structuur aanpassen,
  - ⇨ Uitbreiden,
  - ⇨ Delen hergebruiken
- testbaarheid van afzonderlijke delen
  - ⇨ Unit Tests
  - ⇨ Integratietests
  - ⇨ UI-tests

aspecten

- User Interface
- Data (persistentie)
- Logica

Structuur van toepassing ~ deelaspecten

$\Rightarrow$ Multilayer model

3 lagenmodel

= klassieke indeling

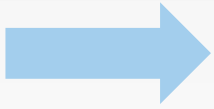
- Presentatielaag: communicatie met buitenwereld
  - ⇨ console, GUI, Webtoepassing, API (als service)
- ‘Data Access’ laag
  - ⇨ koppeling met bestanden, database, web
  - ⇨ koppeling met netwerk, communicatie, API (als cliënt)
- (Business) Logic layer
  - ⇨ logica, intelligentie, ‘Business Model’, ‘Business Rules’

Meerlagenmodel

⇒ Toepassing opsplitsen in afzonderlijke ‘layers’

- Op voorhand vastleggen hoe ‘layers’ met elkaar communiceren (cliënt – service logica, API)

- GUI gebruikt ‘services’ van ‘Logica’,
  ‘Logica’ gebruikt ‘services’ van Data Access layer



‘Dependencies’ (afhankelijkheden)

‘Separation of concern’

Elke layer vormt ‘Black Box’ abstractie.
Van buitenaf geen zicht op details van interne implementatie.
Geen ‘ongewenste intimiteiten’

Bv. In UI niet uitgaan van details van databank structuur

- Wat als databank structuur wijzigt
- Wat als databank vervangen wordt door bestands-I/O of Web-API
- Uitzondering: eenvoudige CRUD-toepassing

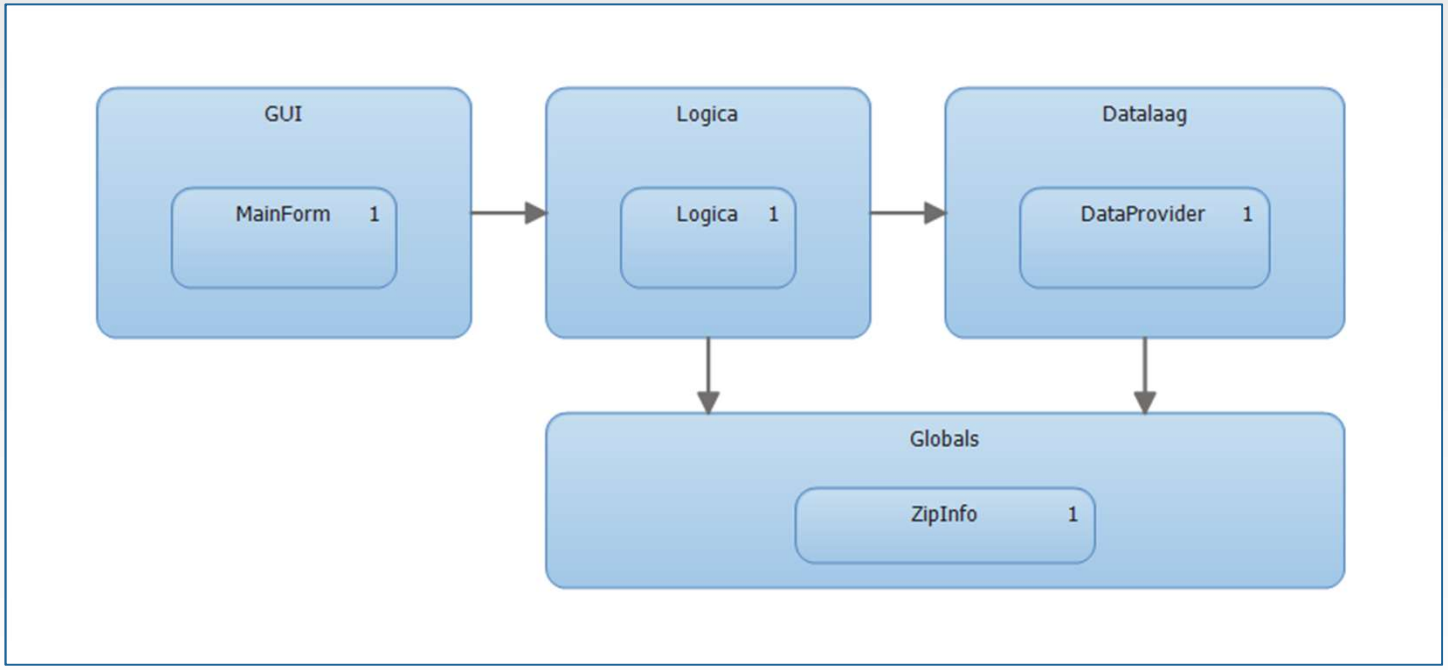
Dus:

- Geen publieke velden
- Enkel noodzakelijke klassen, methodes en properties public maken

Multilayer model in C#

Demo: gemeente zoeker

Opzoeken van gemeente op basis van postcode of (deel van) naam

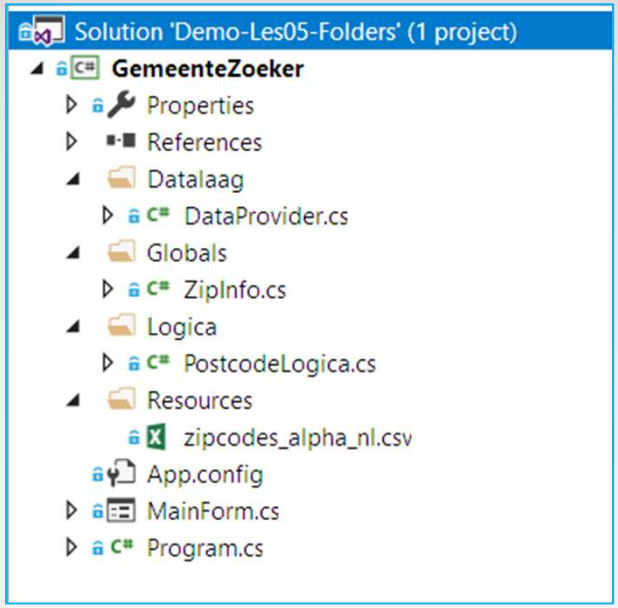


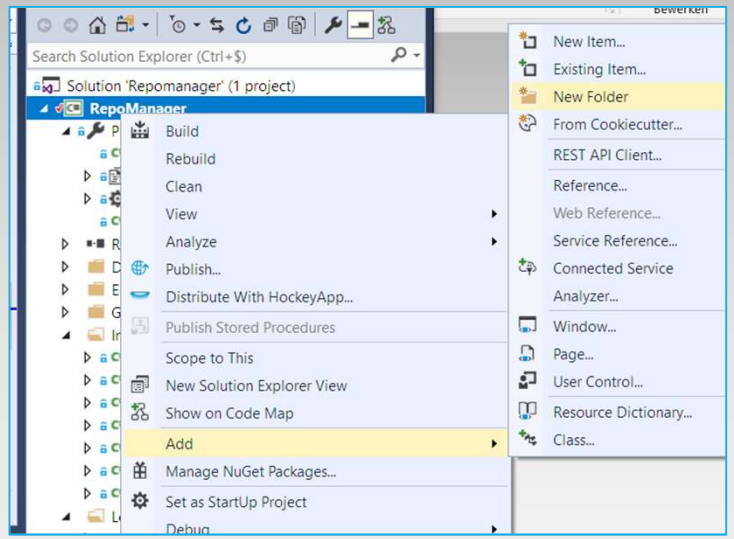
Concreet in C#

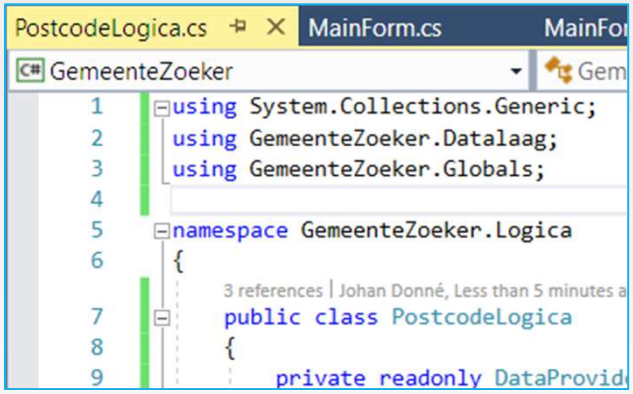
- Via project Folders
- Class libraries

Folder structuur in één project:

- Voor elke laag een eigen folder:







==> Opgelet! Binnen elke folder eigen namespace: eventueel ‘using’s toevoegen…

Implementatie voor ZipInfo (in Globals)

```
namespace GemeenteZoeker.Globals
{
    8 references | Johan Donne, 2 hours ago | 1 author, 1 change
    public struct ZipInfo : IComparable<ZipInfo>
    {
        2 references | Johan Donné, 2 hours ago | 1 author, 1 change
        public int Zipcode { get; }
        4 references | Johan Donné, 2 hours ago | 1 author, 1 change
        public string Gemeente { get; }
        2 references | Johan Donné, 2 hours ago | 1 author, 1 change
        public string Provincie { get; }

        1 reference | Johan Donné, 2 hours ago | 1 author, 1 change
        public ZipInfo(int zipcode, string gemeente, string provincie)...

        2 references | Johan Donné, 2 hours ago | 1 author, 1 change
        public override string ToString()...

        0 references | Johan Donné, 2 hours ago | 1 author, 1 change
        public int CompareTo(ZipInfo other)
        {
            return this.Gemeente.CompareTo(other.Gemeente);
        }
    }
}
```

Implementatie voor Datalaag

```
using System.Collections.Generic;
using System.IO;
using GemeenteZoeker.Globals;

namespace GemeenteZoeker.Datalaag
{
    3 references | Johan Donné, Less than 5 minutes ago | 1 author, 2 changes
    public class DataProvider
    {
        private const string filename = @"resources\zipcodes_alpha_nl.csv";

        2 references | Johan Donné, 1 hour ago | 1 author, 1 change
        public List<ZipInfo> GemeenteLijst { get; }

        1 reference | Johan Donné, Less than 5 minutes ago | 1 author, 2 changes
        public DataProvider()
        {
            GemeenteLijst = ReadGemeenteInfo(filename);
        }

        1 reference | Johan Donné, 1 hour ago | 1 author, 1 change
        private List<ZipInfo> ReadGemeenteInfo(string filename)...
    }
}
```

Implementatie voor Logica

```
using System.Collections.Generic;
using GemeenteZoeker.Datalaag;
using GemeenteZoeker.Globals;

namespace GemeenteZoeker.Logica
{
    3 references | Johan Donné, 2 hours ago | 1 author, 1 change
    public class PostcodeLogica
    {
        private readonly DataProvider dataProvider;

        1 reference | Johan Donné, 2 hours ago | 1 author, 1 change
        public PostcodeLogica()
        {
            dataProvider = new DataProvider();
        }

        1 reference | Johan Donné, 2 hours ago | 1 author, 1 change
        public List<string> GetMatchingResults(string query)
        {
            var matches = new List<ZipInfo>();
            foreach (var zipInfo in dataProvider.GemeenteLijst)
            {
                if (zipInfo.ToString().ToUpper().Contains(query.ToUpper())) matches.Add(zipInfo);
            }
            matches.Sort();
            var result = new List<string>();
            foreach (var zipInfo in matches)
            {
                result.Add(zipInfo.ToString());
            }
            return result;
        }
    }
}
```

Implementatie voor GUI

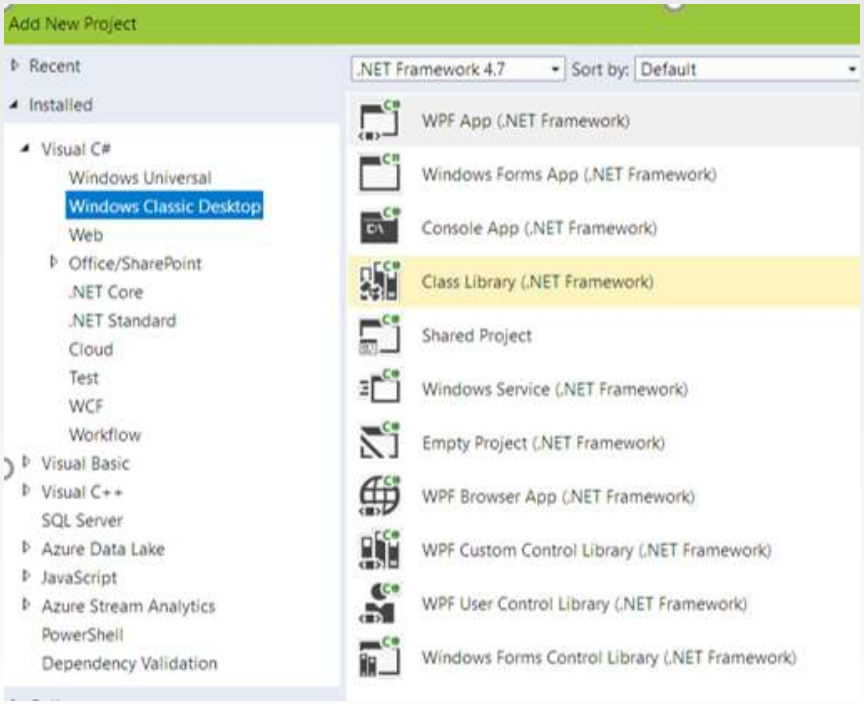
```
using System;
using System.Windows.Forms;
using GemeenteZoeker.Logica;

namespace GemeenteZoeker
{
    3 references | 0 changes | 0 authors, 0 changes
    public partial class MainForm : Form
    {
        private readonly PostcodeLogica logica;

        1 reference | 0 changes | 0 authors, 0 changes
        public MainForm()
        {
            InitializeComponent();
            logica = new PostcodeLogica();
        }

        1 reference | 0 changes | 0 authors, 0 changes
        private void ZoekTextChanged(object sender, EventArgs e)
        {
            resultBox.Lines = logica.GetMatchingResults(zoekBox.Text).ToArray();
        }
    }
}
```

Voorbeeldcode

Demo-Les03-Folders

Class Library

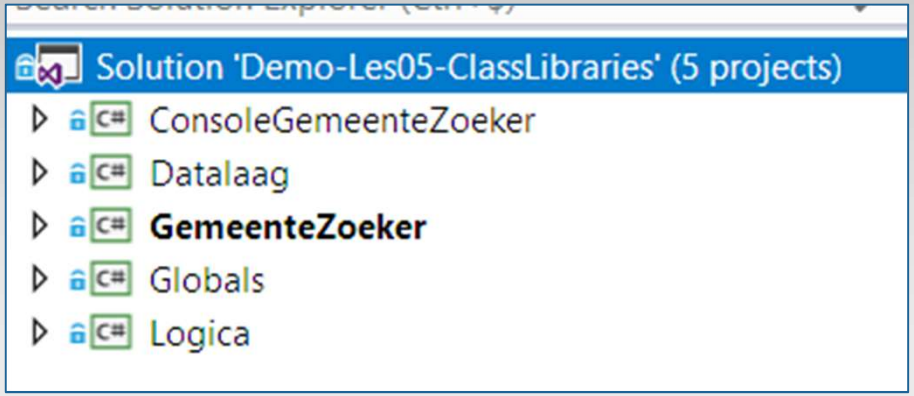
- Afzonderlijk project dat een bibliotheek van klassen bevat (dus geen uitvoerbare toepassing)
- Wordt gecompileerd naar een ‘DLL’ (Dynamic Link Library)
- Elke laag in een eigen Class Library



GemeenteZoeker

Lagen in Class Libraries

Structuur Demo-Les05-ClassLibraries

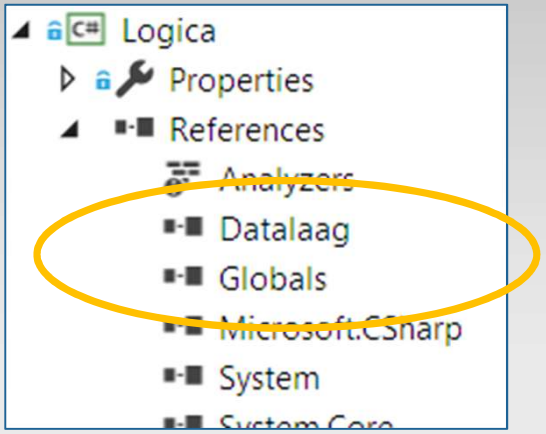


5 projecten:

- Globals, Datalaag, Logica
- GemeenteZoeker (GUI)
- ConsoleGemeenteZoeker (console versie)

Samenstelling van verschillende lagen

- Referenties toevoegen volgens afhankelijkheden
- ‘Using’ toevoegen naar gebruikte namespaces



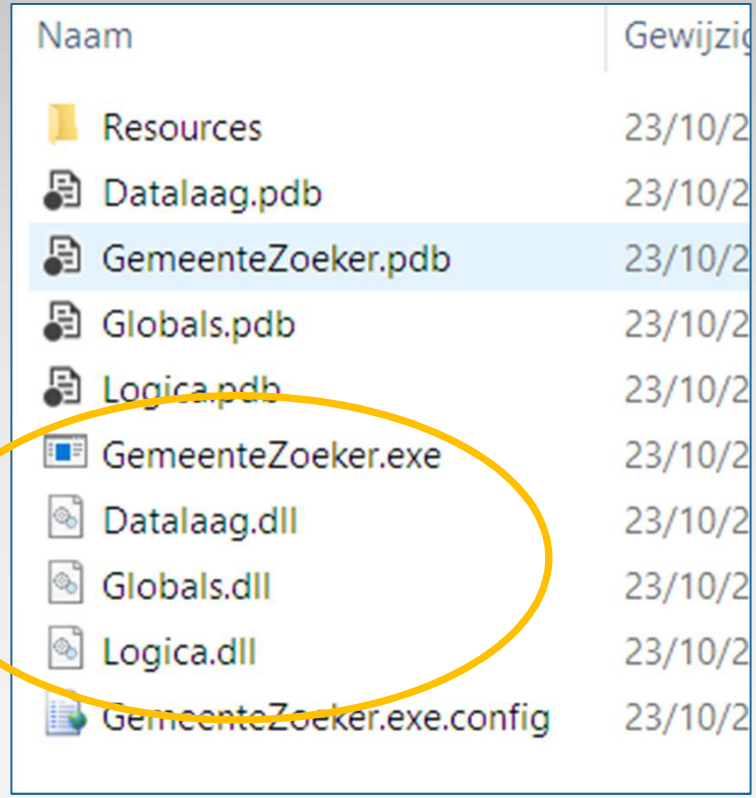

```
using Datalaag;
using Globals;
using System.Collections.Generic;

namespace Logica
{
    5 references | Johan Donné, Less than 5 minutes ago | 1 author, 1 change
    public class PostcodeLogica
    {
        private readonly DataProvider dataProvider;

        2 references | Johan Donné, Less than 5 minutes ago | 1 author, 1 change
        public PostcodeLogica()
        {
            dataProvider = new DataProvider();
        }
```

Daarna gebruiken alsof in eigen project beschikbaar

Resultaat na Build:

In ‘Bin/Debug’ van toepassing:

- ‘.exe’ voor toepassing
- ‘.dll’ voor elke klassebibliotheek



Meerlagenmodel praktische aanpak

Praktische aanpak 3-lagen toepassing: stappen

- ⇨ De grote delen van je toepassing aflijnen (= lagen)
- ⇨ Architectuurschema met ‘lagenstructuur’ en afhankelijkheden tekenen.
- ⇨ Raamwerk voor solution opzetten (met folders of classlibraries).
- ⇨ Beslissen wat er in elk van de lagen moet gebeuren.
  In elke laag de nodige klassen schrijven
- ⇨ Alle delen op de goede manier verbinden
  (binnen elke laag instanties maken volgens de afhankelijkheden).

Praktische aanpak 3-lagen toepassing: naamgeving

- Naam van het (uitvoerbaar) startproject eindigt met ‘Main’.
- De naam van de klasse die de UI implementeert is ‘ConsoleUi’ of ‘MainForm’ (bij WinForm of WPF/UWP toepassingen).
- Klassenbibliotheken (projecten) voor de implementatie van een laag bevatten de naam van de laag + ‘Layer’ of ‘Laag’ (de namespace heeft dezelfde naam).

Multi layer model

Nabeschouwing

Lagenmodel : ‘separation of concern’

Resultaat:

Interactie tussen de lagen gebeurt enkel via vooraf vastgelegde methodes en data types.

Elke laag op zich wordt minder complex, beperkter probleem

‘separation of concern’

Resultaat:

Het wordt snel duidelijk waar een aanpassing of bug moet aangepakt worden

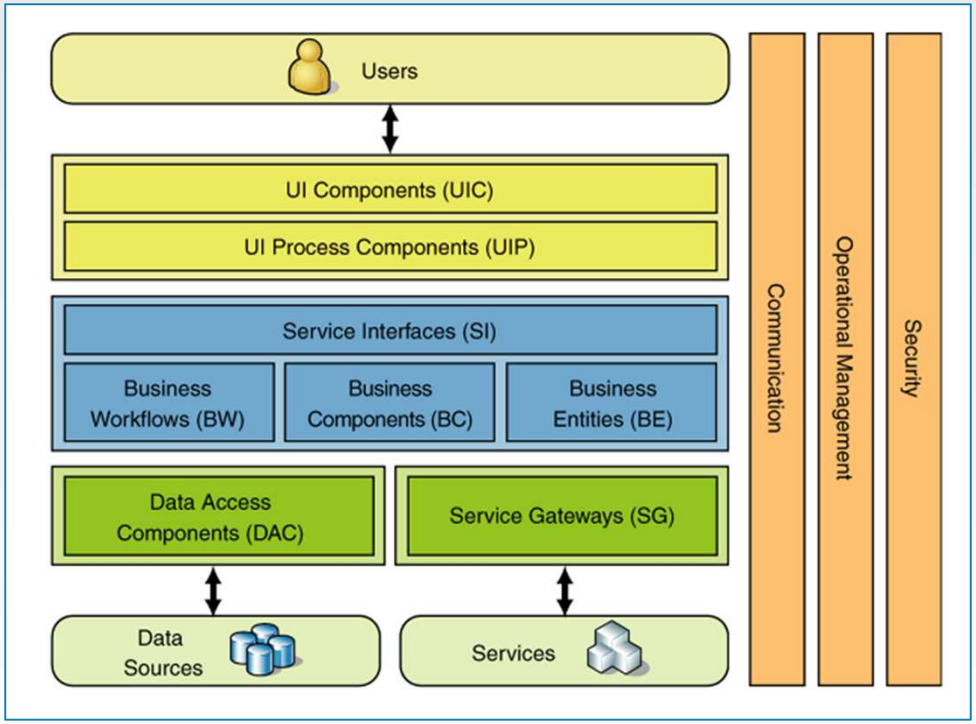
Wijziging van de implementatie van één laag heeft geen impact op de rest van de toepassing

Afzonderlijke units worden gemakkelijker te testen los van de rest van de toepassing.

Multilayer model

3 lagen architectuur komt vaak voor , maar er kunnen ook meer lagen zijn.

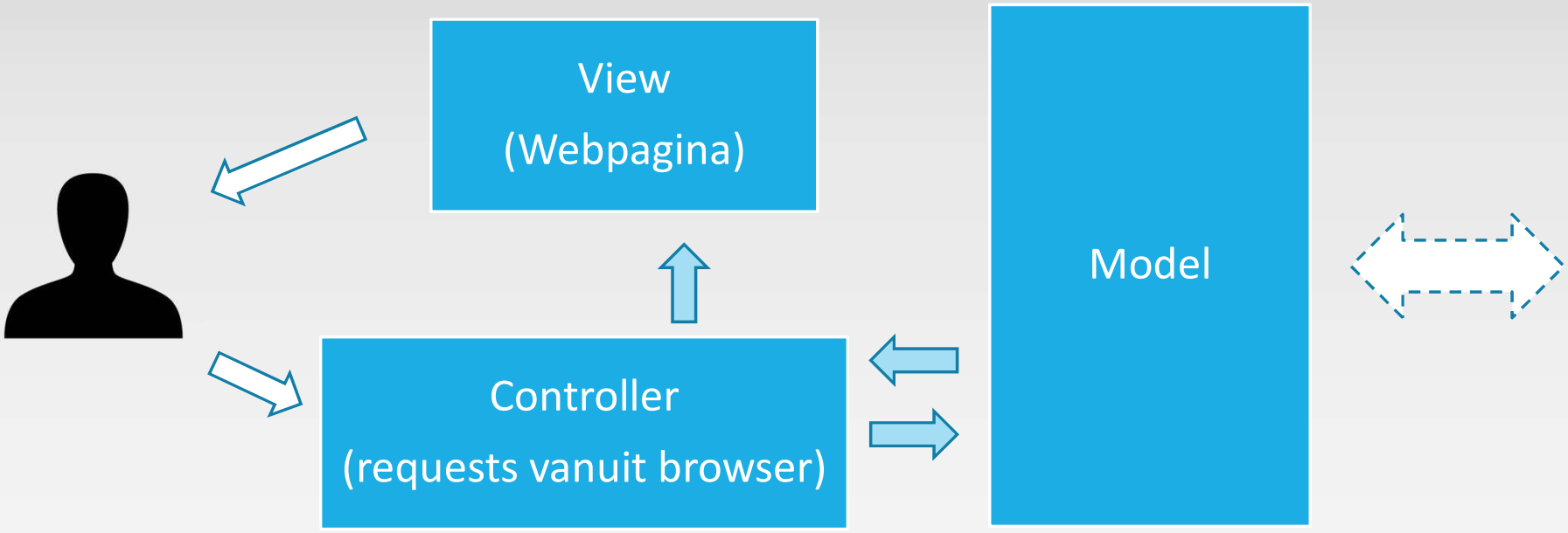
Een toepassing kan ook hiërarchieën van lagen bevatten



Multi layer model

Naast toepassing op architectuur van hele toepassing wordt de techniek van ‘separation of concern’ vaak ook toegepast op architectuur van kleinere delen van een toepassing: ‘architectuur patterns’ of ‘design patterns’.

Design pattern: Model – View – Controller (MVC)



Design pattern: Model – View – ViewModel (MVVM)