# Device Programming

Tuur Vanhoutte

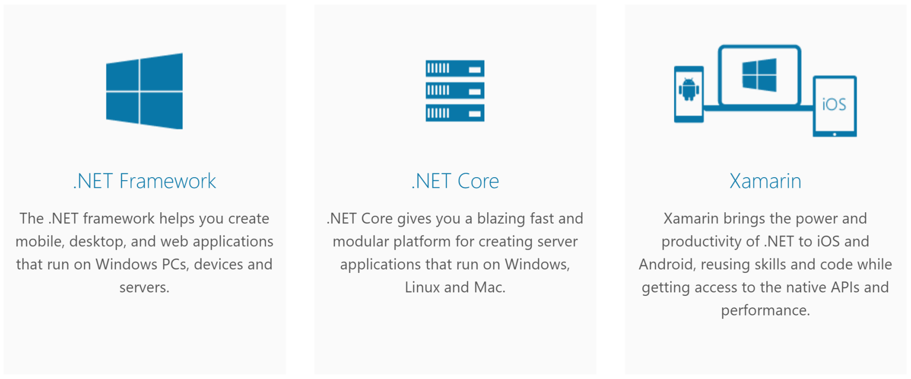
5 oktober 2020

# Inhoudsopgave

# 1 .NET

.NET is a free, cross-platform, open source developer platform (*) for building many different types of applications.

* languages + libraries



.NET Framework

The .NET framework helps you create mobile, desktop, and web applications that run on Windows PCs, devices and servers.

.NET Core

.NET Core gives you a blazing fast and modular platform for creating server applications that run on Windows, Linux and Mac.

Xamarin

Xamarin brings the power and productivity of .NET to iOS and Android, reusing skills and code while getting access to the native APIs and performance.

Figuur 1: .NET ecosystem

## 1.1 Languages

- Syntax very similar to C, C++, Java & JavaScript
- Functional programming language, cross-platform, open source
- Approachable English-like language for OOP

## 1.2 Applications

- desktop
- web & server
- mobile
- gaming
- IoT
- AI

### 1.2.1 Desktop

- UWP (Universal Windows Project)
- Xamarin.Mac
- WPF (Windows Presentation Foundation)
- WinForms (Windows Forms)

### 1.2.2 Web & Server

- ASP.NET
- ASP.NET Core

### 1.2.3 Mobile

- UWP (Universal Windows Project)
- Xamarin

### 1.2.4 Gaming

- Unity
- CryEngine

### 1.2.5 IoT

- UWP
- .NET Core IoT

### 1.2.6 AI

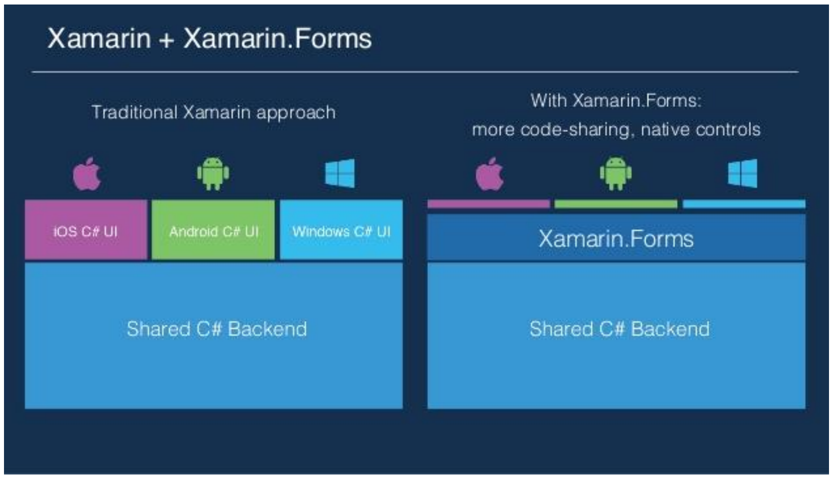
- Cognitive Services
- Azure Machine Learning
- Machine Learning and AI Libraries
- F# for Data Science and ML

## 1.3 Xamarin

- ‘Target all platforms with a single, shared codebase for Android, iOS, Windows’.
- Developen van Mobile devices lastig: verschillende platformen, verschillende talen voor elk device.
- Oplossing: Xamarin
- Extensie op Visual Studio.

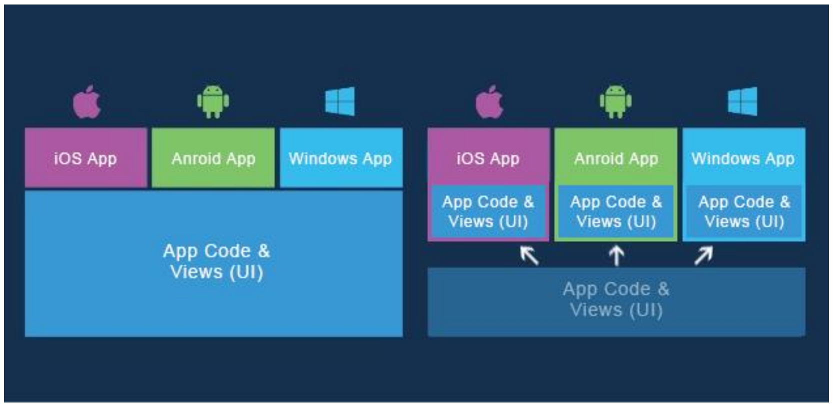
Figuur 2: Xamarin Logo

### 1.3.1 Xamarin - UI Technology



Figuur 3: Native vs Xamarin.Forms

### 1.3.2 Xamarin - Code Sharing strategy



Figuur 4: .NET Standard vs Shared (Assets) Project

Met Shared Assets Project maken we de UI voor elk platform apart. Wij gaan vooral werken met .NET Standard.

## 1.4 Samenvatting

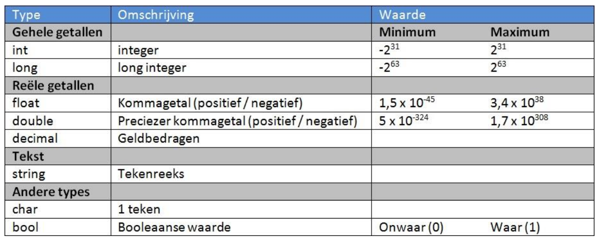
- What devices, platforms, etc. can we target using .NET, and what programming languages can we use?
- What is the basic difference between .NET Standard and Shared Assets projects in Xamarin?
- What is the difference between Xamarin native and Xamarin.Forms? What are the advantages and disadvantages?
- How to set up and understand the structure of a Xamarin project for the labs in this course, and how to debug on the different platforms.

# 2 C# Syntax

## 2.1 Python vs C#

- curly brackets { } in plaats van indenting

## 2.2 Datatypes



| Type | Omschrijving | Waarde | |
|---|---|---|---|
| **Gehele getallen** | | **Minimum** | **Maximum** |
| int | integer | $-2^{31}$ | $2^{31}$ |
| long | long integer | $-2^{63}$ | $2^{63}$ |
| **Reële getallen** | | | |
| float | Kommagetal (positief / negatief) | $1{,}5 \times 10^{-45}$ | $3{,}4 \times 10^{38}$ |
| double | Preciezer kommagetal (positief / negatief) | $5 \times 10^{-324}$ | $1{,}7 \times 10^{308}$ |
| decimal | Geldbedragen | | |
| **Tekst** | | | |
| string | Tekenreeks | | |
| **Andere types** | | | |
| char | 1 teken | | |
| bool | Booleaanse waarde | Onwaar (0) | Waar (1) |

Figuur 5: Datatypes in C#

## 2.3 Collections

- Array
- Dictionary<TKey, TValue>
- List<T>

Collection type = fixed! ⇒ Je kan alleen objecten van het gekozen type toevoegen aan een collection

```
//collections of type Person:
Person[] teacherArr = new Person[10];
List<Person> teacherList = new List<Person>();

//You can only add Person objects to these collections!
```

### 2.3.1 Arrays

= meerdere variabelen van hetzelfde type

```
//initialize int array with 10 positions:
int[] numbers = new int[10];
//save number 13 in the first position
numbers[0] = 13;
//print the value of the first number in the array:
Debug.WriteLine("The␣first␣number␣is:" +numbers[0]);
//intialize and fill another array with 4 numbers:
int[] startPositions = { 4, 1, 9, 3 };
```

### 2.3.2 Dictionary <TKey, TValue>

```
//declare dictionary with key type & value type
Dictionary<string, int> studentScores = new Dictionary<string, int>();
//add two elements (key value pairs)
studentScores.Add("Jean-Jacques", 13);
studentScores.Add("Jean-Louis", 4);
//get the score of Jean-Jacques
int score = studentScores["Jean-Jacques"];
```

### 2.3.3 List<T>

```
//declare list, fill one by one:
List<string> emailList = new List<string>();
emailList.Add("stijn.walcarius@howest.be");
emailList.Add("frederik.waeyaert@howest.be");
//get elements out (two ways):
string first = emailList.ElementAt(0);
string second = emailList[1];
//declare + fill list:
List<string> teacherList = new List<string> { "SWC", "FWA" };
```

## 2.4 Selections

if / else if / else / switch

```
if(findTheoryTeacher == true) {
    email1 = "frederik.waeyaert@howest.be";
    email2 = null;
}
else if(findLabTeachers == true) {
    email1 = "stijn.walcarius@howest.be";
    email2 = "␣frederik.waeyaert@howest.be";
} else {
    email1 = email2 = null;
}
```

```
switch (teacher){
    case "SWC":
        email = "stijn.walcarius@howest.be";
        break;
    case "FWA":
        email = "frederik.waeyaert@howest.be";
        break;
    default:
        email = "info@howest.be";
        break;
}
```

### 2.5 Loops

for / foreach / while / do while

```
for(int i = 0; i < 100; i++) {
    //do something 100 times
}
```

```
List<string> teacherList = new List<string> { "SWC", "FWA" };
foreach(string teacher in teacherList) {
    //do something
}
```

```
while(endOfClass == false){
    //might never be executed
}
```

```
do {
    //executed at least once!
} while(endOfClass == false);
```

### 2.6 Classes

```
public class Person
{
    //property
    public string Name {
        get {...};
        set {...};
    }

    //constructor
    public Person(string name) {
        this.Name = name;
    }

    //method
    public void Subscribe() {
        //do something
    }
}
```

### 2.7 Instantiate objects

```
Persons p1 = new Person("Stijn");

// Based on the following constructor in the Person class:
public Person (string name) {
    this.Name = name;
}
```

## 2.8 Properties

### 2.8.1 Fields vs properties

- Fields store the actual data
- Properties are used to access those fields
- Auto-implemented properties have a hidden field
- Use properties to control field access
- Enhance input/output control using get & set
- Calculated properties build on other properties
  - No field required
  - Reusability

```
//private field
private int _id;

//property (zetten we altijd public)
public int Id {
    // getter
    get { return _id; }
    // setter
    set { _id = value; }
}
```

### 2.8.2 Default values for properties

- Setting default values can be useful
- Default values can be set. . .
  - . . . with full properties
  - . . . with auto-implemented properties
  - . . . in the constructor

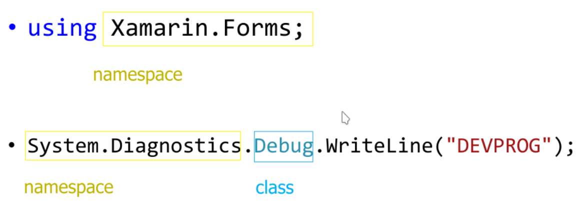
## 2.9 Constructor

- A constructor is called every time you create an instance of a class
- It is used to allow / force the user to provide certain values
- Default constructor is (only) added if a model has no constructors
- Constructor overloading = multiple constructors with either . . .
  - . . . a different number of parameters, or
  - . . . a different type of paramters, or
  - . . . the same parameters in a different order
- Constructors should call each other for enhanced efficiency
- Constructors in inheriting classes call the constructors of the base class

# 3 Streamreader

- Namespaces
- System.Reflection
- Embedded Files
- System.IO

## 3.1 Namespaces



```
• using Xamarin.Forms;

• System.Diagnostics.Debug.WriteLine("DEVPROG");
```

Figuur 6: Namespaces

## 3.2 System.Reflection

*"The classes in the System.Reflection namespace, together with System.Type, enable you to obtain information about loaded assemblies and the types defined within, such as classes, interfaces, and value types. You can also use* ***reflection*** *to create type instances at run time and to invoke them."*

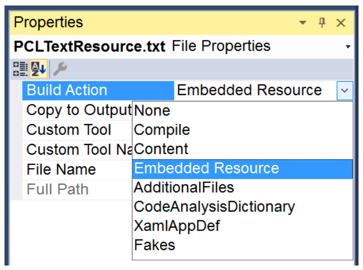
## 3.3 System.IO

= Input/Output `https://developer.xamarin.com/api/namespace/System.IO/`

- StreamReader `https://developer.xamarin.com/api/type/System.IO.StreamReader/`
- StreamWriter `https://developer.xamarin.com/api/type/System.IO.StreamWriter/`

## 3.4 Embedded files

- Textfiles, images, etc.
- Generates a **ResourceID** for the file



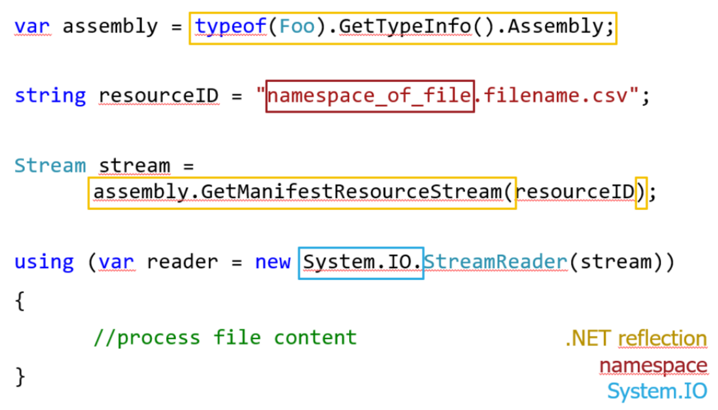
Properties

**PCLTextResource.txt** File Properties

Build Action: Embedded Resource

Figuur 7: Embedded files inladen in een Visual Studio project: rechtermuisknop op 1 of meerdere files ⇒ properties ⇒ build action = Embedded resources

### 3.4.1 Read an embedded file in Xamarin

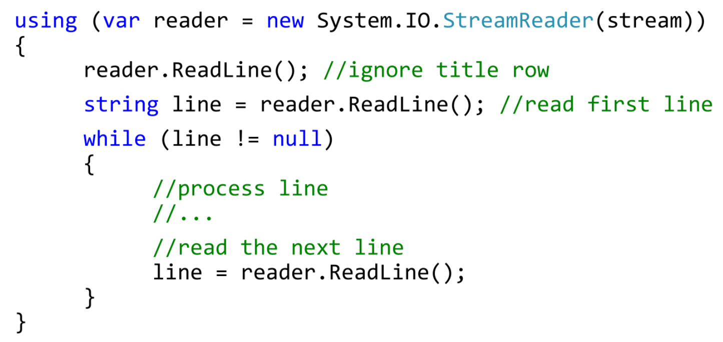


```
var assembly = typeof(Foo).GetTypeInfo().Assembly;

string resourceID = "namespace_of_file.filename.csv";

Stream stream =
       assembly.GetManifestResourceStream(resourceID);

using (var reader = new System.IO.StreamReader(stream))
{
       //process file content
}
```

Figuur 8

•

### 3.4.2 Processing the file's content



```
using (var reader = new System.IO.StreamReader(stream))
{
      reader.ReadLine(); //ignore title row
      string line = reader.ReadLine(); //read first line
      while (line != null)
      {
            //process line
            //...
            //read the next line
            line = reader.ReadLine();
      }
}
```

Figuur 9

## 3.5 Samenvatting

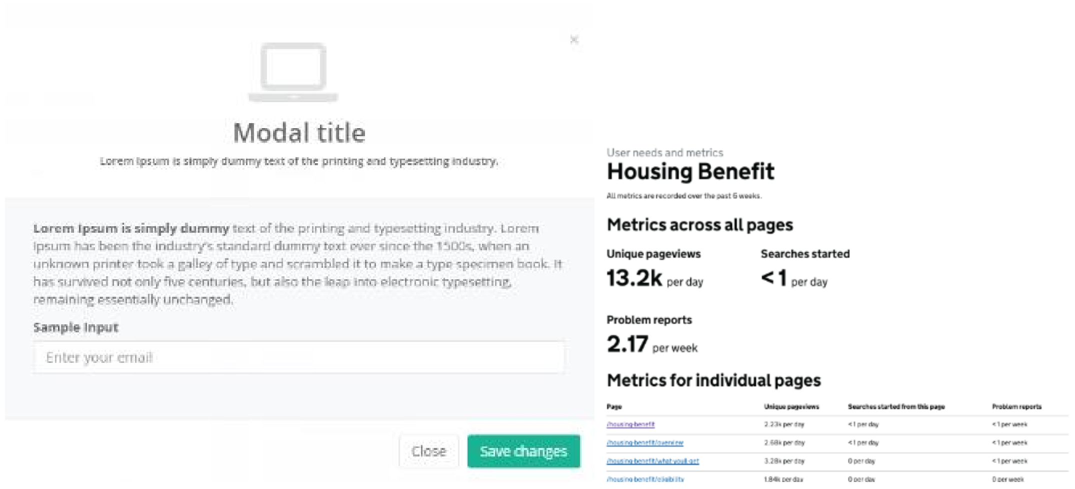
- You understand the importance of **namespaces**, and the techniques of using them in your own projects.
- You can explain the very basics of the **System.IO** and **System.Reflection** namespaces, and what they have to do with reading an embedded file in Xamarin.
- You understand the how and why of the **ResourceID** that's being generated for an embedded file.

# 4 Navigation

## 4.1 Modal vs Modeless

- Modal page: requires user input to continue

- Modeless page: user can go back any time he wants; no input required



Modal title

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

**Lorem Ipsum is simply dummy** text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged.

Sample Input

Enter your email

User needs and metrics

Housing Benefit

All metrics are recorded over the past 6 weeks.

Metrics across all pages

| Unique pageviews | Searches started |
|---|---|
| 13.2k per day | < 1 per day |

Problem reports

2.17 per week

Metrics for individual pages

| Page | Unique pageviews | Searches started from this page | Problem reports |
|---|---|---|---|
| /housing-benefit | 2.23k per day | < 1 per day | < 1 per week |
| /housing-benefit/overview | 2.68k per day | < 1 per day | < 1 per week |
| /housing-benefit/what-youll-get | 3.28k per day | 0 per day | < 1 per week |
| /housing-benefit/eligibility | 1.84k per day | 0 per day | 0 per week |

Figuur 10: Modal page vs Modeless page

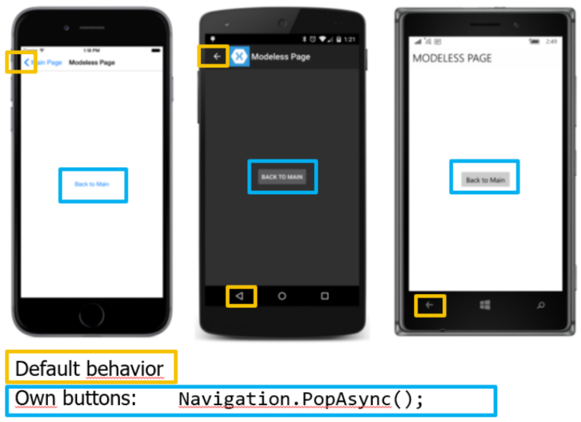
## 4.2 Navigate forward

```
Navigation.PushAsync(new FooPage());
Navigation.PushModalAsync(new FooPage());

// FooPage is hier de XAML page waar we willen naar navigeren
```

- PushAsync vs PushModalAsync
- Navigation object: controls the navigation stack

## 4.3 Navigate back

### 4.3.1 Go back - Modeless page

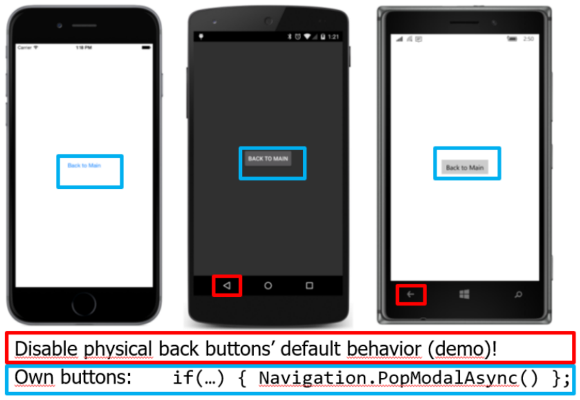


Default behavior

Own buttons: `Navigation.PopAsync();`

Figuur 11

### 4.3.2 Go back - Modal page

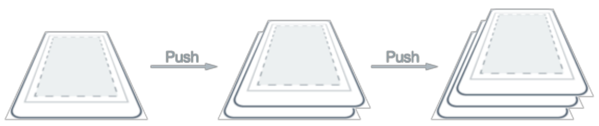


Disable physical back buttons' default behavior (demo)!

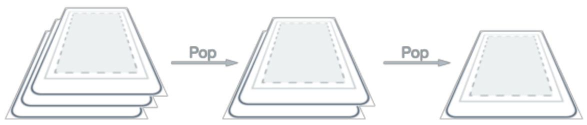
Own buttons: `if(…) { Navigation.PopModalAsync() };`

Figuur 12

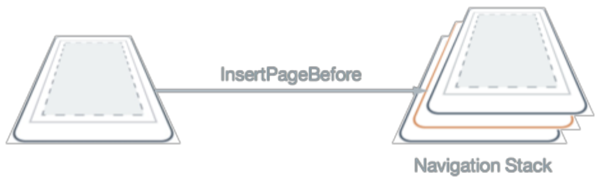
## 4.4 Navigation stack



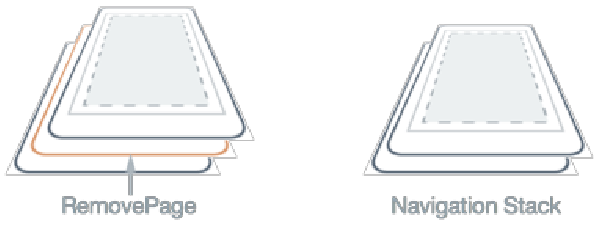
Figuur 13: Pushen op de stack



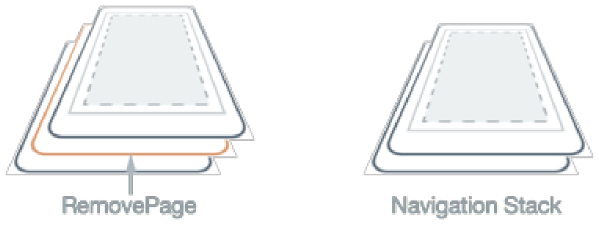
Figuur 14: Pop'en van de stack



Figuur 15: InsertPageBefore



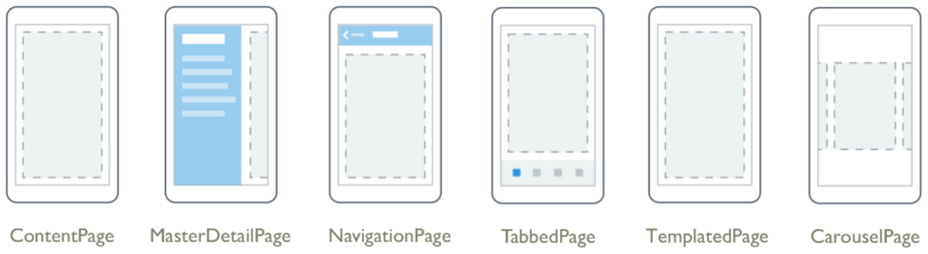
Figuur 16: RemovePage



Figuur 17: PopToRoot

## 4.5 Page types

- ContentPage
- MasterDetailPage (zie Demo_MasterDetail)
- NavigationPage (zie Demo_Navigation)
- TabbedPage (zie Demo_TabbedPage)
- TemplatedPage
- CarouselPage



Figuur 18

## 4.6 Exchanging data

How to exchange data between several pages:

1. Constructor (Demo_MasterDetail)
2. Properties (Demo_TabbedPage)

## 4.7 Samenvatting

- The different **page types** and how to use them.
- The difference between **Modal** and **Modeless** pages, and how to manage navigation for both.
- You know how to **exchange data** between pages in the navigation process.
- You understand the **navigation stack** and how you can manipulate it.
- You can explain the concept of a **master-detail** relation with an example