

Building a Real-world C# Application

Introduction



Filip Ekberg

Principal Consultant & CEO

@fekberg | fekberg.com



Version Check



This version was created by using:

- C# 12
- .NET 8
- Visual Studio 2022



Version Check



This course is 100% compatible with:

- C# 11
- .NET 7
- Visual Studio 2022



Relevant Notes



A note on frameworks, libraries and tools:

- Free community version of Visual Studio 2022
- Many language features work in older versions of C#
- Applicable to all types of .NET applications



**Pause the video at any time
to fill in the exercise!**



Globomantics ToDo

Todo

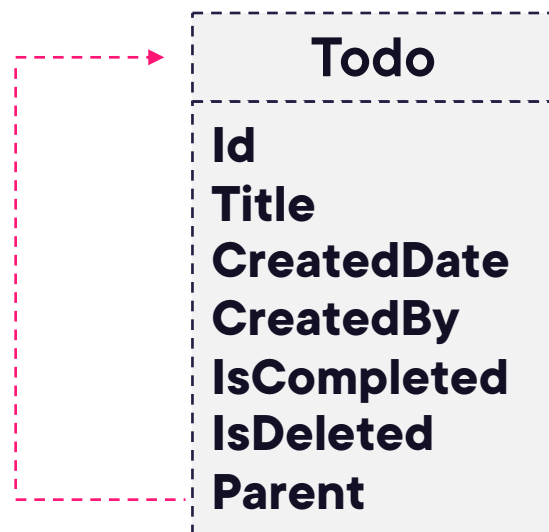


Globomantics ToDo

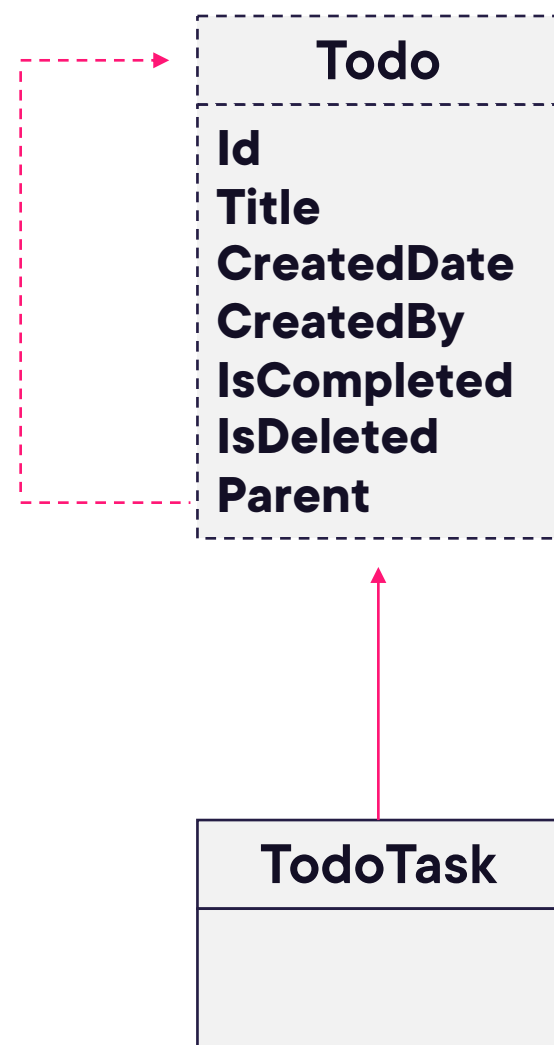
Todo
Id
Title
CreatedDate
CreatedBy
IsCompleted
IsDeleted
Parent



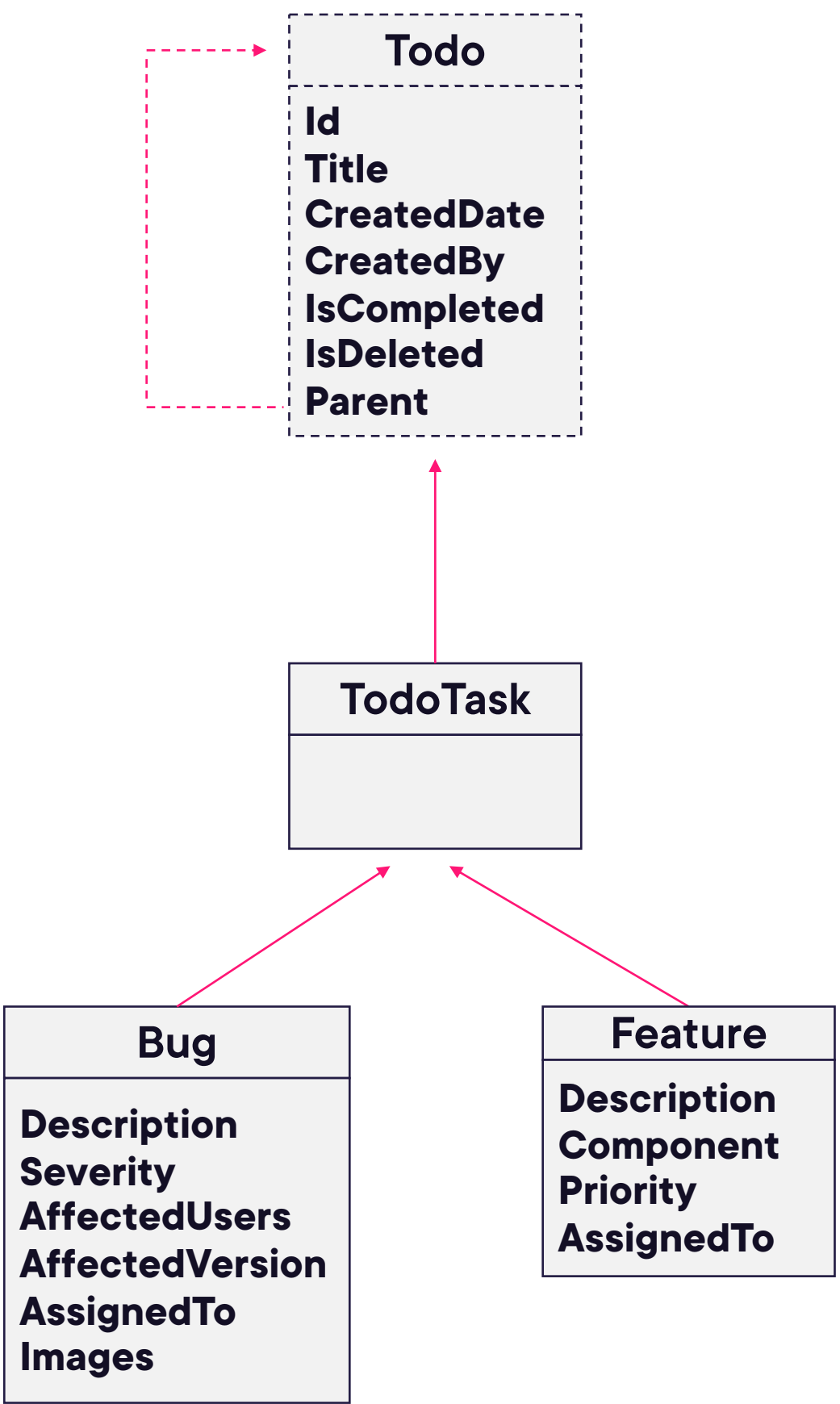
Globomantics ToDo



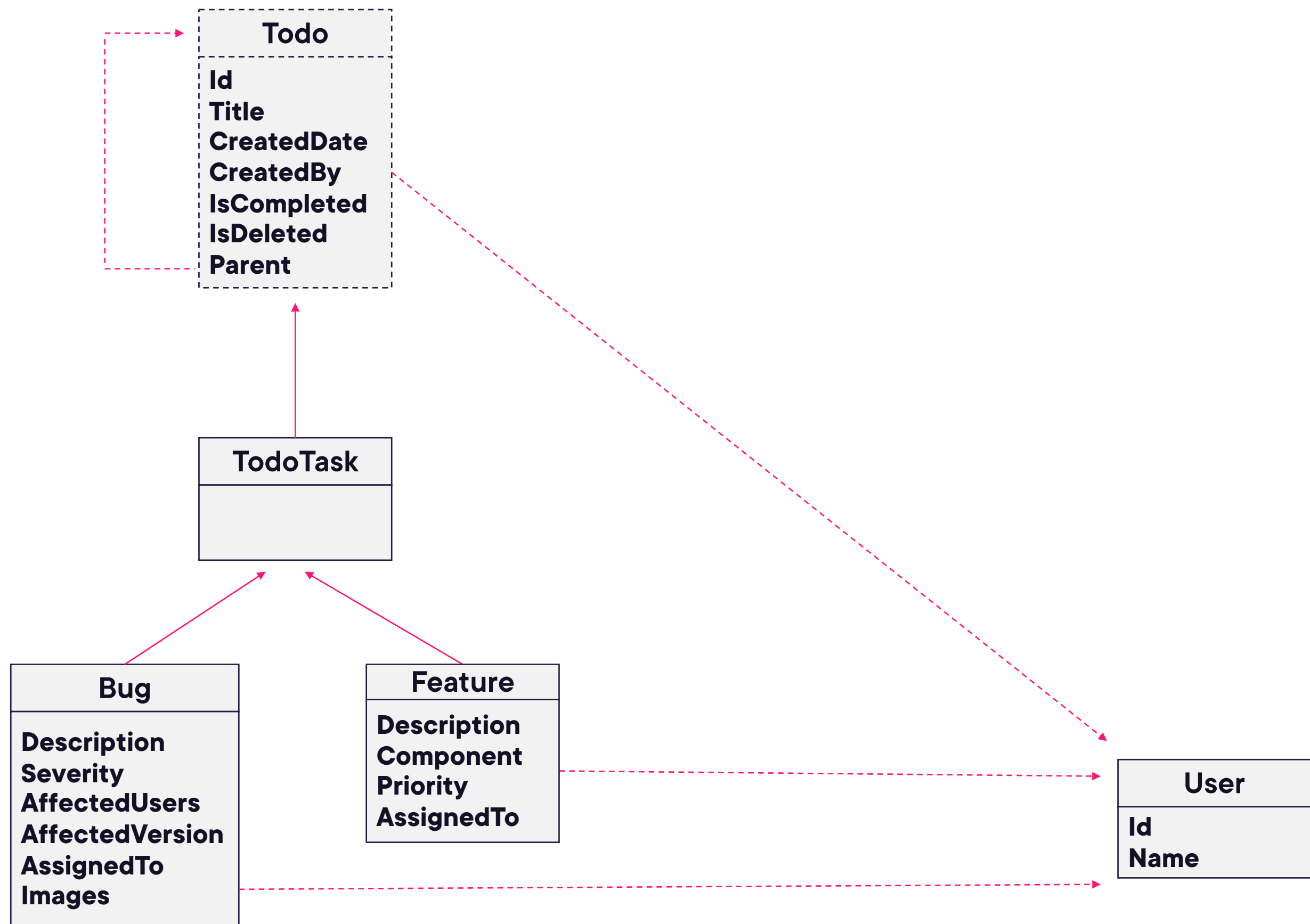
Globomantics ToDo



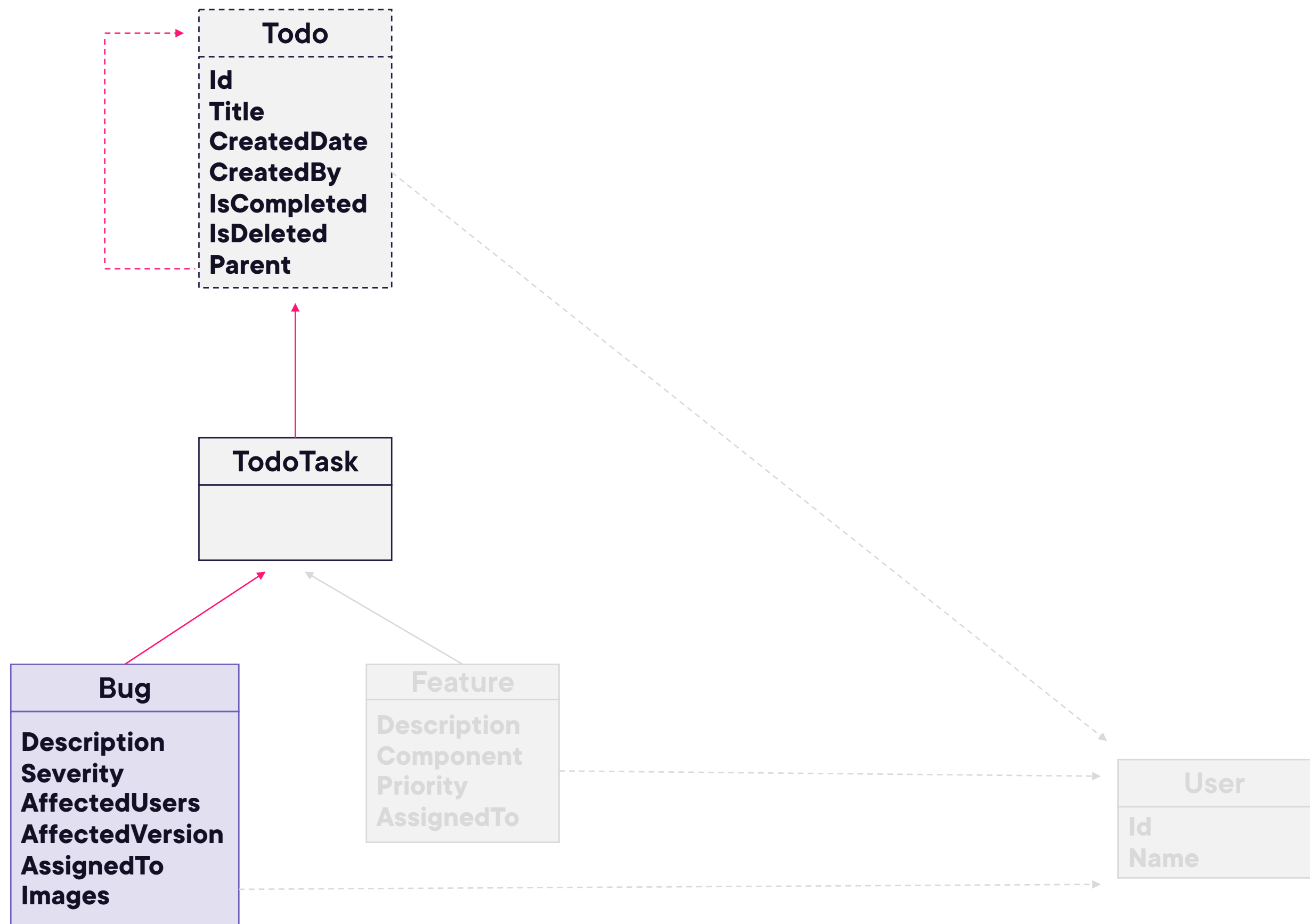
Globomantics ToDo



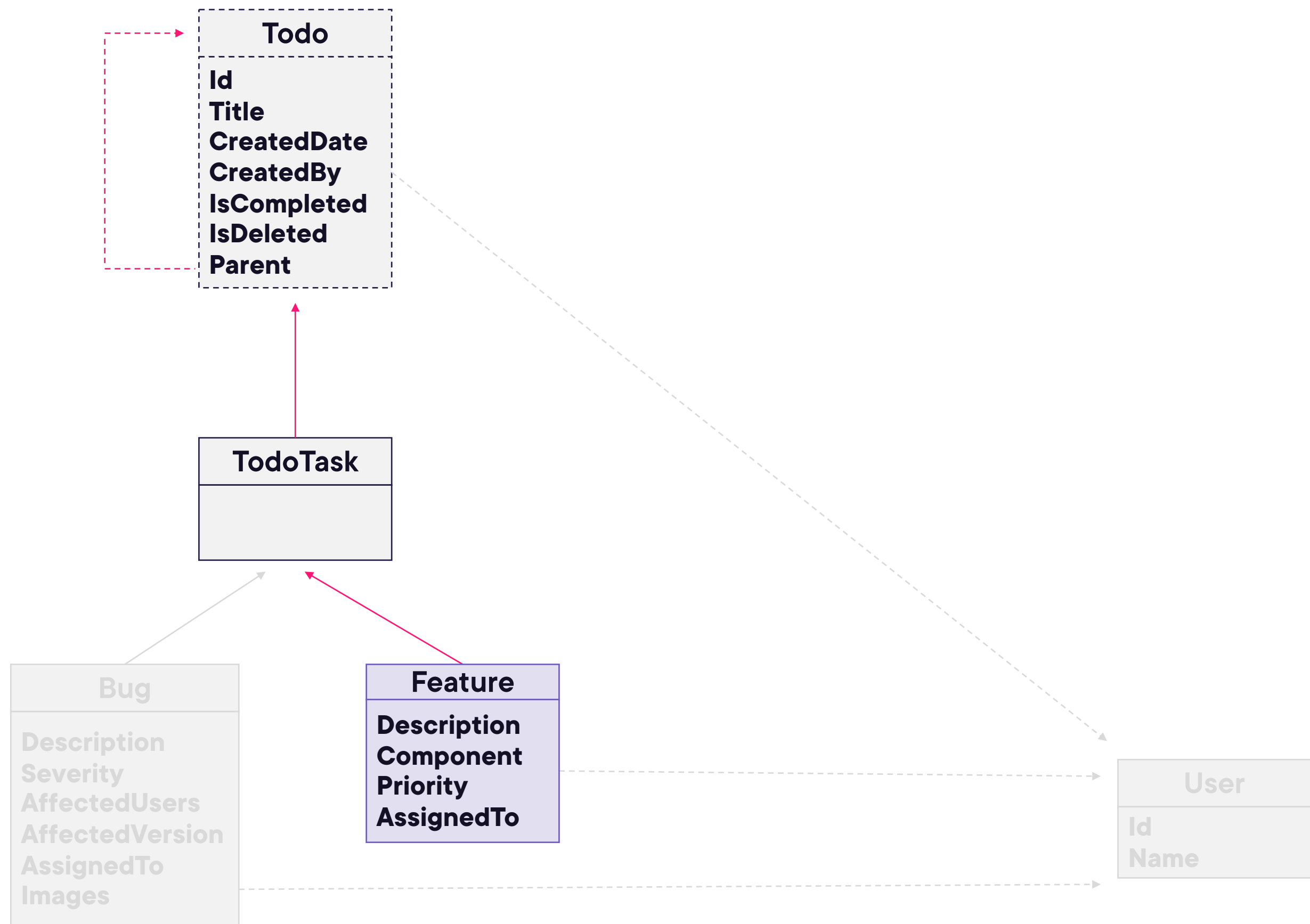
Globomantics ToDo



Globomantics ToDo



Globomantics ToDo



Using Inheritance

```
abstract record Todo(Guid Id,  
    string Title,  
    DateTimeOffset CreatedDate,  
    User CreatedBy,  
    bool IsCompleted = false,  
    bool IsDeleted = false)  
{  
    public Todo? Parent { get; init; }  
}
```

```
record TodoTask(string Title, DateTimeOffset DueDate, User CreatedBy)  
    : Todo(Guid.NewGuid(), Title, DateTimeOffset.UtcNow, CreatedBy);
```



**You will keep improving the
application as you go!**

**Just as you will in real-world
applications.**



You **DO NOT** have to know
WPF or UI programming to
follow along!



What We Are Going to Add

Business logic

Repositories

Domain models

Data entities

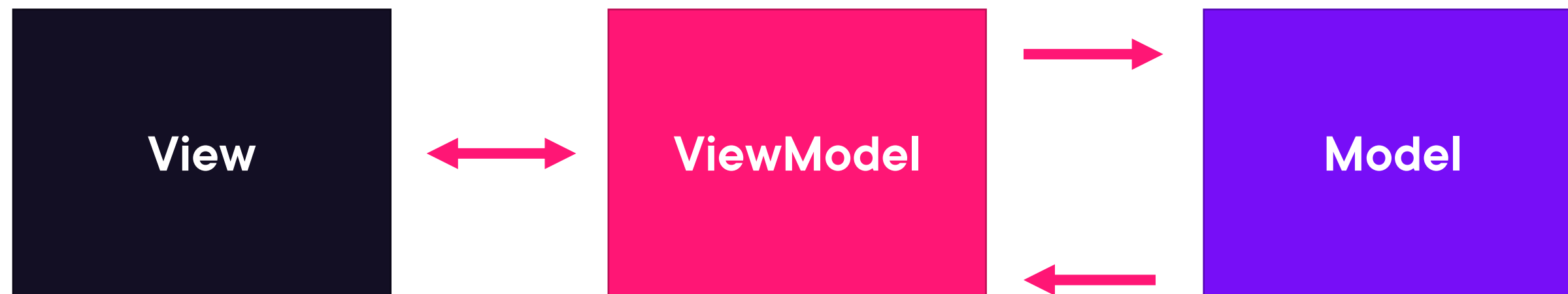
Database interaction

**Patterns & best
practices**



Model-View-ViewModel (MVVM)

“An architectural pattern in computer software that facilitates the separation of the development of the graphical user interface (GUI; the view)—be it via a markup language or GUI code—from the development of the business logic or back-end logic (the model) such that the view is not dependent upon any specific model platform.”



Separating UI from Business Logic using MVVM

**View
(BugView)**

**Connect to the
ViewModel to
interact with the
Model**

**ViewModel
(BugViewModel)**

**Load, Create,
Validate Bug**

**Model
(Bug)**

Title, Description



ObservableObject

“The ObservableObject is a base class for objects that are observable by implementing the INotifyPropertyChanged and INotifyPropertyChanging interfaces.

It can be used as a starting point for all kinds of objects that need to support property change notifications.”



**You will use the C#
language features you have
learnt about.**

**To fill in the missing parts of
the application!**



Relying on the Delegate or Action

```
class MainViewModel
{
    public Action<string>? ShowAlert { get; set; }

    public void Import()
    {
        ShowAlert?.Invoke("This method will be available later!");
    }
}
```



Relying on the Delegate or Action

```
class MainViewModel
{
    public Action<string>? ShowAlert { get; set; }

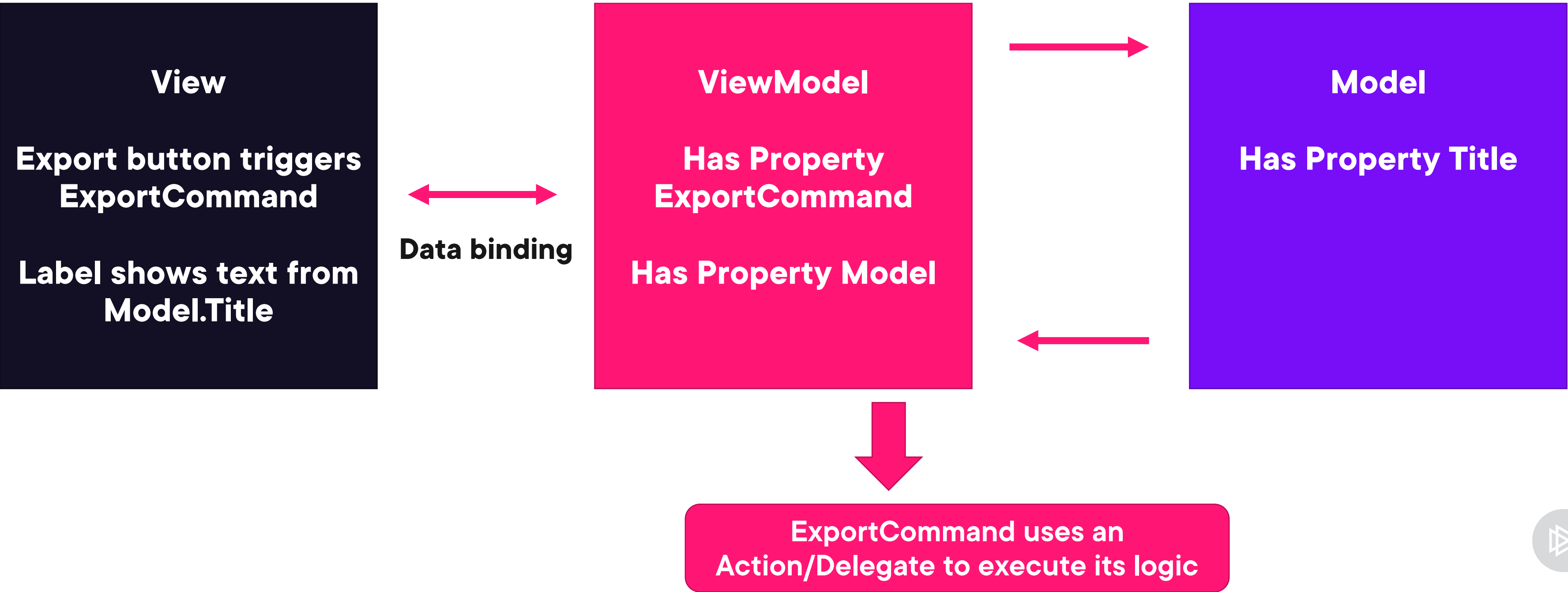
    public void Import()
    {
        ShowAlert?.Invoke("This method will be available later!");
    }
}

class MainWindow
{
    public MainWindow(MainViewModel mainViewModel)
    {
        InitializeComponent();

        mainViewModel.ShowAlert = (message) => MessageBox.Show(message);
    }
}
```



Consuming Properties from the ViewModel in the View



Inject Implementations of Interfaces

```
class MainViewModel
{
    public MainViewModel(IRepository<TodoTask> todoRepository)
    {
    }
}
```



No need to know about the
actual implementation!



Creating a ViewModel without a View

```
class MainViewModel
{
    public MainViewModel(IRepository<TodoTask> todoRepository)
    {
    }

    public void Import() { }
}

var model = new MainViewModel(...); // Pass a fake IRepository?

model.Import(); // Test to see that it does not throw exceptions
```

