Short introduction to the C# MAUI.NET / ASP.NET Sample Application

Dr. Georg Hackenberg BSc MSc

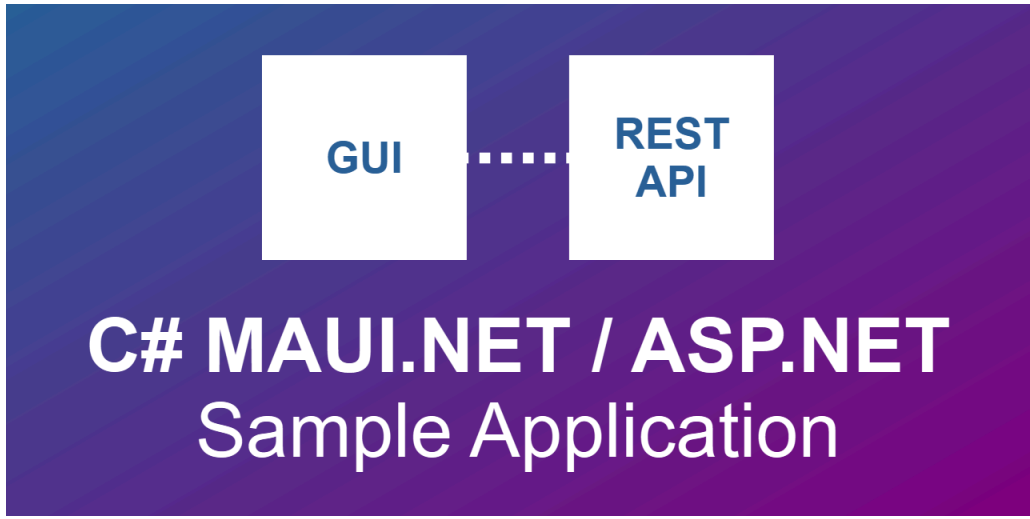
Professor for Industrial Informatics | School of Engineering
University of Applied Sciences Upper Austria
Stelzhamerstr. 23, 4600 Wels, Austria

👤 https://github.com/ghackenberg
👤 https://linkedin.com/in/georghackenberg
👤 https://youtube.com/@georghackenberg

Also check out https://mentawise.com and https://caddrive.org 😎

Deck overview

- **Section 1 - Basics** you should know before diving deeper
- **Section 2 - The `CustomLib` project** containing common classes
- **Section 3 - The `CustomApi` project** implementing the backend
- **Section 4 - The `CustomApp` project** implementing the frontend



Section 1 - Basics

Domain model and **package structure**

Domain model

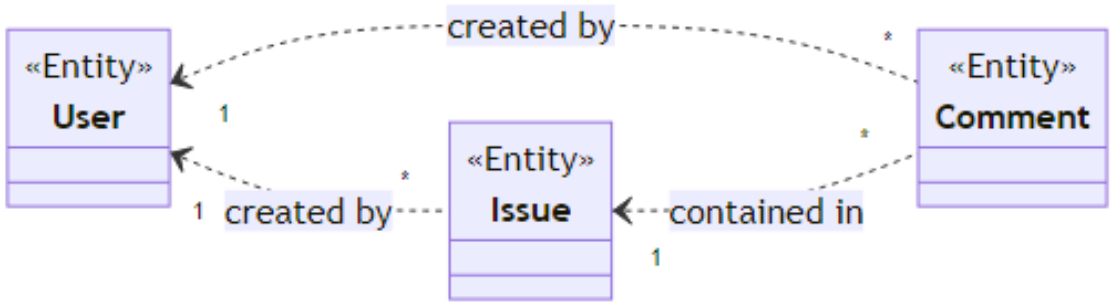
Coming soon

Domain model overview

The diagram on the right shows the **domain model** implemented by the application.

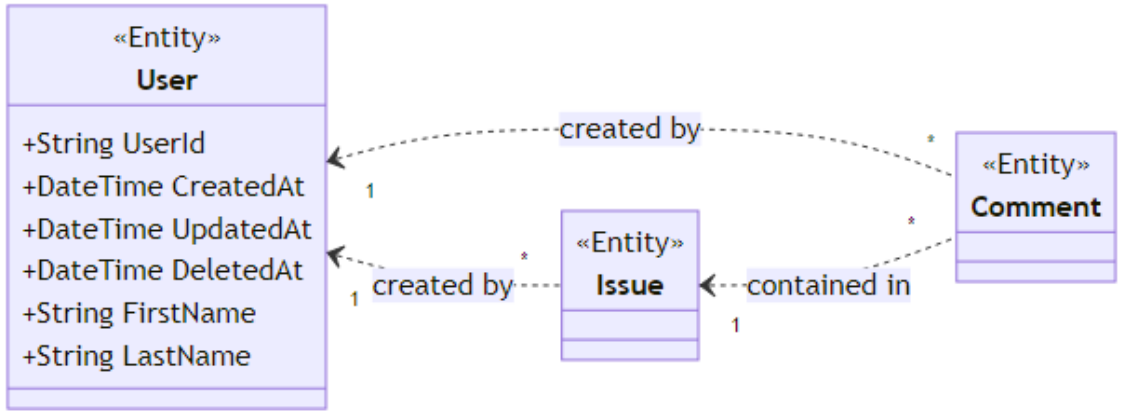
The data model consists of **three entities**, namely `User`, `Issue`, and `Comment`.

Issues and comments are `created by` users, comments are `contained in` issues.



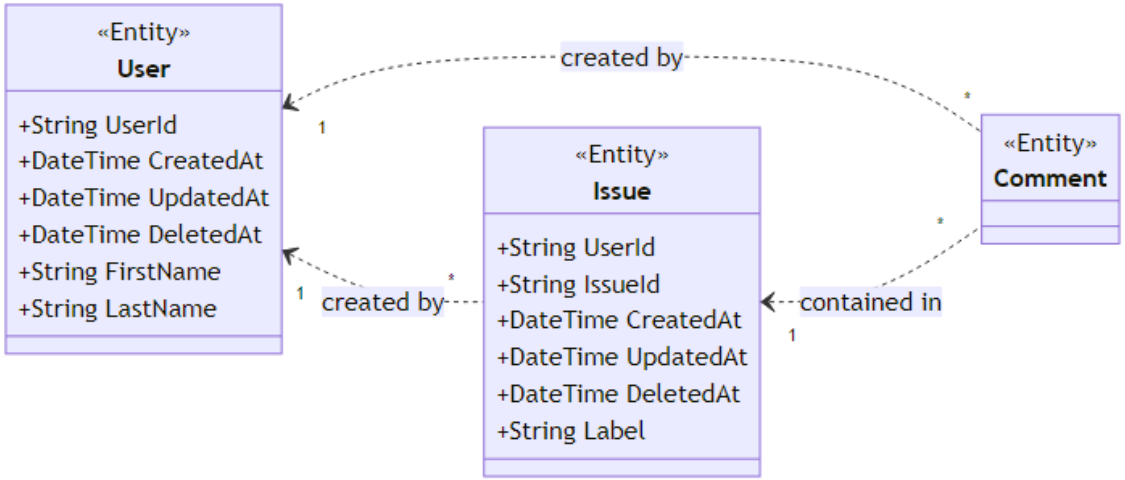
The `User` entity

Coming soon



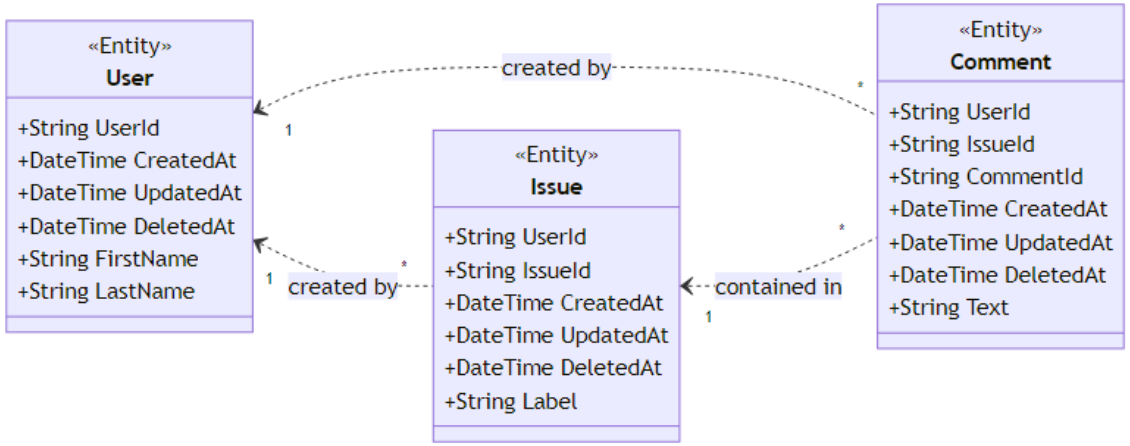
The `Issue` entity

Coming soon



The `Comment` entity

Coming soon



Project structure

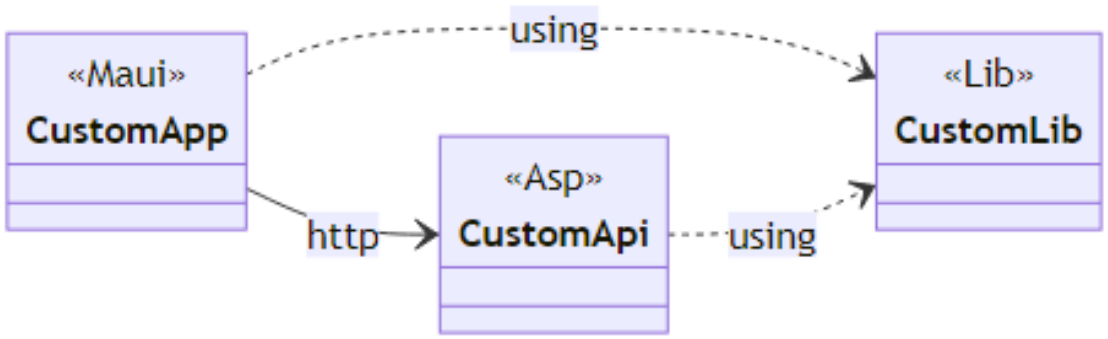
Coming soon

The project structure

The sample application comprises **three code projects**: `CustomLib`, `CustomApi`, and `CustomApp`.

The `CustomLib` provides a common **class library** used by the other two projects.

The `CustomApi` implements the **backend**, the `CustomApp` the **frontend**.



Section 2 - The `CustomLib` project

Messages, **exceptions**, and **interfaces**

Messages

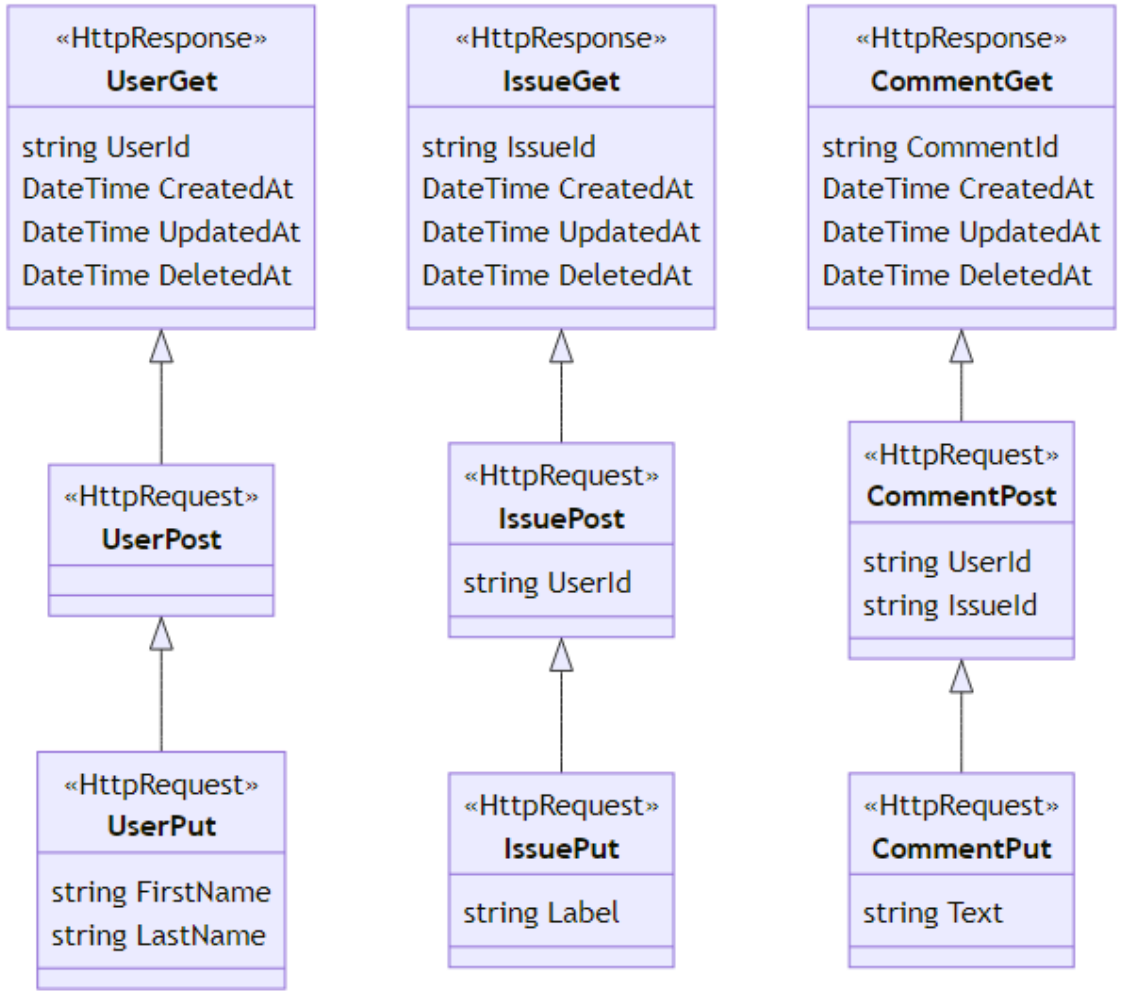
Data exchange between frontend and backend

Message overview

The REST API uses request and response **messages** for working with these entities.

For **each resource** (i.e. user, issue, comment) we distinguish `Get`, `Post`, and `Put` messages.

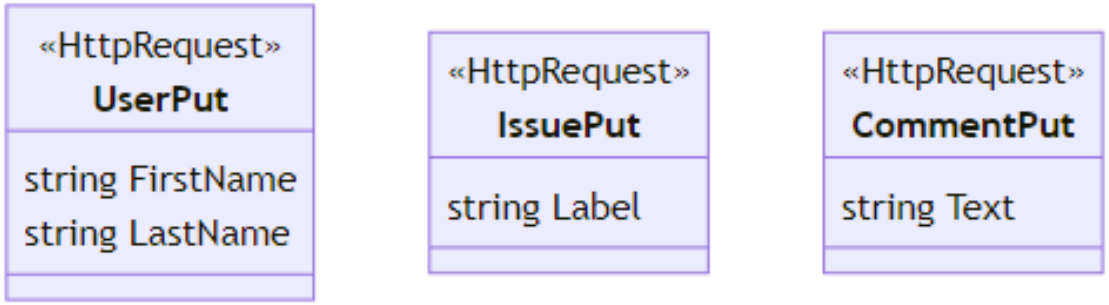
In the following, we describe each **type of message** in more detail including their data fields.



`Put` messages

The `Put` structures contain the fields that you can **override later** after creating an instance.

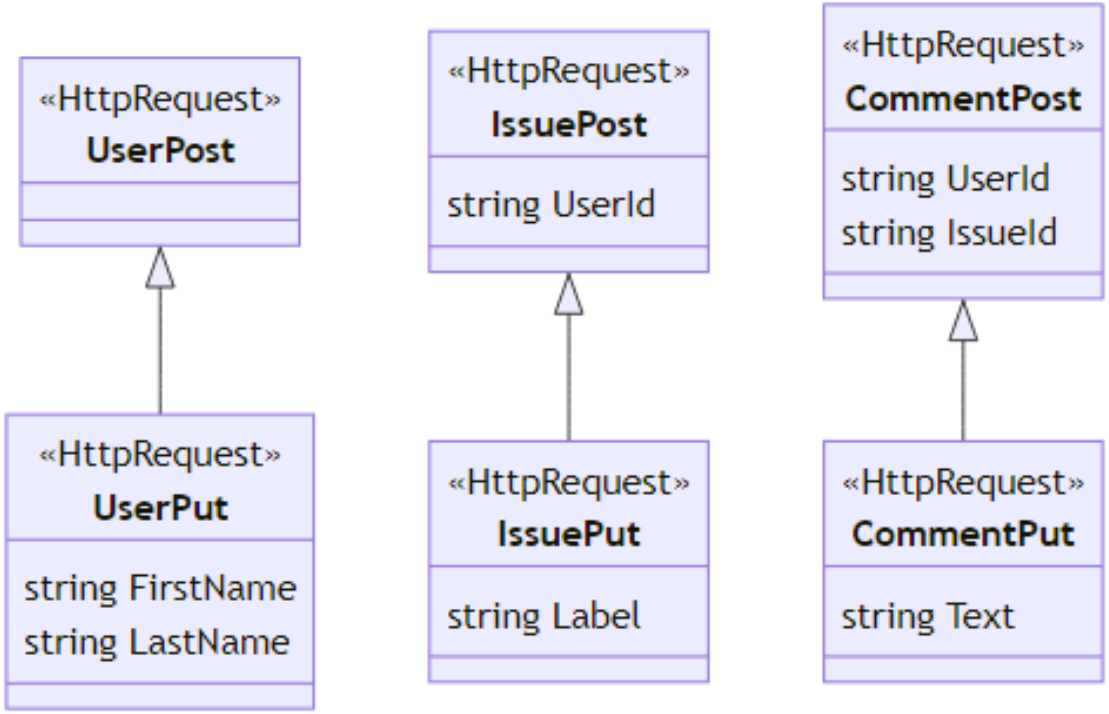
More coming soon



`Post` messages

The `Post` structures derive from the `Put` structures and add the fields that you can **set only initially** when creating an instance such as entity references.

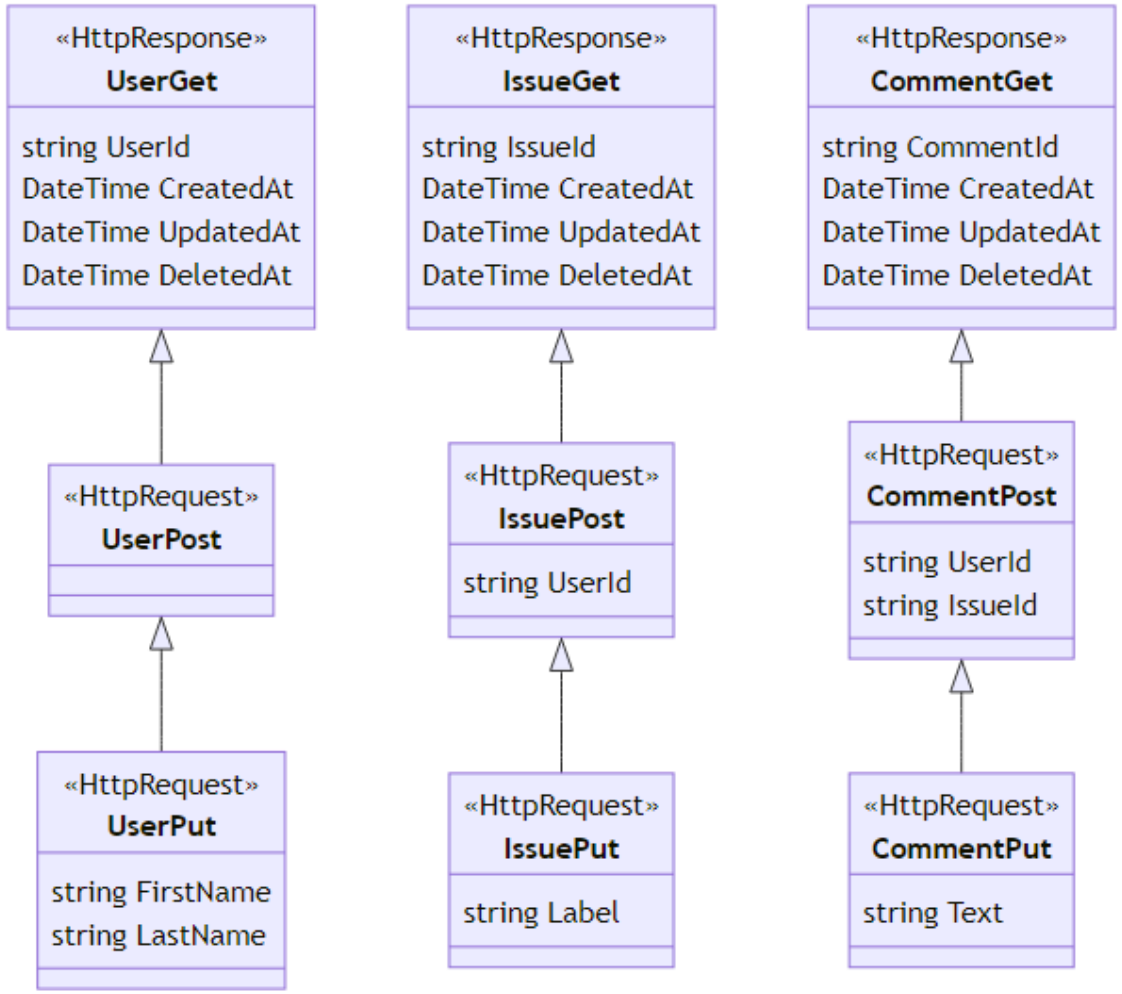
More coming soon



`Get` messages

Finally, the `Get` structures derive from the `Post` structures and add the fields that are **read only** such as instance identifiers and timestamps.

More coming soon



Exceptions

Problems during service execution

Exception overview

Coming soon

Interfaces

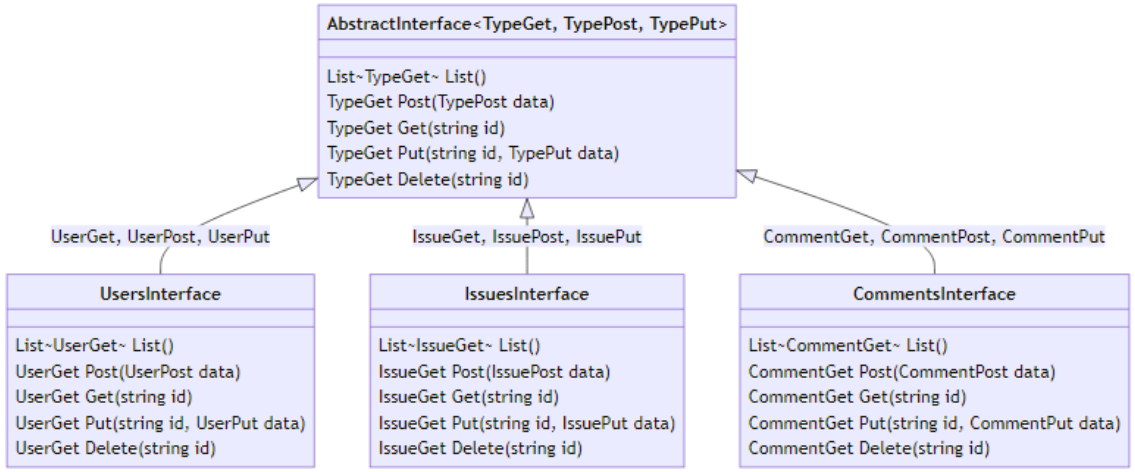
Methods *provided by* backend and *required by* frontend

Interface overview

Based on the message data structures we **define the methods** of the REST API.

We use a **generic interface** model including `List`, `Post`, `Get`, `Put`, and `Delete` methods.

In the following, we explain each method **in more detail** including inputs and outputs.

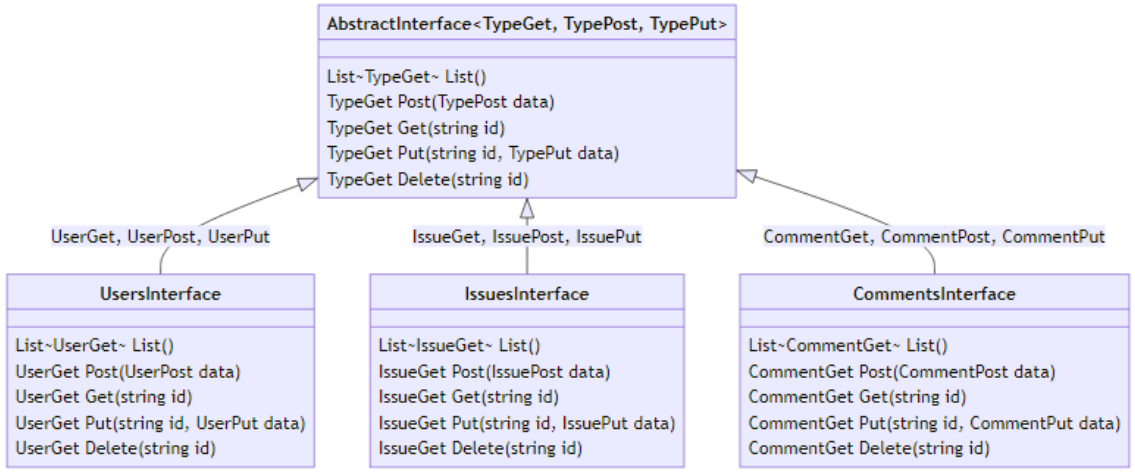


The `List` method

The `List` method returns a **collection** of created (and *not* deleted) instances.

Note that in our case the method **does not require** any input parameters.

*Usually the input parameters are used for **filtering and paging** the instances.*

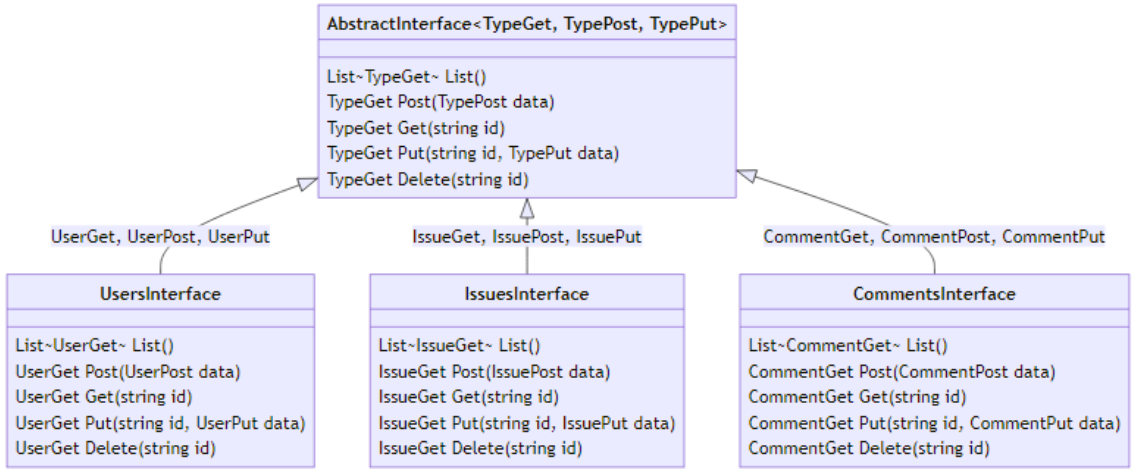


The `Post` method

The `Post` method **creates and returns** new instances of a given entity type.

The **input parameters** use the corresponding `Post` message defined previously.

The **return type** corresponds to the respective `Get` message from before.

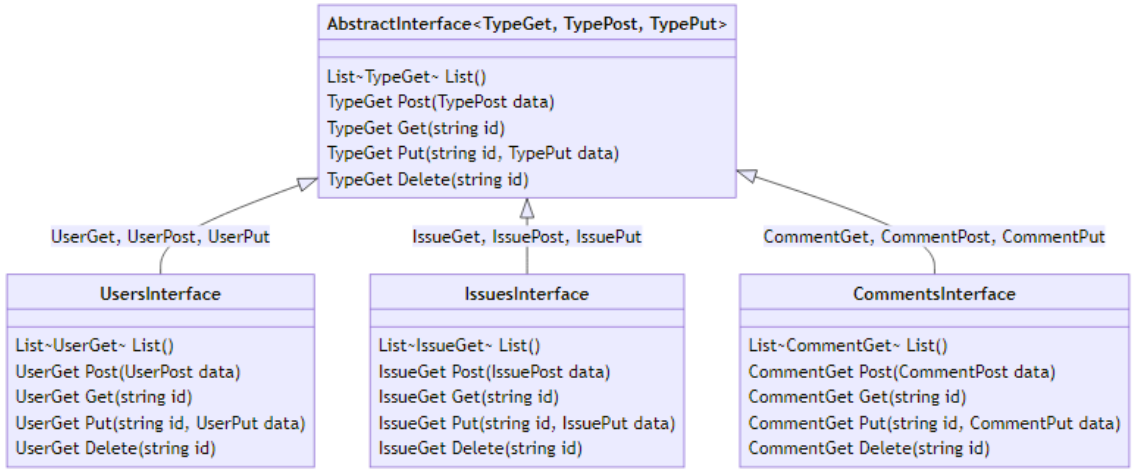


The `Get` method

The `Get` method **returns** an existing instance with a given identifier.

The **single input parameter** represents the identifier of the desired instance.

The **return type** corresponds to the respective `Get` message from before.

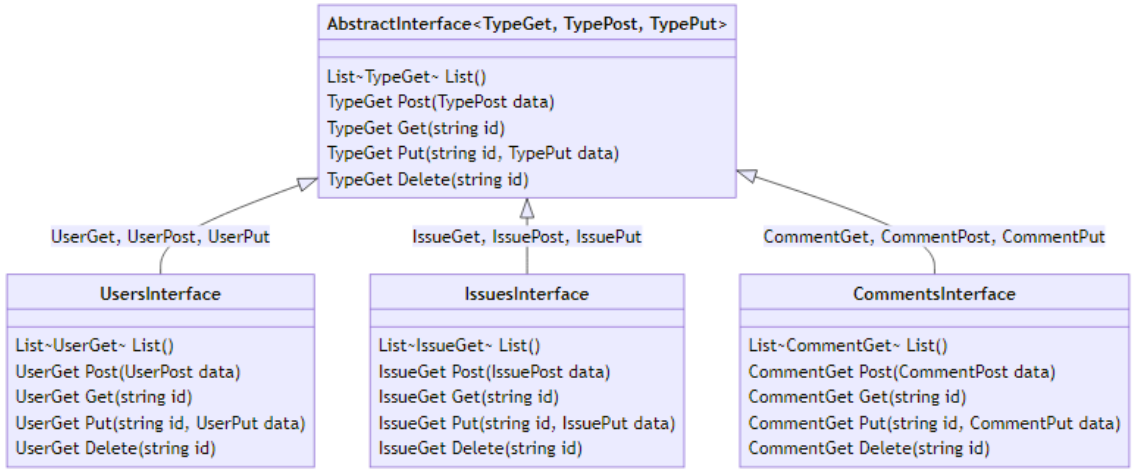


The `Put` method

The `Put` method **overrides and returns** an existing instance with a given identifier.

The **two input parameters** are the identifier of the instance and the respective `Put` message.

The **return type** corresponds to the respective `Get` type as introduced before.

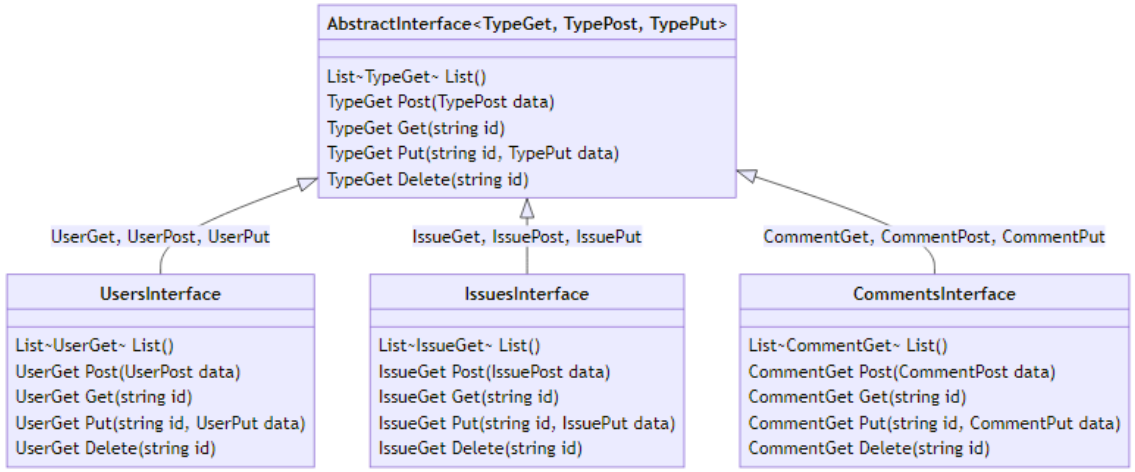


The `Delete` method

Finally, the `Delete` method **deletes and returns** an existing instance with a given identifier.

Note that deleting an instance **does not remove** the dataset from the database.

Instead, the `DeletedAt` timestamp of the instance is **set to the current timestamp**.



Section 3 - The `CustomApi` project

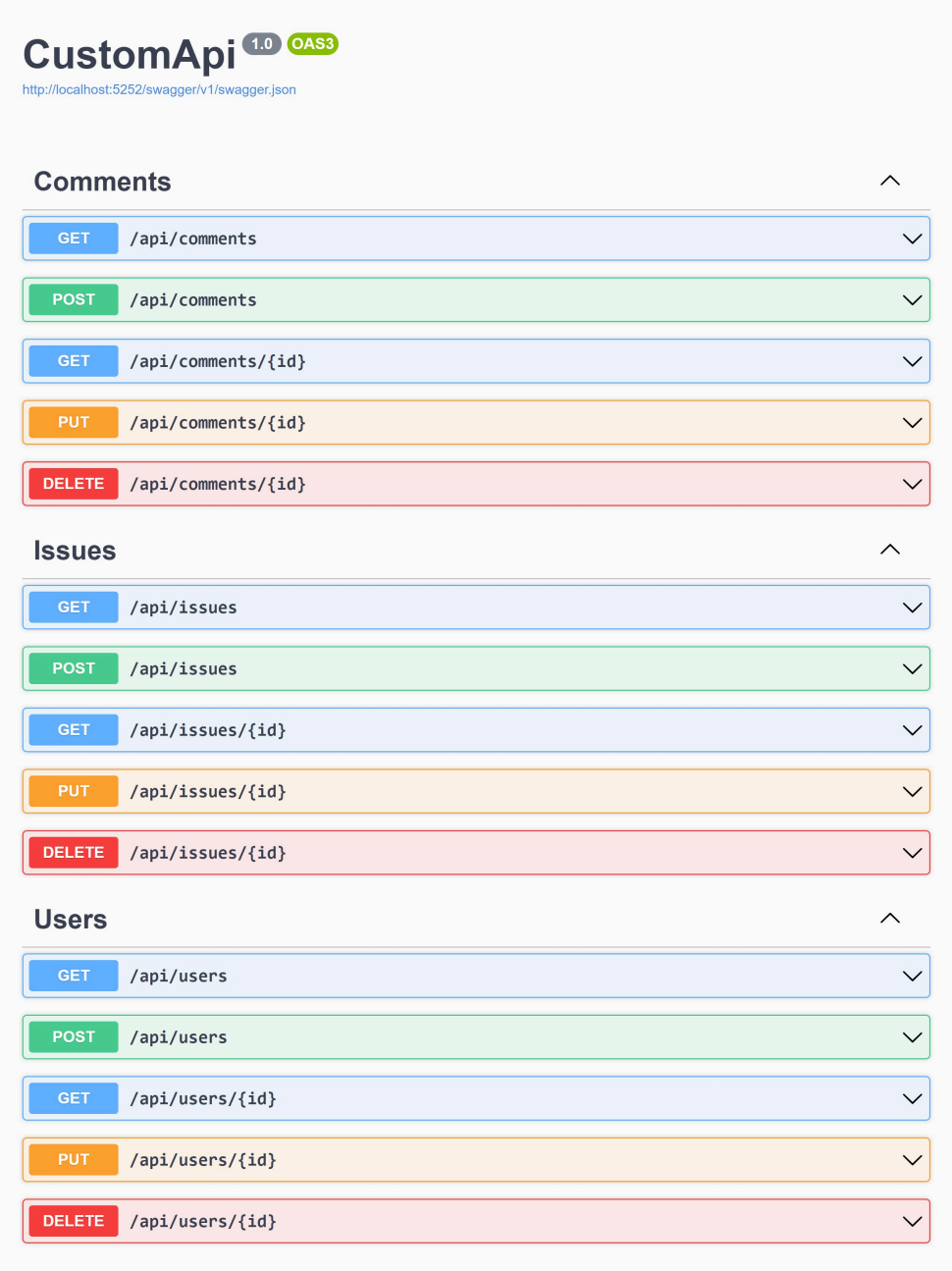
ASP.NET backend and **controllers**

ASP.NET backend

The data itself is managed by an **ASP.NET backend** service with standard REST API.

The screenshot on the right provides an **overview** of the API services exposed.

For each **resource** (i.e. user, issue, comment), the same set of functions is defined.



Controllers

Coming soon

Controller overview

Coming soon

Section 4 - The `CustomApp` project

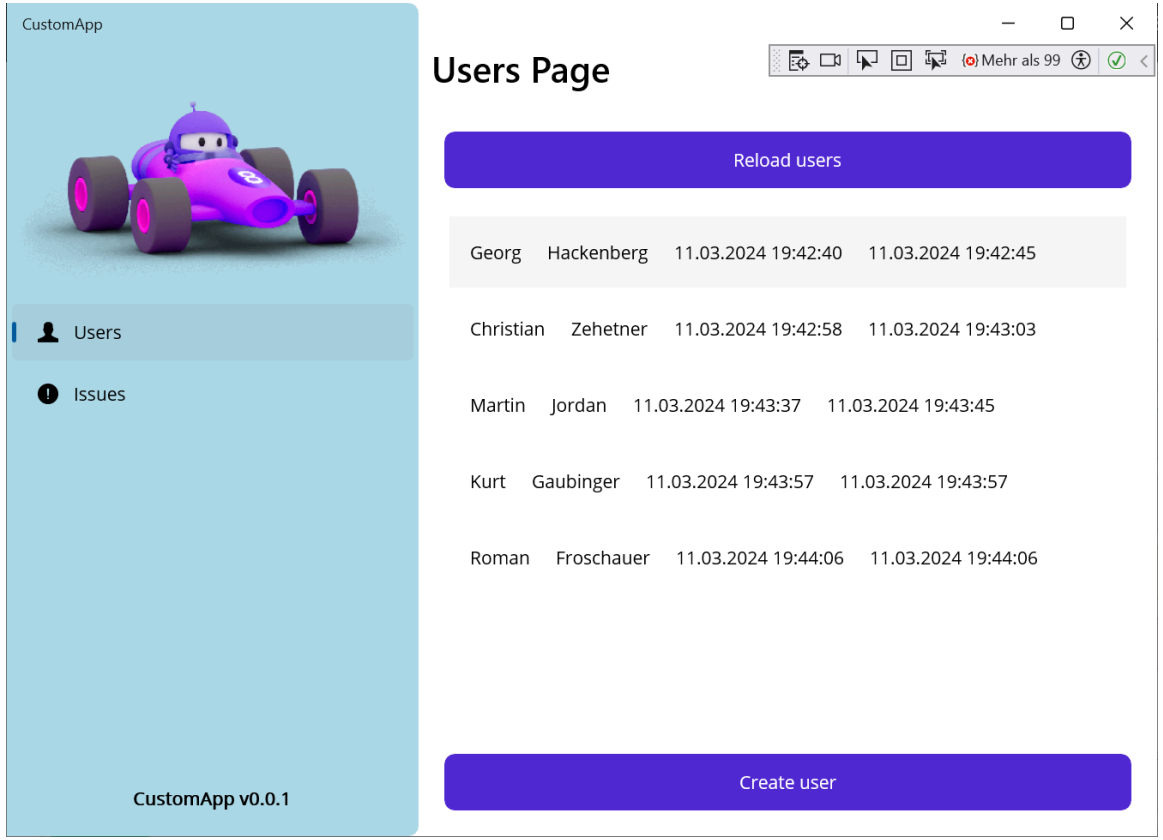
MAUI.NET frontend, **view models**, and **pages**

MAUI.NET frontend

The C# MAUI.NET / ASP.NET Sample Application features a **basic** graphical user interface (GUI).

With the GUI you can manage the **users** and the **issues** stored in the underlying database.

The screenshot on the right shows the **users page** listing all created user entities.

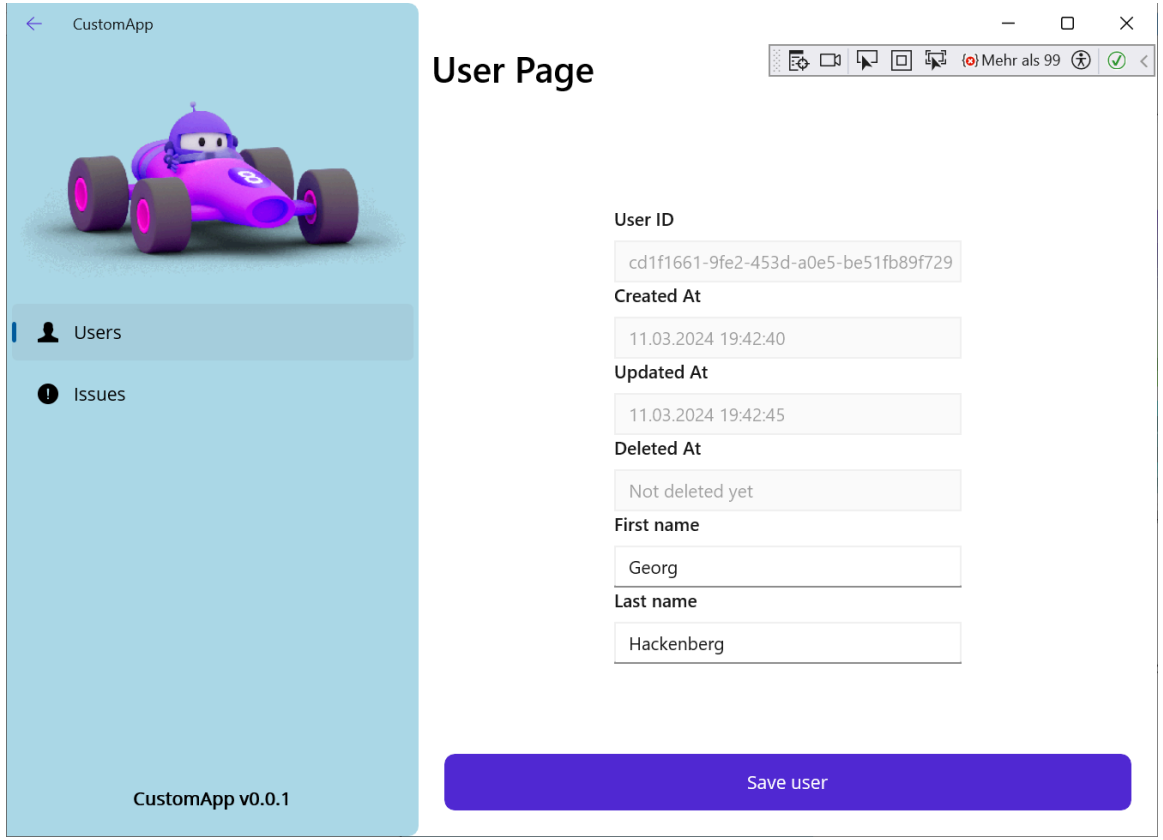


MAUI.NET frontend (cont'd)

When clicking an existing user or creating a new user, you enter the **user detail** page.

The user detail page shows all the **data associated** with a user entity in the database.

You can change the **first and last name**, the other fields are set automatically.

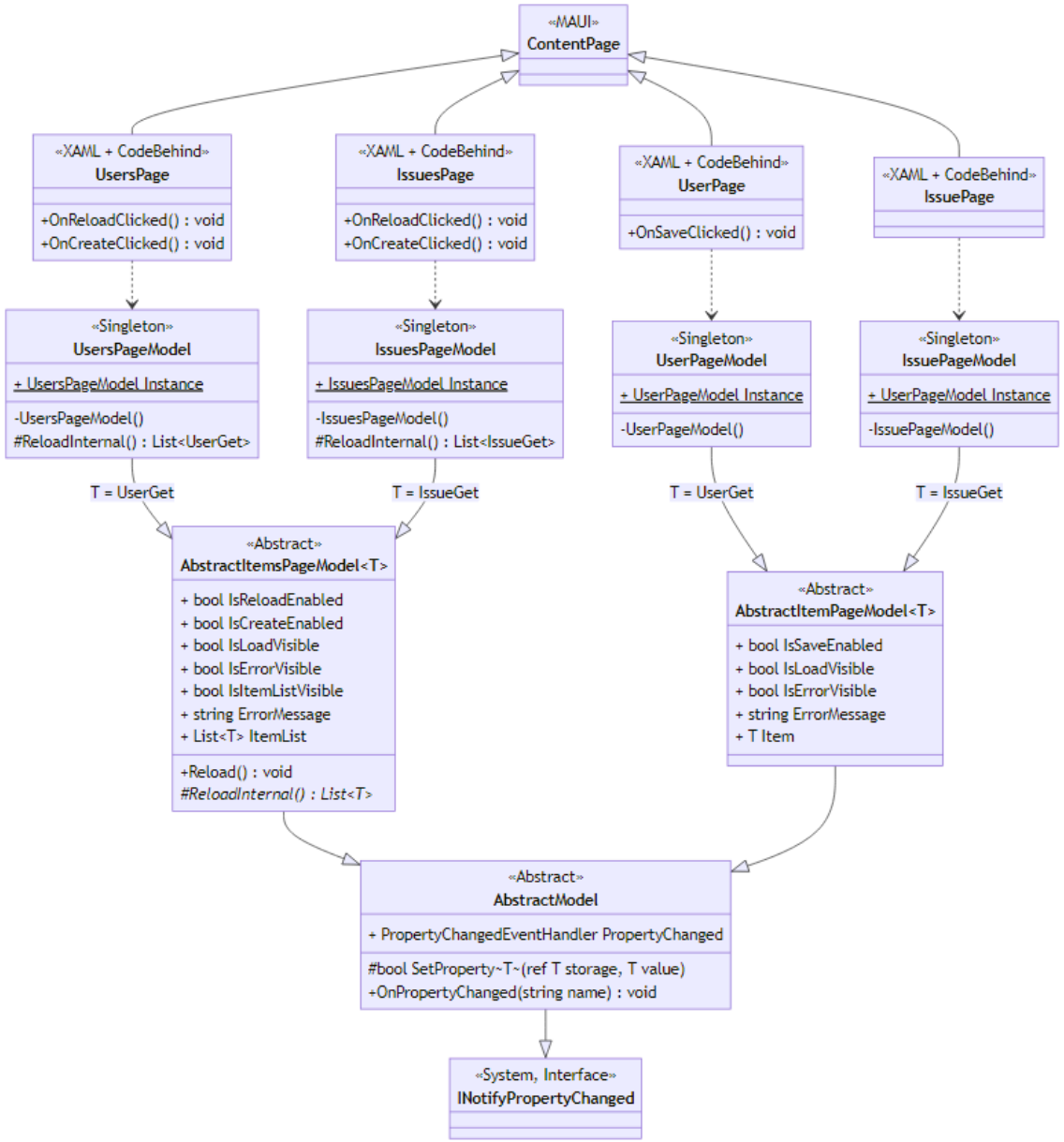


View models

Coming soon

View model overview

Coming soon

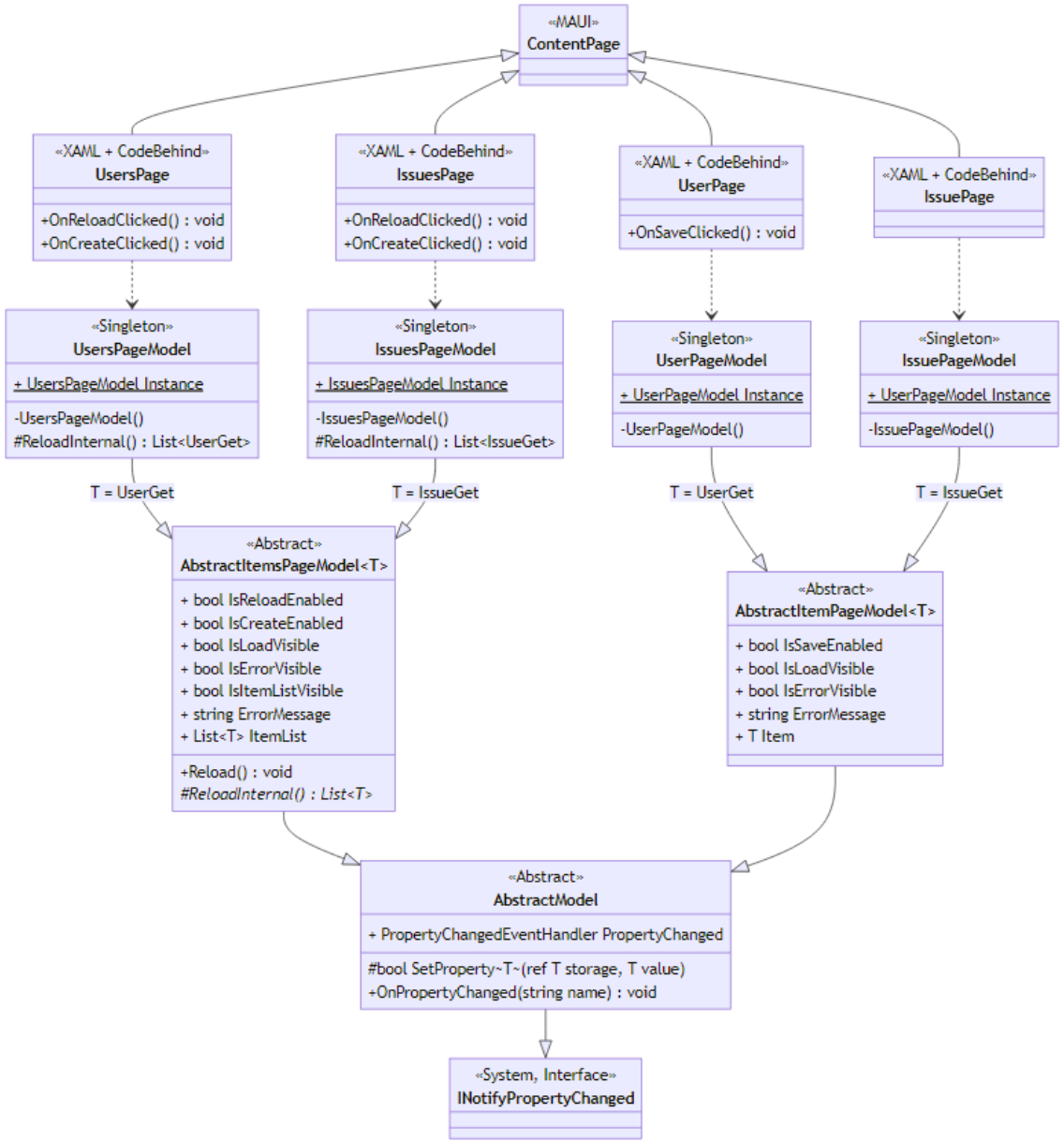


Pages

Coming soon

Page overview

Coming soon



You are ready to code 👩‍💻

Well done!