Horváth Lóránt

TIMETABLE DESIGNER

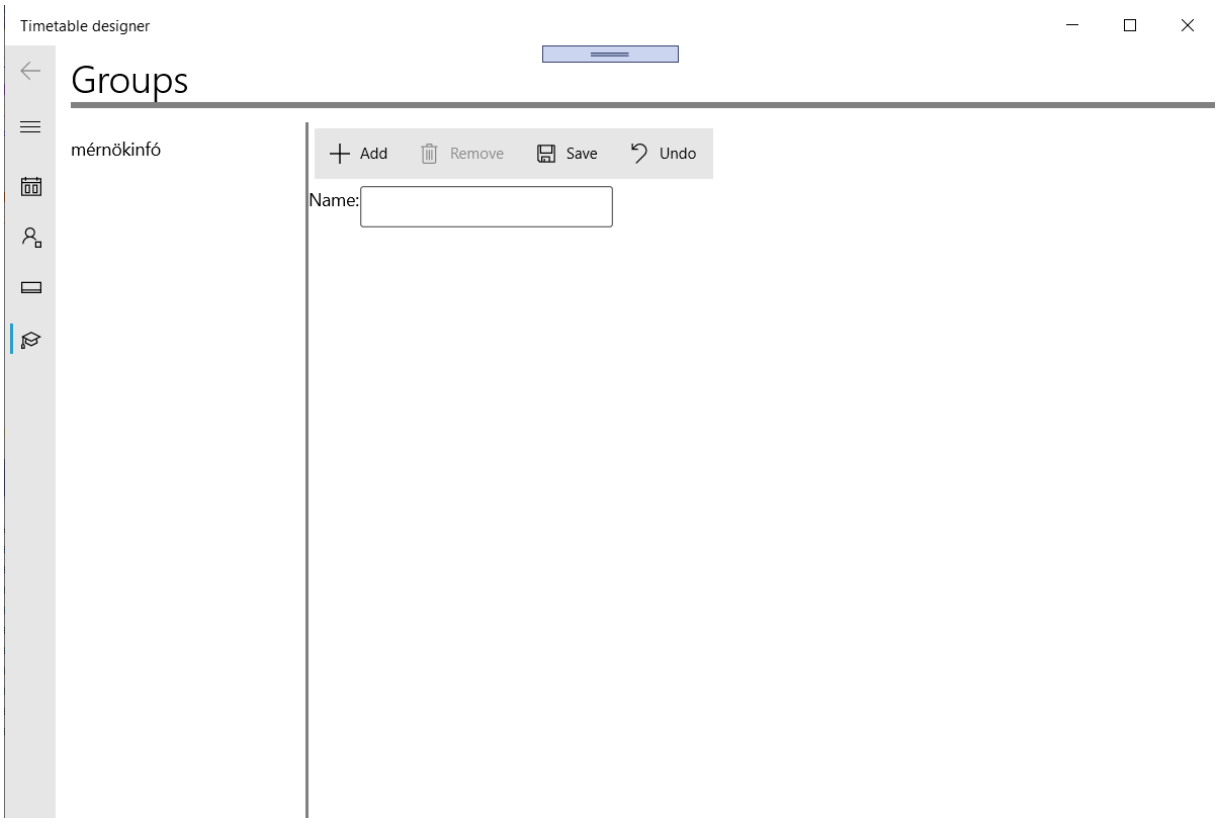
Github: https://github.com/horlor/TimeTableDesigner

User Manual

With this application you can design a Timetable like the one used in elementary and secondary schools. In this application you will have to manage four types of data: teachers, subjects, groups, and courses. The software will make sure, that you can only add valid versions of these (for example you can't make a course with a teacher, who does not teaches its subject).

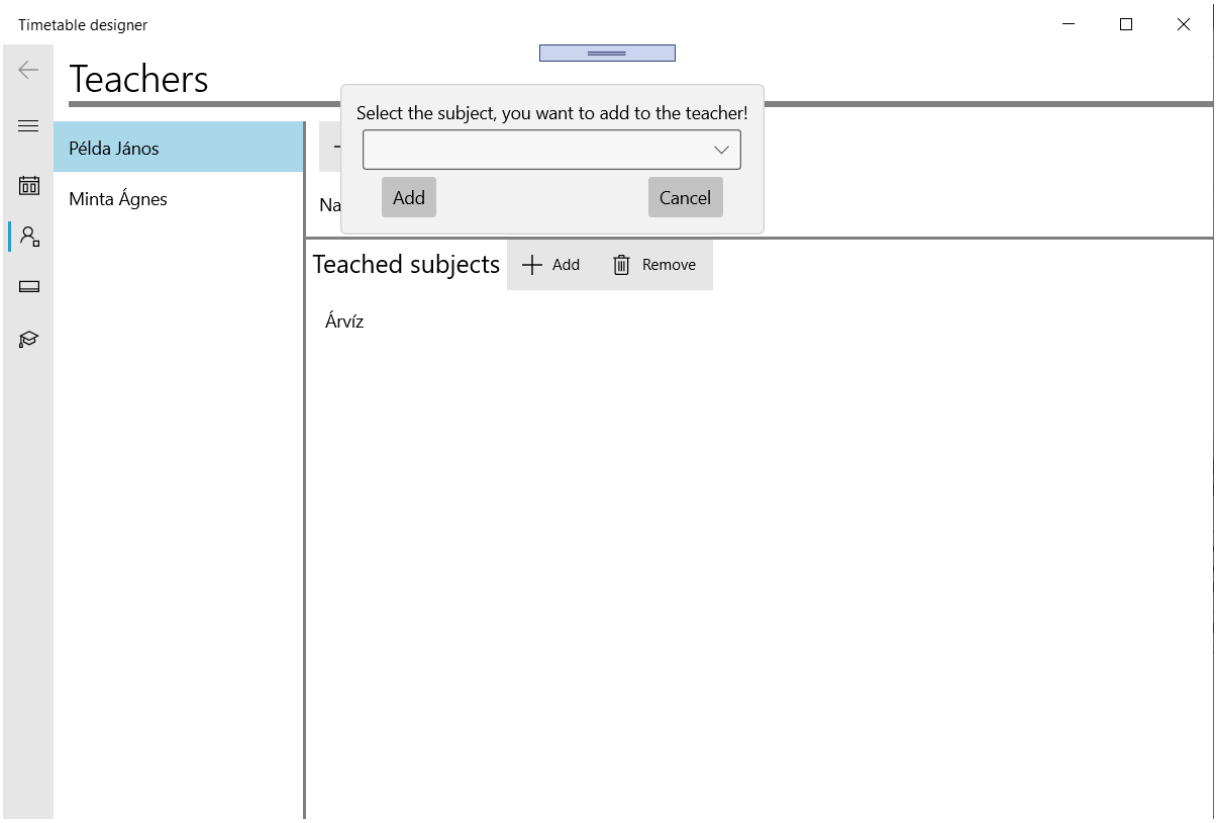
At first you have to make teachers, subjects and groups before you can design the timetable with the actual courses. You can choose these functions from the Menu on the left (with the Hamburger icon, you can open it).

On the Group page you can manage your groups.



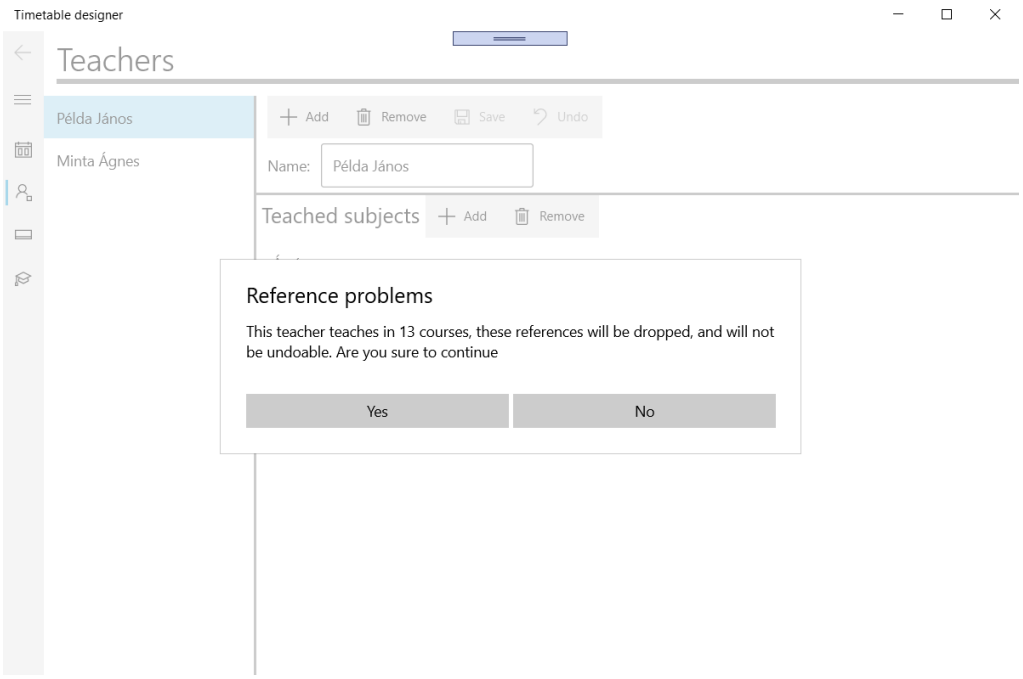
With the Add button you can add a new Group to your application, with the remove you can delete the selected Group (chosen in the list on the left). With the Save and Undo button you can make changes to the selected Group.

The modifications and functions on the Subject's Page is exactly the same like the Group's one.

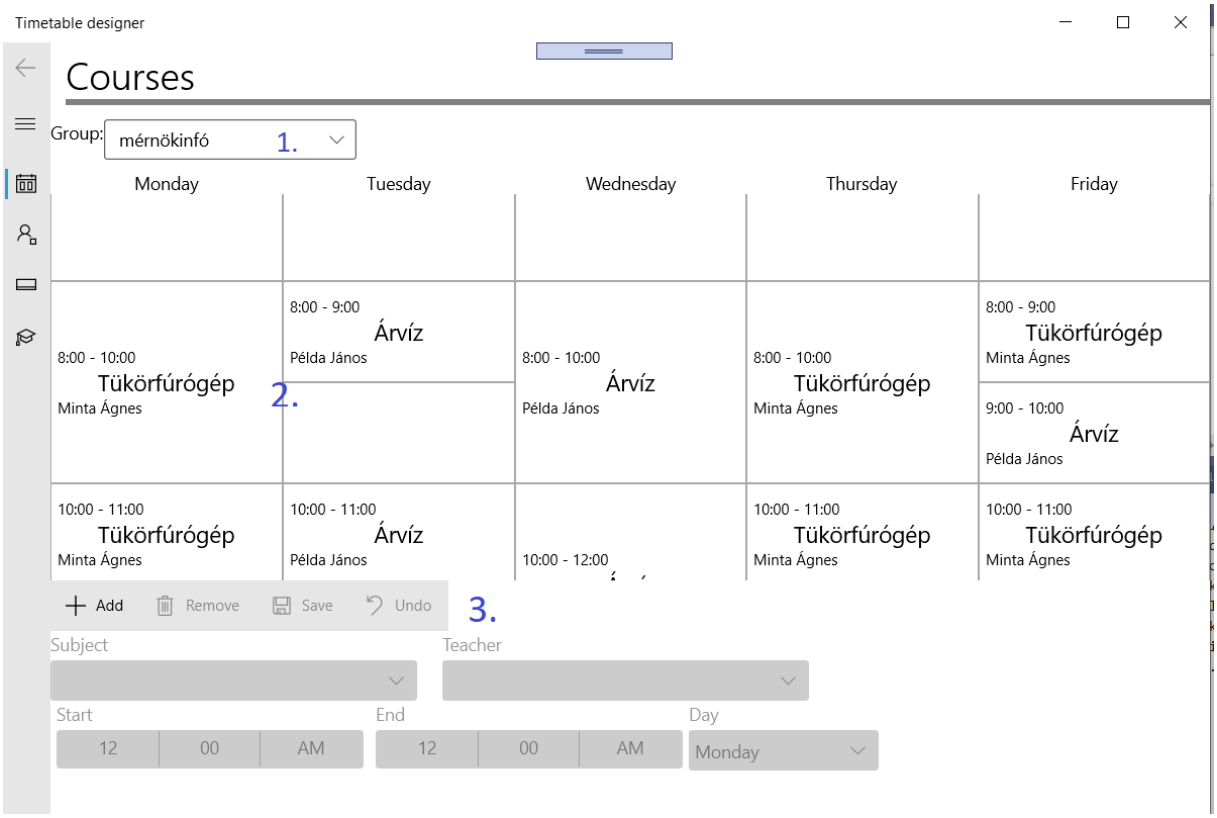


The Teacher Page is similar in functionality to the previous ones, but it has another list with the teached subjects. You can edit this with the two button besides it. If you add one you have to choose the wanted subject in a little popup window.

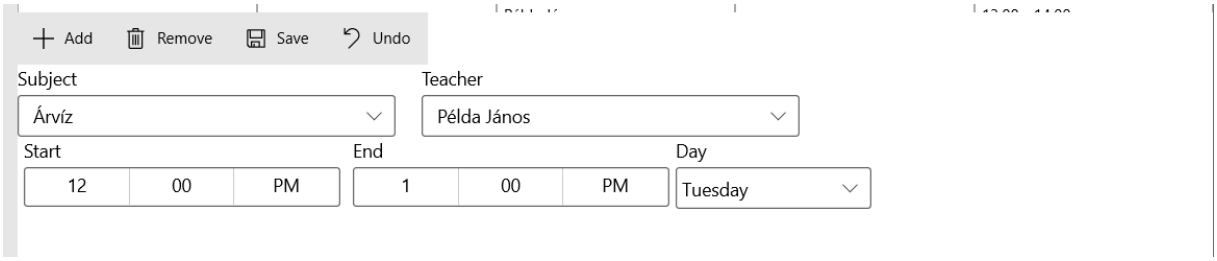
During using these functions you can made changes that can have serious impact on the data, but then you'll be warned in dialogs, so you can make sure that you'll want to make that change.



The Course designer

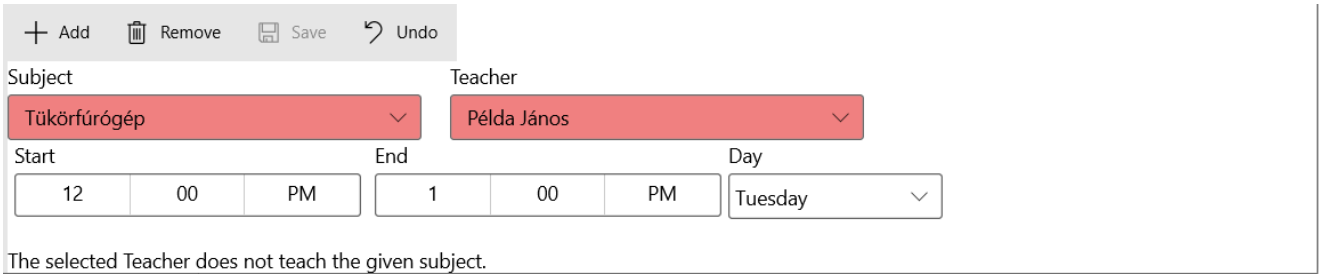


This page contains three main element. The first one is a drop down list with you can choose which group's timetable you want to edit. The second one is a visual representation of the timetable, with you can make changes to it much easier. The third one is an editor to the selected course, we will continue this tutorial, with it.



This editor bar consists buttons similar to the ones on the other page, the only important change, that after using the Add button you must use the Save button as well, without it the new course will not be saved.

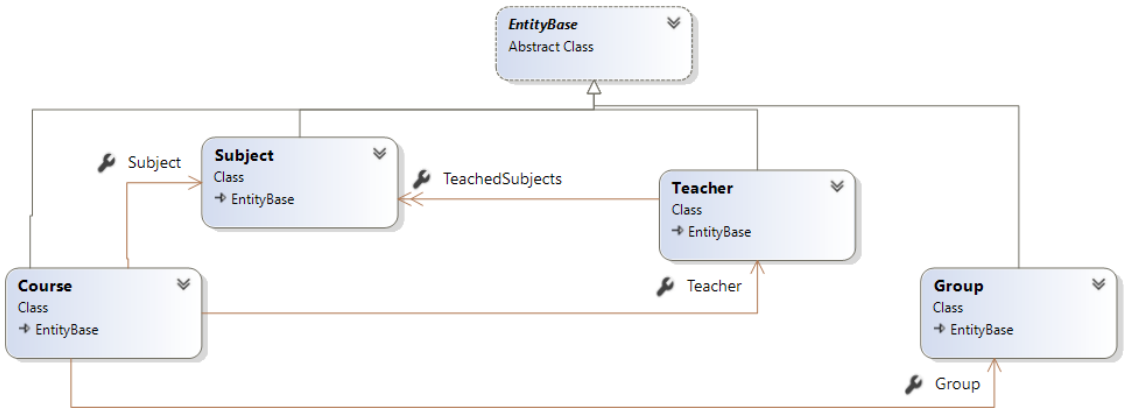
You can use its Time pickers and drop down lists to make the course have the wanted properties. It also gives you a real-time validation to make your work easier – it will give you a message about the problem and mark the involved properties with red background, for an example:



The Architecture of the Application

The application's architecture consists three main layers, (each represented in an Visual Studio Project, although it is not a true three layered application, it is done to make its main services loosely coupled. The layers main responsibilities are persistence, model or business logic, and frontend. The whole architecture is domain driven, so the persistence and the frontend is dependent on the logic. Now let's look those layers.

The domain



The model consists four types of entities: teachers, groups, subjects and courses. To make the handling easier and avoid code duplication all of them are inherited from an abstract EntityBase class.

As the app's goal to help to design a timetable, the main focus is on the Course objects. All Course objects have a reference on one Subject, Teacher and Group objects. Although on the class diagram it is not shown, the Teacher and Group associations are navigable on both sides. That means that the Group and Teacher objects have a collection about the Courses. To make sure this one to many association is valid, it can only be modified from the Course.

Smaller validations and changes are available from these classes, but for the most you will have to use a CourseManager object (it uses the Singleton pattern, you can get it from the class's Instance property. The CourseManager have two types of methods, one for validation, and one for modification, the main difference is that the modification (Change….) functions will change the given course object, and in case of invalid data, it will throw a TimetableException. The validating methods will only return with a list of the validity causes, and will not

change the Course object (this list is also provided in the TimetableException). The list consists an enum with the possible error types.

The persistence

As this app is domain driven, the persistence layer contains the implementation of the repository interfaces from the domain. Its classes are using JSON serialization for the model objects provided by Newtonsoft's Json.Net classes. The controller class is the one to write the serialized data to a file, so only one file will be the output. To make deserialization easier, there are JsonAttributes on the Model classes (because of the internal and private variables).

In the persistence project there are also Mock classes for testing purposes, they provide the same interface as the JSON ones, but they don't do any true serialization, they just store the objects in memory.

The Frontend

This application uses Microsoft's new Universal Windows Platform as a framework for providing the GUI. To make the UI loosely coupled with the Model classes, it uses the Model-View-Viewmodel design pattern in it (MVVM is greatly supported in UWP), but it not uses any third party framework for it.

In the ViewModel namespaces there are the used viewmodel classes, with the classicaly used naming conventions. It provides both wrapper classes for the model objects, and data and behavior holder for the UI elements. These classes are inherited from the ViewModelBase class, which implements the INotifyChanged interface and provides a function for its easy use, thus enabling the bindings used in the Xaml files.

The View namespace holds the pages, and the custom usercontrols. To follow the MVVM architecture they have the least Code Behind as possible, most functions are in the xaml files. The usercontrol classes are providing the UI element for showing a whole timetable, to make the use much easier.

As in the MainPage it can be seen, for the navigation the app uses the new NavigationView control to provide the drawer like hamburger menu, used widely in UWP apps. Its logic is in the MainPage's Code file.

The managing of the Model data and its conversion to ViewModels are the responsibility of the DataManager class (to segregate responsibilities it implements interfaces for only managing one kind of entities). It also makes sure the serialization and deserialization works with the UWP system (in a classical UWP app you don't have write rights to most of the filesystem).

There can also be found another helper classes: a CommandBase class for implementing the ICommand interface and make the Command properties and binding easier for viewmodels, and a few Converter classes, to make xaml files more logical and clear.