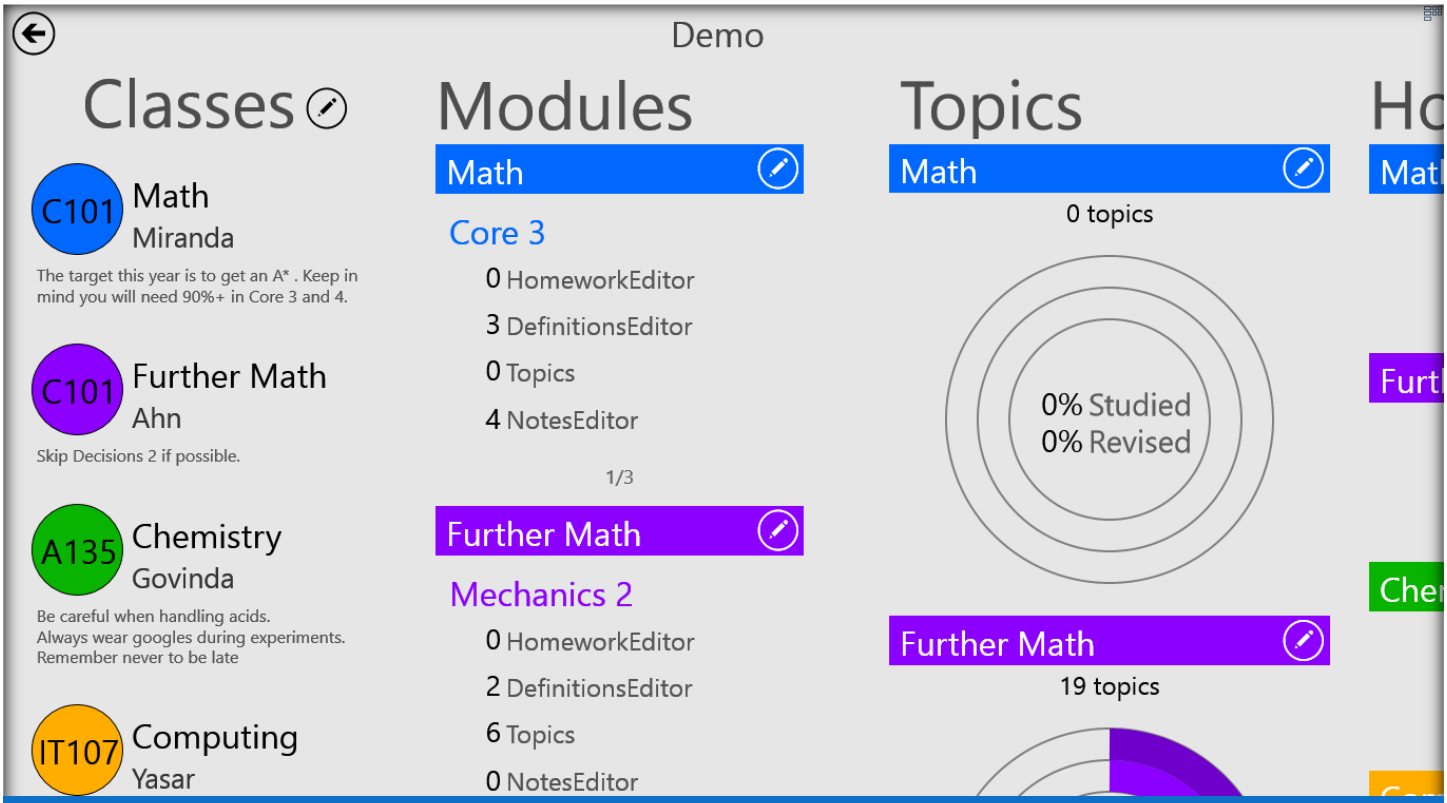


Classes for Windows 8

Leonardo Ciocan

COMP4 Project

TABLE OF CONTENTS

ANALYSIS

BACKGROUND IDENTIFICATION

My college is host to a large number of students between the ages of 16 and 20. There are currently around 1200 hundred students which study a wide range of subjects ranging from vocational courses to A Levels. The abundance of courses implies very different schedule structures, as well as quite an ordeal in the way the students manage their course related material. There are only 30 teachers so the process of distributing material and ensuring the students are in line with their targets is a complicated and time consuming matter. The application I will write will help alleviate those problems, it will be available for students to download on their own PCs/tablets and the Computing Department will preinstall the app on each computer in IT rooms and library computers.

DESCRIPTION OF CURRENT SYSTEM

Students are assigned a specific schedule for their year which outlines a recurring pattern of lessons that is repeated every week. In our college, the data is stored on a private network and inaccessible to the student. In order to allow the user to know their schedule, they are given a printed copy of their timetable. The paper is a standard non-laminated A4 sheet which is issued at the start of the year and can be replaced if needed by going to the student services room.

This system is inefficient in a number of ways which are directly related to the disadvantages of using a fragile physical medium such as paper. Students are often loaded with huge amounts of papers which make the timetable very likely to get mixed in, thus the timetable must be kept separate – for instance I personally keep it in my pocket at all times for easy access – this is however very damaging to the thin paper which does not handle folding, creasing and friction (i.e. from jeans) well. The paper is very likely to wear out from being handled, it is also very volatile to liquids and humidity. The paper – when folded – is small and its retrieval is harder than it should be – the user/student has to search for the paper which is non-trivial since the location of the paper is not always the same and might have changed (i.e might's put it somewhere else when in a hurry).

The current system also suffers from another big disadvantage – the data on the paper is fixed in ink – it is not editable. Therefore the whole paper must be discarded in the event of a change in the schedule, which is annoying , inefficient and a waste of natural resources (note that this happens on a large scale so the paper waste is significant). Also the student cannot modify the schedule beyond the school-approved schedule. For instance the user cannot add an extracurricular lesson to the system – he must either scribble it or remember it.

Last but not least, the paper medium conveys very little information, it tells the user the lessons in the schedule but little information is conveyed by this. The user might want to attach data to the schedule – for instance a homework system is something that would benefit from sharing the same space as the homework. Currently the student has to note the homework down on something like his textbook and there is no centralized place for all the homework to be shown. There is also no indication to the student of which homework are due soon and which homework correspond to which subject (for instance "Page 301 question 1a and 3c" does not convey the information needed to derive the related subject/course). Things like writing definitions and notes are currently based on paper but could be easily deprecated by a suitable software alternative. Lastly, it is hard for students to remember the structure of their course and their progress , this is why the app could simplify the process.

Using the current system is also not enjoyable - most people learn their timetable and discard the paper because it offers no functionality outside of showing their lessons. Therefore a improved timetable system must aggregate the user's school related information in one view.

IDENTIFICATION OF PROSPECTIVE USERS

The end user are students at my college .The terminology used in the program will assume that this is the end user (for instance words like "classes" and "teacher" will be user) but the end user could use the app for something different – like a gym schedule – which suits the system even if the event scheduled is not what I intended. Teachers can also use the program by just not writing the optional "Teacher" field - as they are the teacher.

I may also release the application publicly such that students from other colleges/universities may use them for their own courses.

The application covers a wide spectrum of tasks that the user might undertake. It could be used to check the timetable in the morning to plan their day. The dynamic info screen would allow them to easily get an overview of due homework. Within a class, the user could add definitions to the app. They could also take notes of the lecture and format them. The homework manager will help them remember which homework is due when.

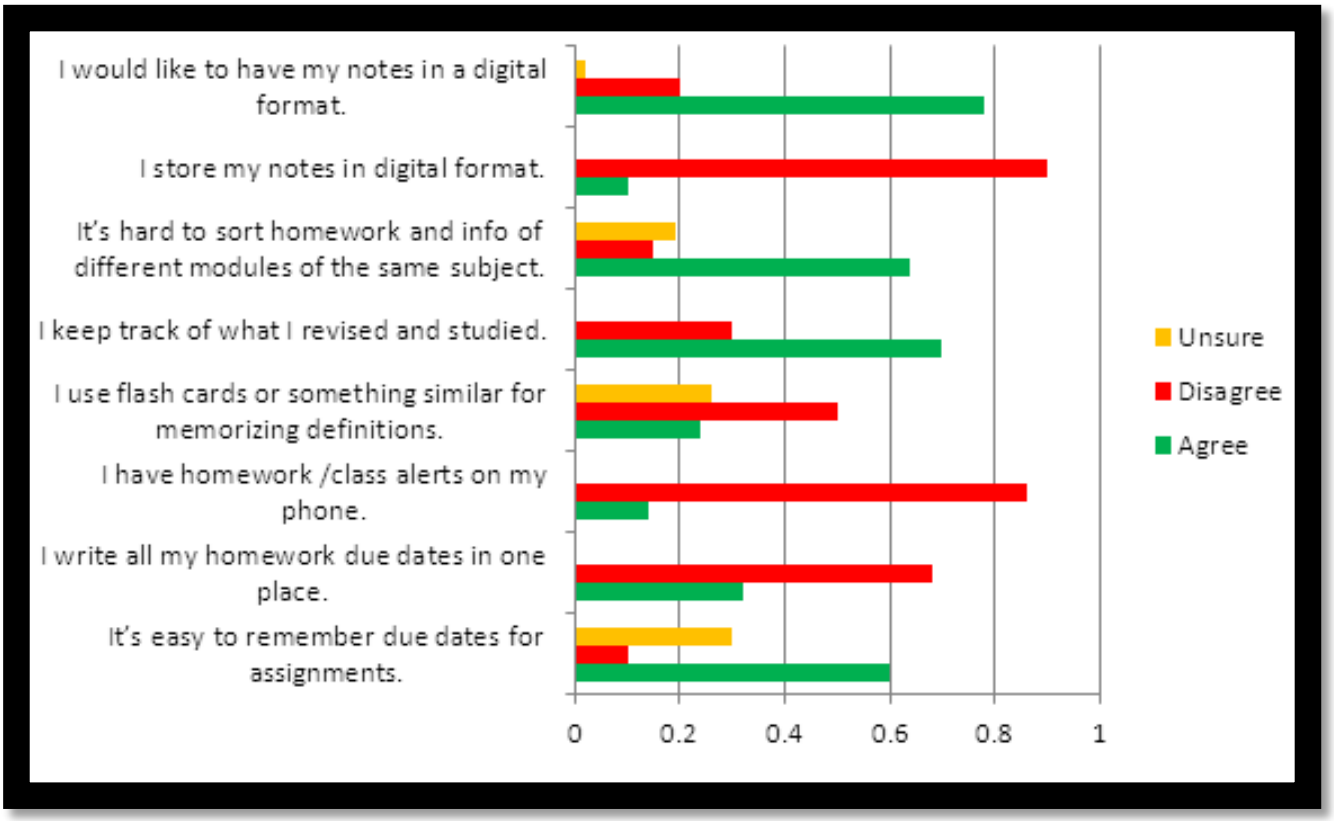
While the app is meant to be single user, it is easy to export a timetable.

As for my actual end-users testers, I will have two of them. One of them will be part of the IT department who will check and assist me with making sure it is technically compliant to their needs (deployment to college computers, branding) and a student who will comment on the actual application functionality.

Note that my client , the IT department manager is not the end-user , but he facilitates the delivery of the app to all my users.

IDENTIFICATION OF USER NEEDS AND ACCEPTABLE LIMITATIONS

The end product should provide the user with a easy to use and flexible timetabling software which also works as aggregator for data related to their subject such as homework and notes . I have talked with a lot of students to get an overall idea of what the user might want and how it should function. I have also taken a questionnaire to decide on whether the issues my application is trying to solve exist at large. The questionnaire was filled by 52 students spread across a variety of subjects and ages. AS , A2 and BTEC students were all represented in this survey.



The questionnaire above shows that a lot of people have very bad habits regarding data management which is a result of the mediocrity of the current system. The application must provide an experience that encourages the user to manage his tasks more responsibly by allowing him to easily input the data to the digital format.

The results are normalized such that 1 represents 100%.

It appears that a lot of students would like to have their notes in a digital format (Q1) but are not able to do so now (Q2) for a variety of reasons we will explore in the interview later on. Sorting data by modules also appears to be quite a challenge to the students as shown by Q3 , which is understandable given each course has 2-4 modules per year. Keeping track of what they revised seems to be quite common at 70% but I still believe this is a necessary feature as its very important to keep track of this. Q5 implies not many people use/are aware of techniques to memorize definitions - which is a disadvantage to their educational performance. Homework management is lacking as shown by Q6 and Q7 - this is because it's a task that is not suited for an analog medium like paper.

INTERVIEW WITH END USER

I had a short interview with my student end user who will actually use the app. He does Math, Computing, Physics and Chemistry.

Q: How do you handle homework? Where do you make a note of it and how do you remember it?

I just make a small note at the end of last lesson's notes. I did find myself with overdue homework when I simply forgot about it, I don't have any specific way of remembering - it usually just becomes a habit to check my book, however many of my classmates do not do this.

Q: Are you interested in typing your college notes?

It depends on the course, I certainly do want something like that for physics, but I will stick to paper for math because I need to draw shapes and equations.

Q: What things do you think a computer application would enhance?

Well, I guess it's safer to have it on a computer - paper could be lost or damaged. You could also make instant copies and give it to your friends - it's hard to do that with paper – nobody likes photocopying.

Q: How do you memorize definitions and keywords? Where do you get information from?

They are usually written on the sides in the text book , but you have to look for them which is kind of long.

Q: Do you generally study a topic in the order given or randomly?

It depends on the subject – some topics in Math require information in previous topics. This is also the case with the others but not always. I do revise in order just to keep it easy but I do have some problems remembering what I revised last.

INTERVIEW WITH CLIENT

I have also asked the IT administrator who will handle deployment of the application

Q: What do you hope this application will achieve?

It will hopefully improve the performance of our students since we found many of them have organization issues.

Q: How would the college want to customize the app content?

There should be some sort of branding in the app, to help the user know where to seek help and advice.

Q: What sort of file format would you like the application to interact with?

Any text format is fine, we could generate timetables with a simple script that takes our database and generates the file that we would give the user to plug into the application.

Q: When will the app be deployed?

We will upgrade the operating system to Windows 8 over the holidays to avoid any issues , we will then deploy the application after that and have it working by the time the students return.

LIMITATIONS

There are a few limitation with my solution. Firstly, sharing will not be as prominent as I would like - in my ideal version everything would be sharable as an image - the most common medium for transmitting formatted data. However I can't do that due to a limitation in the API. The WinRT runtime does not feature WritableBitmap.Render() so I am unable to render UI elements to a

bitmap. This will be probably be fixed in the future, as all languages of the WinRT platform reach feature parity.

This is will also not be cross platform or backwards compatible with previous versions of Windows. Porting the application would be too time consuming and would decrease the overall quality of the app as time would be allocated inappropriately.

DATA SOURCE(S) AND DESTINATION(S)

The user will be entered by the user using their mouse/keyboard or touch (if used on a tablet). The data can will mostly consists of text with a few exception such as color and time which are picked using the custom color picker and date picker respectively. Another data source is the XML file that the user can export from the app - this can be used to restore a timetable.

Every single piece of data is stored in a Timetable instance, which is then serialized to a XML file. The XML file can be exported and edited by the user by hand or using any 3rd party software. When importing an XML timetable file, the user will need to ensure no other timetables with the same name exist (an error pop up will appear otherwise).

After the data is inputted, it is stored in the memory until the user performs a operation that initiates a complete save to the disk (for instance creating a new timetable or just moving between pages)

ANALYSIS DATA DICTIONARY

DETAILED

Timetable

Field	Description	Data Type	Validation
Name	The name of the timetable	string	Up to 20 characters.
Events	A list of events (classes/activities)	List<EventModel>	No limit

EventModel

Field	Description	Data Type	Size
Name	The name of the event.	string	Up to 20 characters.
Class	The class (location where the event takes place)	string	Up to 7 characters (is usually a code i.e A222)
Teacher	Name of teacher	string	Up to 25 characters
Notes	Any additional notes regarding the event	string	Up to 300 characters
Instances	A list of instances that show when an instance of the event occurs.	List<Instance>	No limit

Instance

Field	Description	Data Type	Size
StartTime	The start time of the instance.	Time	N/A
EndTime	The end time of the instance	Time	N/A
Day	Indicates on what day the instance occurs on.	int	1-7
EventName	A reference to the name of the owner event	string	Up to 20 characters.

Time

Field	Description	Data Type	Size
Hours	Hour component of time.	int	23
Minutes	The minutes component of time	int	59

Module

Name	**The name of the module. This is called "Default" when no other module exists.**	**string**	**Up to 20 characters.**
Homeworks	A list of all homework added to this module	List<HomeworkTask>	N/A
Definitions	A list of all definitions added to this module	List<Definition>	N/A
StructureBlocks	A list of all course structures added to this module	List<StructureBlock>	N/A
Notes	A list of all notes added to this module	List<Note>	N/A

HomeworkTask

Field	Description	Data Type	Size
DueOn	When the homework is due	DateTime	N/A
Notes	Description of homework	string	500

Completed	Was it completed already?	bool	N/A

Definition

Field	Description	Data Type	Size
Caption	The title of the definition	string	35
Content	The actual definition	string	250

StructureBlock

Field	Description	Data Type	Size
Name	Name of block	string	20
Topics	List of all topics in block	List<Topic>	N/A

Topic

Field	Description	Data Type	Size
Name	Name of topic	string	20
Studied	Was this topic studied?	bool	N/A
Revised	Was this topic revised?	bool	N/A

Note

Field	Description	Data Type	Size
Name	Name of note	string	35
Description	Description of note	string	200
Fragments	A list of fragments (paragraphs) that make up the note object	List<NoteFragment>	N/A

NoteFragment

Field	Description	Data Type	Size
Content	Information of fragment	string	1000
BackColor	Color of the container that holds the text	Color	N/A
FrontColor	Color of the text	Color	N/A
FontSize	Size of text	double	48
Bold	Is the text bold?	bool	N/A
Italic	Is the text italic?	bool	N/A
Centered	Is the text centered?	bool	N/A

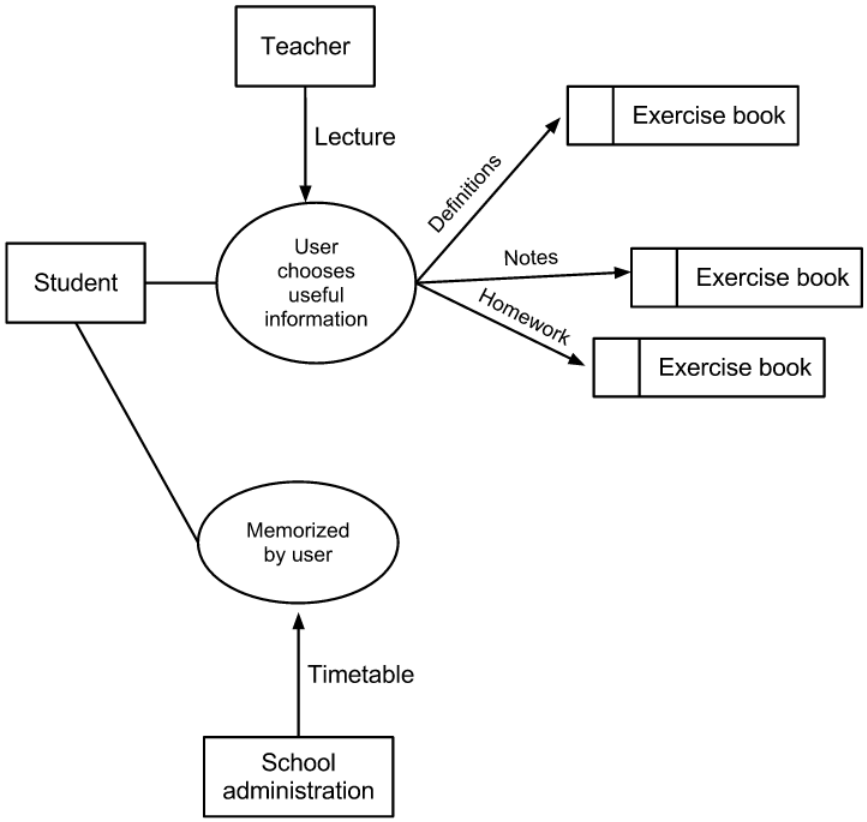
DATA VOLUMES

Every timetable is stored in an XML file. Since it's a plain text file , the data does not take a lot of space. An average timetable would never go above 500KB. The software will process data through out the day as the user might want to take notes. However changes to class/event data would mostly occur at the creation of the timetable. The data is saved asynchronously - and it does not save for every change , changes that are small are saved in the memory data model and saved when a larger event or transition occurs. The number of usages per day could vary based on the user.

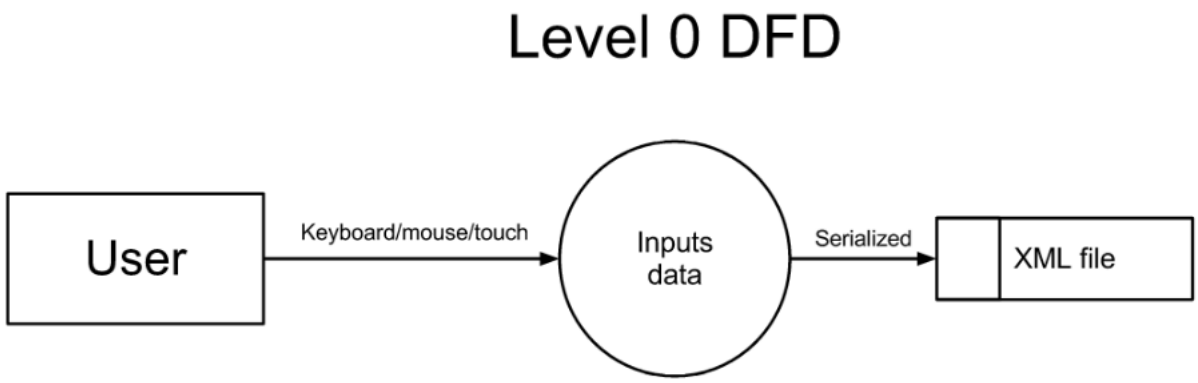
DFD OF EXISTING SYSTEM

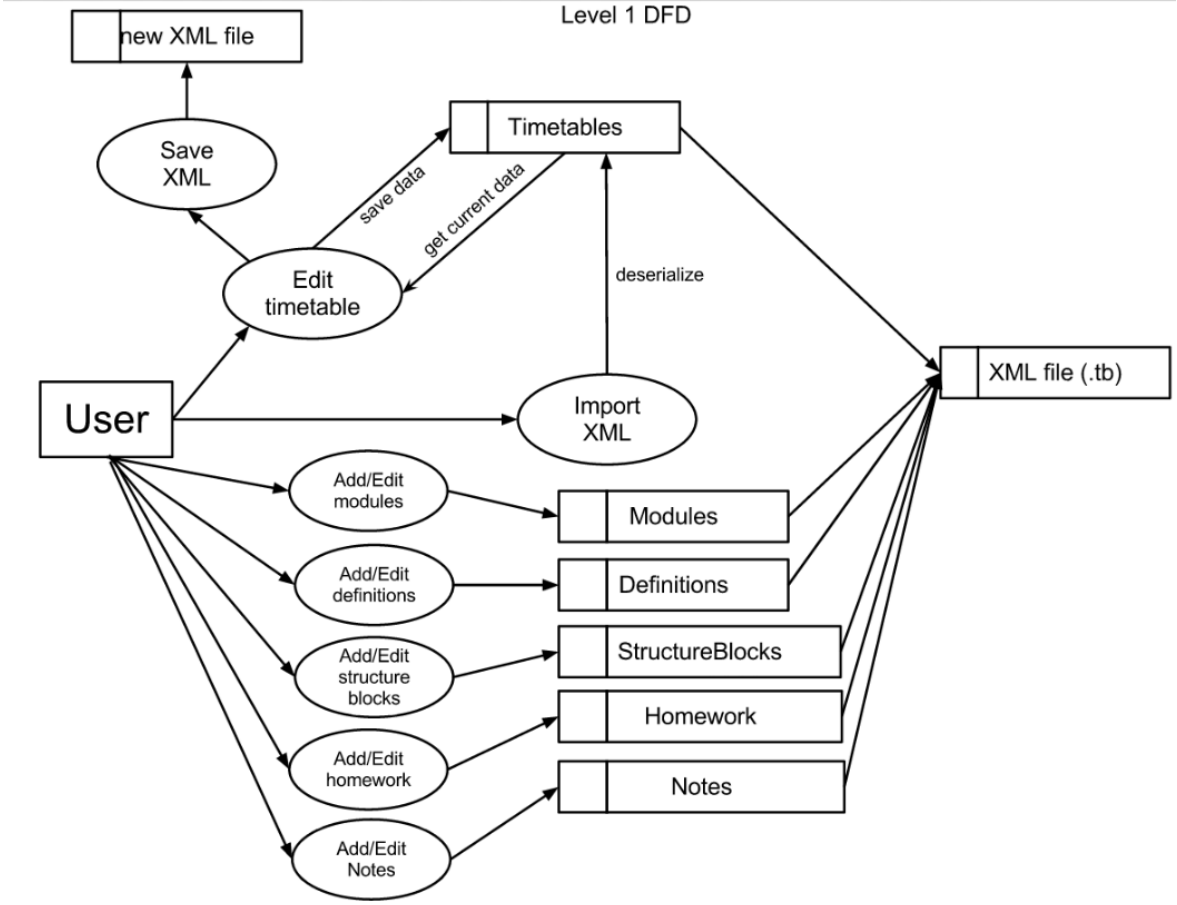
Level 0 DFD

Level 1 DFD



DFD OF PROPOSED SYSTEM





JUSTIFICATION OF CHOSEN SOLUTION

Traditionally a timetable shows a schedule and nothing more. This is a very old custom from when an analog system is the better choice. In my vision of a digital timetable , the system is responsible for much more than showing the schedule , a function which is often unnecessary , and pulls all the student's data into one consistent design. My application revolves around a timetable system but enhances it by allowing the user to build on top of the timetable and create a unified place for their information. This is extremely useful because every medium of information that my application conveys suffers from very similar defects - such as being sparse across books and papers , being fragile and hard to share.

In order for this application to work properly , it must be usable on both desktops/laptops and mobile devices , since the user might have such a device with them while going to lessons. Therefore choosing the right platform is essential – it can't be desktop-only since the user will need to check their schedule/homework on campus. And it can't be just for mobile devices since the user will want to check their schedule/homework status at the comfort of their home computers.Keeping this in mind I have decided to build this app for the Windows 8 metro environment which can be used on both Intel computers and arm tablets. Furthermore , windows 8 apps can be easily ported to Windows Phone 8 while using almost the same code (note that I will not be porting it to the phone platform for this coursework since the platform is yet to be released).

Any data entered is mutable and thus every action done by the user can be undone. The title and description of a timetable can also be changed. Even changing the file name of an exported timetable is handled gracefully , the data including the name is stored inside the file and the filename is ignored during import. to make sure the user didn't accidentally rename the file. All input is done via the user interface and can be operated by both mouse and touch. The interface is resolution independent and fits any given display.

After the information is entered it will be saved asynchronously to the application's isolated storage as an XML file. The user can also export the data as a .tb file (same xml format but unique extension for app association purposes -double clicking a .tb file imports it).

The user can choose the name and description of the timetable. Within the timetable he can add Classes and modify their Name , Teacher , Room and description. Each Class contains a list of Class Times which have a Start Time , End Time and Day property.

Other information which the user might want to associate with their timetable are : notes , modules , homework ,definitions and course structure. My solution will provide all of those in a consistent and friendly UI.

The user should also get a preview of all the information on a general-purpose screen , where he could check his schedule , his upcoming homework deadlines , his notes , refresh his memory by displaying random definitions and at what point on his revision he is.

OBJECTIVES FOR PROPOSED SYSTEM

1. Provide a way to digitally input a current timetable schedule
2. Allow the user to schedule lessons in the weekend and have the timetable adjust accordingly
3. Only show relevant data which is appropriate to the user.
4. Let the user specify modules for each course and aggregate all other information types on them
5. Allow the user to enhance the visual difference between classes by letting the user assign colors
6. Enable exporting of timetables to XML
7. Enable importing of timetables from files
8. Give the user the ability to create homework
9. Present the homework with a useful view that aggregates them including due time warnings to indicate the proximity of the date the homework is due on.
10. Let the user plan and track his revision based on a table of contents which described the structure of the course
11. Aggregate notes based on individual modules
12. Let the user add definitions
13. Help the users memorize them by using quizzes or similar memory games.
14. Enable the user to customize the list of classes with attributes like teachers and notes
15. Enable the user to pick lesson times for each lesson.
16. Visualize the user's schedule for the day (i.e next lesson)
17. Visualize the user's week schedule using a timetable
18. Show the user all his timetables and courses.
19. Help the users memorize them by using quizzes or similar memory games.
20. Allow the user to delete and create new timetables.
21. Let the user customize their notes with formatting
22. Add definition to notes
23. Show the user a visual indication of where he currently is in the day so he may see upcoming lessons easily.
24. Enable each timetable to have bundled information about the academy the timetable is related to, either entered by the user or generated by the academy.
25. Share timetable as picture

APPRAISAL OF FEASABILITY OF POTENTIAL SOLUTIONS + JUSTIFICATION

This system could be implemented in a number of ways , a good solution will be the result of an interplay between practical hardware and great software. I will explain some of the choices I made when deciding on the tools and frameworks for this project.

The user could use some general purpose tools like Office Word/Excel but it would provide a poor and unsuitable experience , it would not be a major improvement over an analog solution , so bespoke software is needed.

It is really important to ensure the application can be run without inconveniencing the user. The application carries very small amount of data - most of it being text. The UI is broken into pages which are not loaded until needed and everything is loaded in the memory asynchronously at the start in order to avoid having to query the hard drive at any point after the initialization of the app thus ensuring the using will not see any load times. Every PC in the school is more than enough to run this application without any problems .The application can also be installed on Windows 8 tablets which have been shown to run applications that are much more intensive with no trouble. Choosing this platform is important because the user can opt between a pc or tablet or both which makes it much more versatile.

In terms of how the application is written there were also a few important choices made. The language used is C# because it is easy to read and mantain due to features such as lambda expressions , extension methods , operator overloading and others.

Other options were Javascript+HTML , Visual Basic and C++. JavaScript was ruled out because the language and compiler are too forgiving and thus make the application hard to debug - managing complex objects and logic is also quite hard in JavaScript. It also is very inefficient in terms of its integration with HTML - which acts as the UI layer, but is not bound to the native UI on each platform thus making it useless for it to be a native app. Web based apps need storage space , bandwidth and maintenance and rely on infrastructure which I do not have and do not want to pay for.

Visual Basic is too verbose for this task , it is also used less in professional environments - I have some experience with it and I think it would increase the amount of code that needs to be written and do little to improve readability (in fact it's over verbose syntax is sometimes noise that differs

from the actual code). C++ is not needed since the application does not use DirectX and managing memory without a garbage collector is too complicated and overs little gains in performance since the application is not performance intensive.

C# is also coupled with XAML, an xml-like language for writing UI - they are extremely integrated and Visual Studio makes development a very streamlined process. The IDE is also very widely used and so finding other people to work with it would be easy.

I also had to decide on a way to store data. After some research/prototyping I decided to serialize the whole thing to xml. Because:

-A database introduces uneeded complexity to an overall simple data layout

-XML files are tangible to the user and he can email them , or even edit them by hand

-The administration of the college can write a simple tool to parse the XML and/or issue timetables to students.

-It's easier to maintain a file-based system

-Debugging files is more efficient than outputting database logs

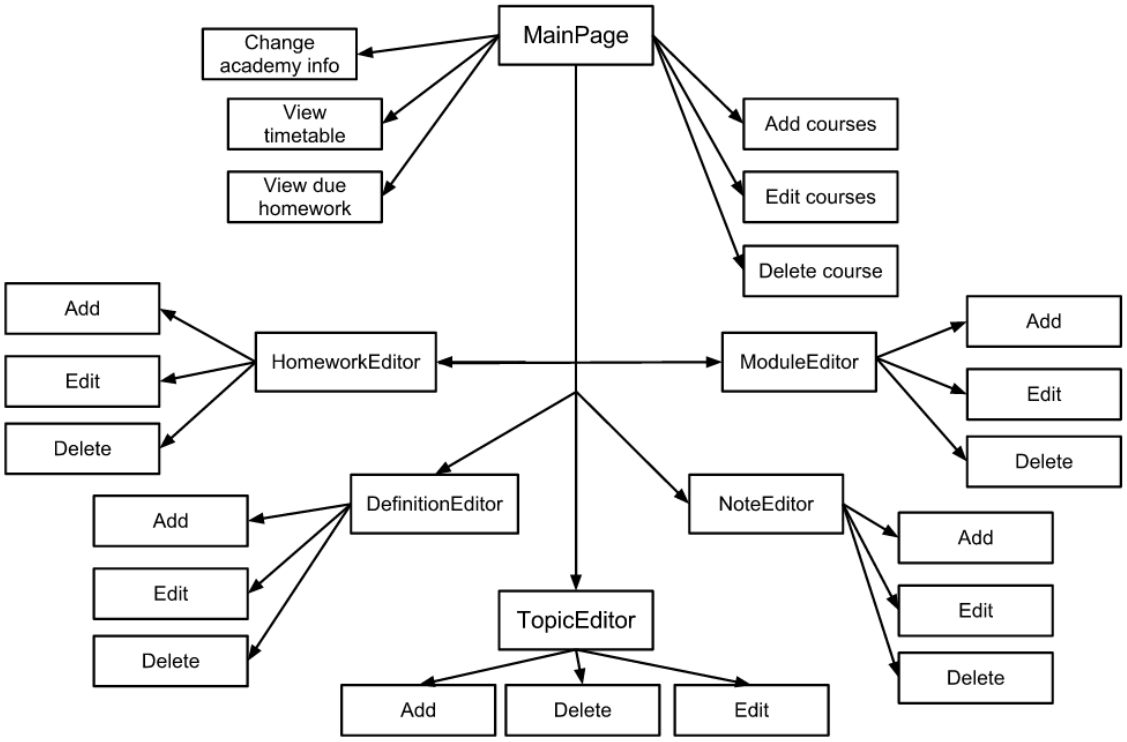
DESIGN

OVERALL DESIGN SYSTEM

Input	Processes
Class details • **Name** • **Teacher** • **Color** • **Room** **Class times** • **start time** • **end time** • **day** • **Homework** • **due date** • **description** **Notes** • **title** • **paragraphs** **Definition:** • **label** • **content** **Topics:** • **subtopics** • **title** **Academy Information:**	**Calculate earliest / latest lesson** **Compute and sort due homework times** **Serialize timetable to XML** **Calculate percentage of topics studied/revised** **Add courses** **Edit courses** **Remove course** **Add homework** **Edit homework** **Remove homework** **Add notes** **Edit notes** **Remove notes** **Add definitions** **Edit definitions** **Remove definitions** **Add topics** **Edit topics** **Remove topics**

• **Name** • **Location** • **Website** • **Branding color**	
Output **Exported backup file** **Class info** **Definitions** **Topic structure** **Academy information**	**Storage** Class Details Class Times Homework details Notes details Definitions Course details

MODULAR STRUCTURE



DATA DICTIONARY

TIMETABLE

Field	Description	Data Type	Size
Name	The name of the timetable	string	Up to 20 characters.
Events	A list of events (classes/activities)	List<EventModel>	No limit

COURSE

Field	Description	Data Type	Size
Name	The name of the event.	string	Up to 20 characters.
Class	The class (location where the event takes place)	string	Up to 7 characters (is usually a code i.e A222)
Teacher	Name of teacher	string	Up to 25 characters
Notes	Any additional notes regarding the event	string	Up to 300 characters
Instances	A list of instances that show when an instance of the event occurs.	List<Instance>	No limit

INSTANCE

Field	Description	Data Type	Size
StartTime	The start time of the instance.	Time	N/A
EndTime	The end time of the instance	Time	N/A
Day	Indicates on what day the instance occurs on.	int	1-7

EventName	A reference to the name of the owner event	string	Up to 20 characters.

TIME

Field	Description	Data Type	Size
Hours	Hour component of time.	int	Up to 23
Minutes	The minutes component of time	int	Up to 59

MODULE

Name	**The name of the module. This is called "Default" when no other module exists.**	**string**	**Up to 20 characters.**
Homeworks	A list of all homework added to this module	List<HomeworkTask>	N/A
Definitions	A list of all definitions added to this module	List<Definition>	N/A
StructureBlocks	A list of all course structures added to this module	List<StructureBlock>	N/A
Notes	A list of all notes added to this module	List<Note>	N/A

HOMEWORKTASK

Field	Description	Data Type	Size
DueOn	When the homework is due	DateTime	N/A

Notes	Description of homework	string	500
Completed	Was it completed already?	bool	N/A

DEFINITION

Field	Description	Data Type	Size
Caption	The title of the definition	string	35
Content	The actual definition	string	250

STRUCTUREBLOCK

Field	Description	Data Type	Size
Name	Name of block	string	20
Topics	List of all topics in block	List<Topic>	N/A

TOPIC

Field	Description	Data Type	Size
Name	Name of topic	string	20
Studied	Was this topic studied?	bool	N/A
Revised	Was this topic revised?	bool	N/A

NOTE

Field	Description	Data Type	Size
Name	Name of note	string	35
Fragments	A list of fragments (paragraphs) that make up the note object	List<NoteFragment>	N/A

NOTEFRAGMENT

Field	Description	Data Type	Size
Content	Information of fragment	string	1000
BackColor	Color of the container that holds the text	Color	N/A
FrontColor	Color of the text	Color	N/A
FontSize	Size of text	double	48
Bold	Is the text bold?	bool	N/A
Italic	Is the text italic?	bool	N/A
Centered	Is the text centered?	bool	N/A

ACADEMYBUNDLE

Field	Description	Data Type	Size
Name	Name of the academy	string	15
Location	The adress of the academy	string	300
Website	Their website	string	75
Accent	Branding color , used to subtly brand the app.	Color	N/A
Events	Holidays and events.	string	2000

FILE ORGANIZATION

Windows 8 stores all the files relating to the app within an isolated storage container, All the XML files are created within the DataFolder (that is in the isolated storage) and they are retrieved by getting all the files that match the pattern "*.tb" , where tb is the standard extension of timetable XML files. This is done to abstract the XML from the user , enable easy debugging by hand and associate the file type such that it opens the application.

The files are written and read using the .NET Serializer class , which converts the appropriate timetable class using the a FileStream pointing at a file with the same name.

When de-serialized , the timetable object is kept in the memory and only written when a flag indicating a change is raised. When importing a timetable the xml file is de-serialized in the memory before saving it , the name is read and that is used as the file name for the file created in the isolated storage (this is to prevent the user from accidentally change the name.

IDENTIFICATION OF STORAGE MEDIA.

By default all files are saved in the isolated storage , a folder within the user's device (PC / tablet/laptop) that cannot be modified by any other software. The user may choose to export the data as a .tb file and put in a memory stick or upload it to the web. Students will probably keep the files in the isolated storage (default location) , since the data will automatically sync with any other windows 8 pc (that has the app).

IDENTIFICATION OF SUITABLE ALGORITHMS

One of the aspects of my application is to enable the user to view which homework are due soon. In order to do this I will be using bubble sort to sort them in ascending order.

Note that it sort HomeworkPreview objects by their DueOn time.

Pseudo code:

```
arr = []
while not ordered:
    x = 0 to lenght(arr) - 1:
        if arr[x] > arr[x+1]:
            temp = arr[x]
            arr[x] = arr[x+1]
            arr[x+1] = temp
```

Another algorithm used is Linear Search . It is used in the methods to get the earliest and latest timetable. Here is how the earliest time works:

time = max_time

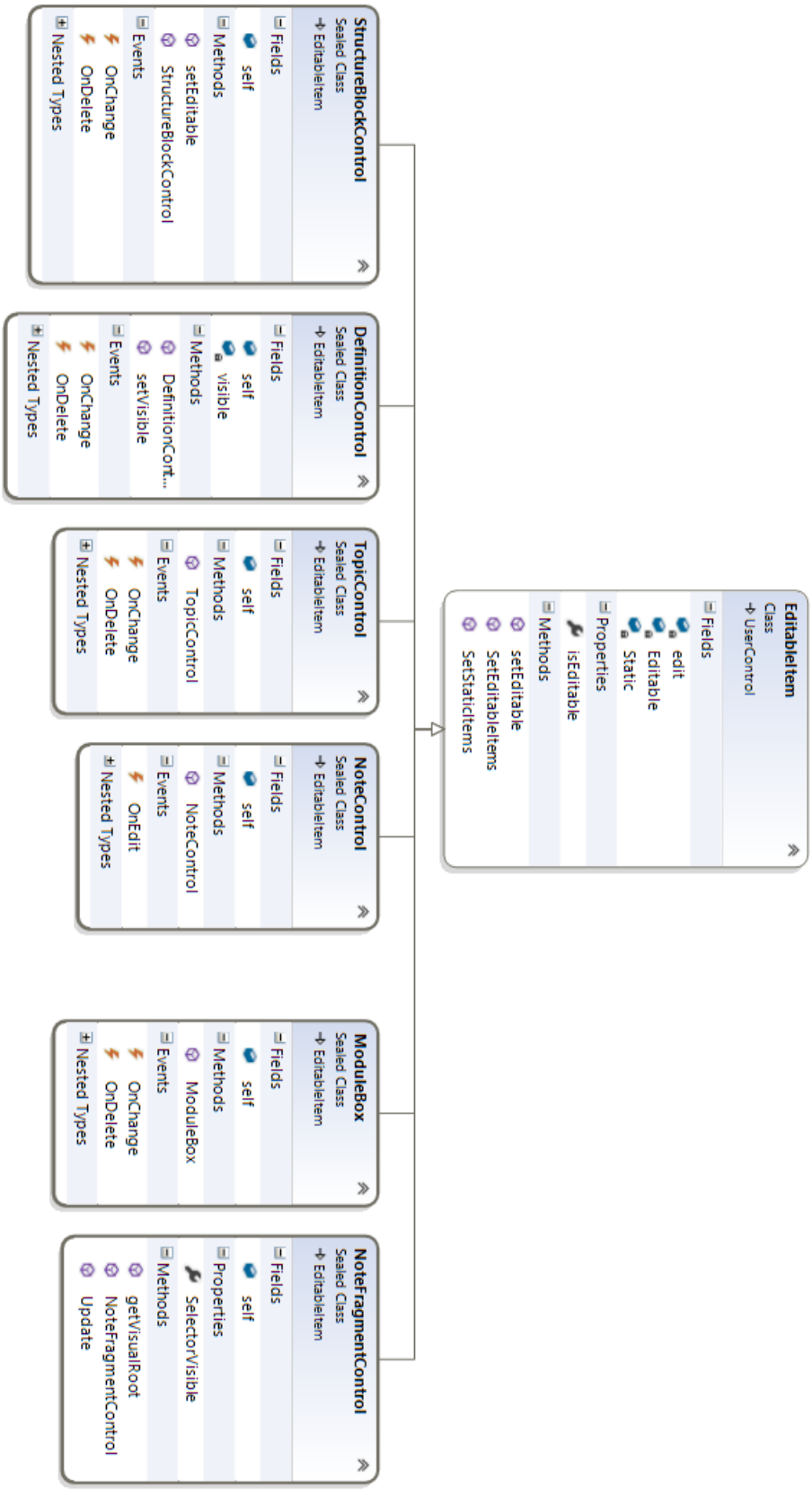
```
foreach instance t in instances:
        if t.time < time:
                time = t.time
```

CLASS DIAGRAMS AND DETAILS OF OBJECT BEHAVIOR

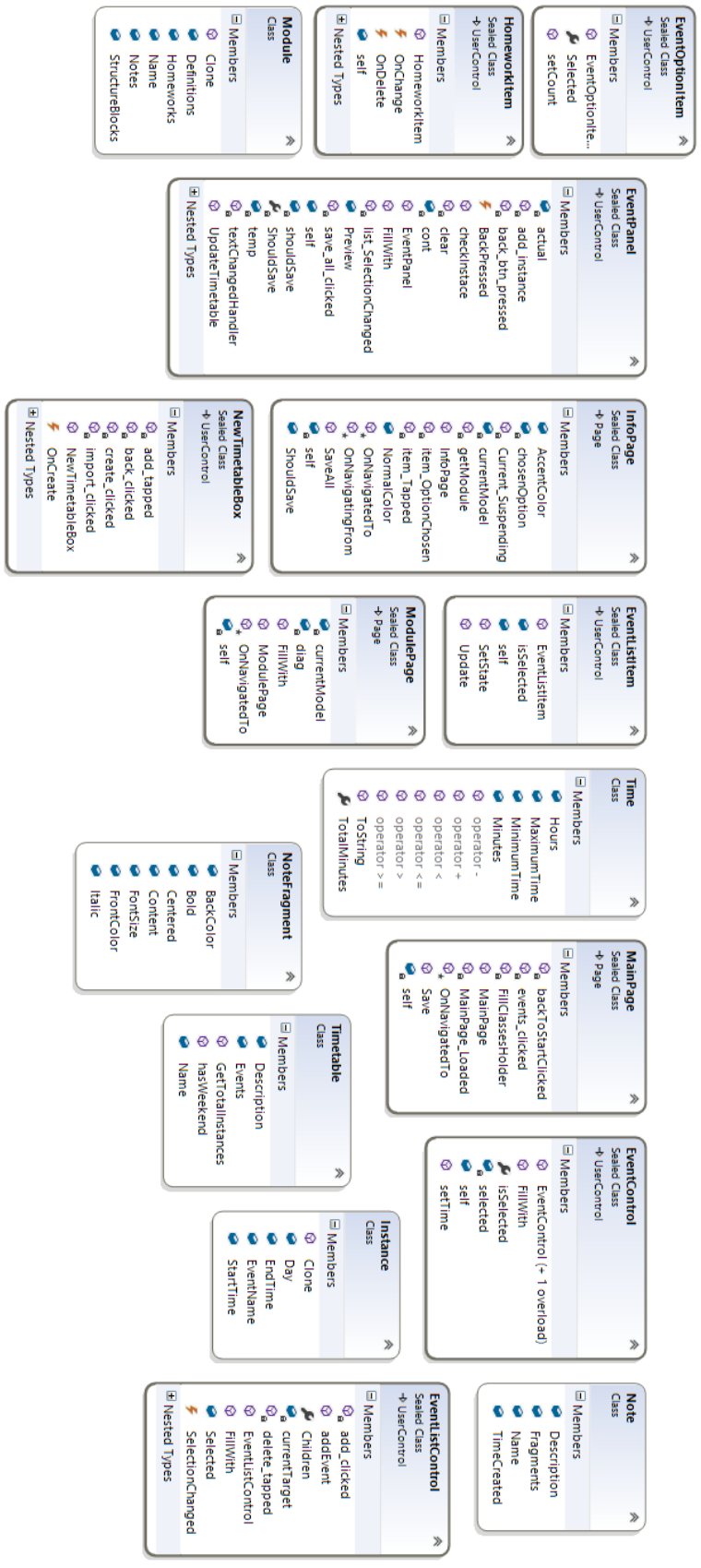
Most controls in my project derive from the .NET type UserControl. The only exception is editable items which derive from a class called EditableItem. It has a collection of static objects and one of editable objects. The derived class sets which items to show during edit mode. It will toggle the visibility of the items within the two collections automatically. Editable item derives from UserControl so the derived controls can be used like normal UI components.

A common pattern used in my classes is to store their model as "self". For instance , a definition control will have a self field that contains a Definition instance. Likewise each page contains a self field representing the current timetable being edited.

Another pattern is to pass one single parameter to controls which convey a single object. If the item is already present then a FillWith method is used to update it with new data. Most controls that represent one object have two events OnDelete and OnChange , which are used to delete and trigger xml save respectively. The OnDelete delegate (OnDeleteHandler) returns the object as a parameter. The OnChange delegate (OnChangeHandler) returns objects that are appropriate to the purpose of the control (usually the model).


EditableItem
Class
UserControl
Fields
edit
Editable
Static
Properties
isEditable
Methods
setEditable
SetEditableItems
SetStaticItems
StructureBlockControl
Sealed Class
EditableItem
Fields
self
Methods
setEditable
StructureBlockControl
Events
OnChange
OnDelete
Nested Types
DefinitionControl
Sealed Class
EditableItem
Fields
self
visible
Methods
DefinitionCont...
setVisible
Events
OnChange
OnDelete
Nested Types
TopicControl
Sealed Class
EditableItem
Fields
self
Methods
TopicControl
Events
OnChange
OnDelete
Nested Types
NoteControl
Sealed Class
EditableItem
Fields
self
Methods
NoteControl
Events
OnEdit
Nested Types
ModuleBox
Sealed Class
EditableItem
Fields
self
Methods
ModuleBox
Events
OnChange
OnDelete
Nested Types
NoteFragmentControl
Sealed Class
EditableItem
Fields
self
Properties
SelectorVisible
Methods
getVisualRoot
NoteFragmentControl
Update


AnimationHelper
Class
Members
AddPointerAnimation
animateMarginIntro
FadeIn
FadeOut
ScaleTo
SmoothMove
SwitchPosition
Nested Types
Definition
Class
Members
Caption
Clone
Content
App
Sealed Class
Application
Members
App
currentTimetable
OnFileActivated
OnLaunched
OnSuspending
CoreTime
Static Class
Members
FormatTime
GetEarliest
GetLatest
IsOverlapping
BoxInfo
Sealed Class
UserControl
Members
BoxInfo
EventPreviewBox
Sealed Class
UserControl
Members
EventPreviewBox
FillWith
EventPreview
Sealed Class
UserControl
Members
EventPreview
BubbleSort
Static Class
Members
IsListOrdered
result
SortPreviews
EditButton
Sealed Class
UserControl
Members
EditButton
StructureBlock
Class
Members
Name
Topics
CentralPage
Sealed Class
Page
Members
CentralPage
OnNavigatedTo
self
EventAppBarItem
Sealed Class
UserControl
Members
EventAppBarItem
self
ClassesEditor
Sealed Class
UserControl
Members
ClassesEditor
FillWith
setEdit
ColorChooser
Sealed Class
UserControl
Members
ColorChanged
ColorChooser
SelectedColor
SelectedRectangle
Nested Types
ExtendedTextBox
Sealed Class
UserControl
Members
ExtendedTextBox
Text
TopicsPage
Sealed Class
Page
Members
OnNavigatedTo
TopicsPage
Topic
Class
Members
Name
Revised
Studied
Extensions
Static Class
Members
GetModule
Constants
Static Class
Members
Blue
Dark
Days
EventControlHeight
EventControlWidth
Months
OptionLabels
Violet
Nested Types
HomeworkView
Sealed Class
UserControl
Members
FillWith
HomeworkView
TimetableView
Sealed Class
UserControl
Members
FillWith
instances
TimetableView
TaskPage
Sealed Class
Page
Members
addHomework_clicked
backButton_Click
currentModel
fillDefinitionsWithModule
fillHomeworksWithModule
fillModuleHeaders
fillStructuresWithModule
OnNavigatedTo
OnNavigatingFrom
saveTimer
saveTimer_Tick
self
ShouldSave
TaskPage
NoteObject
Class
Members
Text

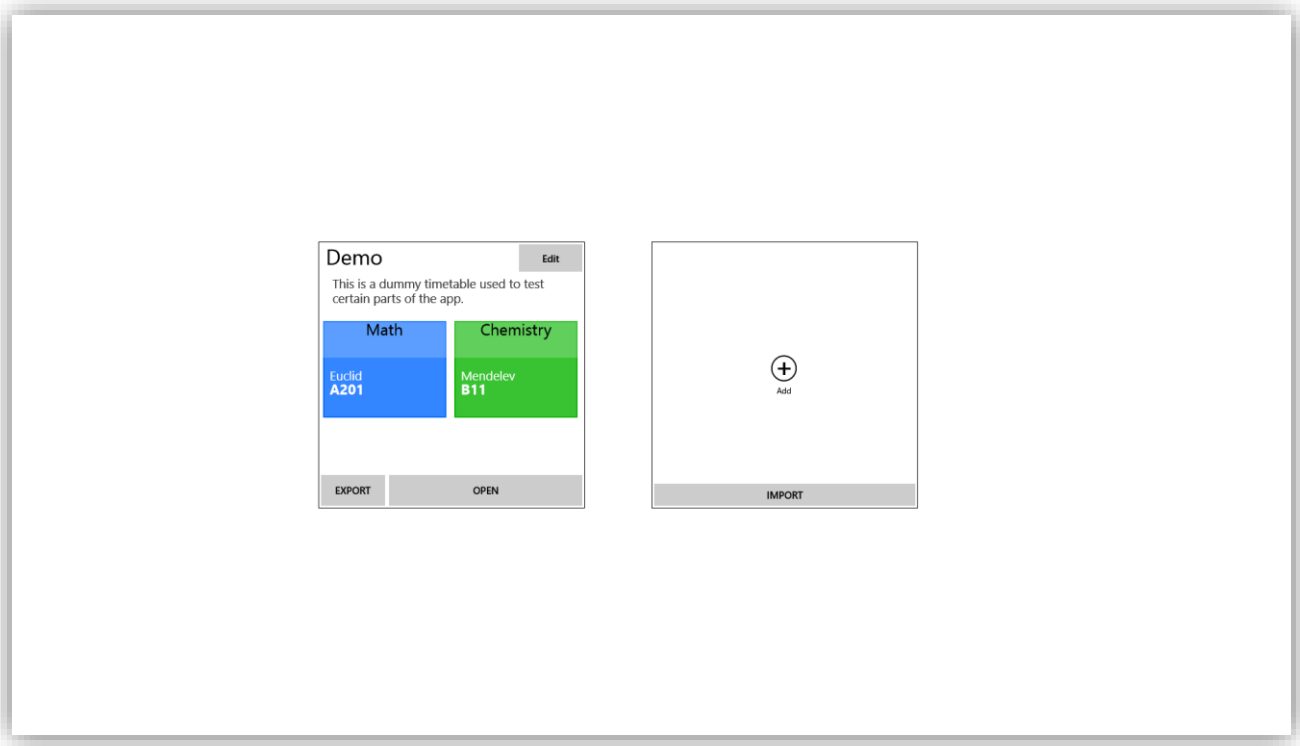

EventOptionItem
Sealed Class
UserControl
Members
EventOptionIte...
Selected
setCount
HomeworkItem
Sealed Class
UserControl
Members
HomeworkItem
OnChange
OnDelete
self
Nested Types
Module
Class
Members
Clone
Definitions
Homeworks
Name
Notes
StructureBlocks
EventPanel
Sealed Class
UserControl
Members
actual
add_instance
back_btn_pressed
BackPressed
checkInstace
clear
cont
EventPanel
FillWith
list_SelectionChanged
Preview
save_all_clicked
self
shouldSave
ShouldSave
temp
textChangedHandler
UpdateTimetable
Nested Types
InfoPage
Sealed Class
Page
Members
AccentColor
chosenOption
Current_Suspending
currentModel
getModule
InfoPage
item_OptionChosen
item_Tapped
NormalColor
OnNavigatedTo
OnNavigatingFrom
SaveAll
self
ShouldSave
NewTimetableBox
Sealed Class
UserControl
Members
add_tapped
back_clicked
create_clicked
import_clicked
NewTimetableBox
OnCreate
Nested Types
EventListItem
Sealed Class
UserControl
Members
EventListItem
isSelected
self
SetState
Update
ModulePage
Sealed Class
Page
Members
currentModel
diag
FillWith
ModulePage
OnNavigatedTo
self
Time
Class
Members
Hours
MaximumTime
MinimumTime
Minutes
operator -
operator +
operator <
operator <=
operator >
operator >=
ToString
TotalMinutes
MainPage
Sealed Class
Page
Members
backToStartClicked
events_clicked
FillClassesHolder
MainPage
MainPage_Loaded
OnNavigatedTo
Save
self
EventControl
Sealed Class
UserControl
Members
EventControl (+ 1 overload)
FillWith
isSelected
selected
self
setTime
NoteFragment
Class
Members
BackColor
Bold
Centered
Content
FontSize
FrontColor
Italic
Timetable
Class
Members
Description
Events
GetTotalInstances
hasWeekend
Name
Instance
Class
Members
Clone
Day
EndTime
EventName
StartTime
Note
Class
Members
Description
Fragments
Name
TimeCreated
EventListControl
Sealed Class
UserControl
Members
add_clicked
addEvent
Children
currentTarget
delete_tapped
EventListControl
FillWith
Selected
SelectionChanged
Nested Types

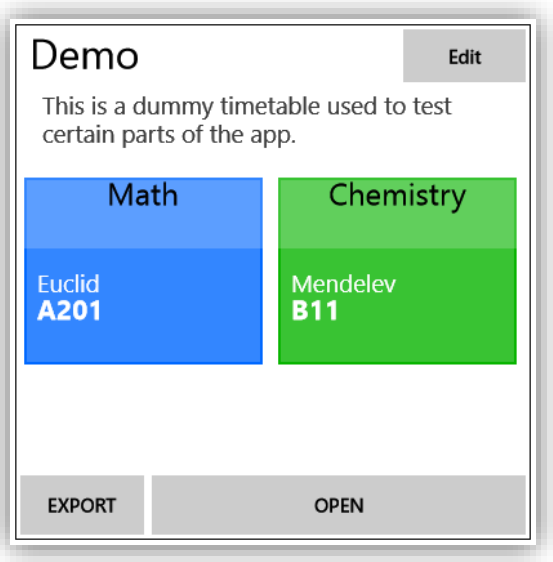
SimpleEventItem
Sealed Class
+ UserControl
Members
Selected
selectedColor
self
SimpleEventItem
SimpleDatePicker
Sealed Class
+ UserControl
Members
BackPressed
Day
fillBoxes
Month
PickedTime
SimpleDatePicker
Nested Types
EventModel
Class
Members
Class
Clone
Instances
Modules
Name
Notes
Teacher
TileColor
HomeworkPreview
Sealed Class
+ UserControl
Members
DueOn
HomeworkPreview
self
ModelViewFactory
Class
Members
CreateNewObject
StartPage
Sealed Class
+ Page
Members
box
Fill
StartPage
StartPage_Loa...
TimetableBox
Sealed Class
+ UserControl
Members
back_clicked
delete_clicked
edit_clicked
export_clicked
OnDeleted
OnOpen
open_clicked
save_clicked
self
task_editor_clicked
TimetableBox
Nested Types
InstanceControl
Sealed Class
+ UserControl
Members
changedEvent
delete_clicked
InstanceChanged
InstanceControl
InstanceControl_Loaded
isValid
OnDelete
self
selfParent
Nested Types
NoteEditor
Sealed Class
+ UserControl
Members
currentFragment
currentFragmentControl
currentModule
fillNotes
FillWith
ItemSelected
NoteEditor
NoteEditor_Loaded
OnBack
self
Nested Types
EventSideBarItem
Sealed Class
+ UserControl
Members
ActuallyTapped
EventSideBarItem
OptionChosen
refreshCounts
selected
Selected
self
Nested Types
CoreData
Static Class
Members
CreateTimetable
DataFolder
DeleteTimetable
EditTimetable
ExportTimetable
GetTimetables
ImportTimetable
Initialize
LocalFolder
SaveTimetable
serializer
Timetables

USER INTERFACE DESIGN RATIONALE

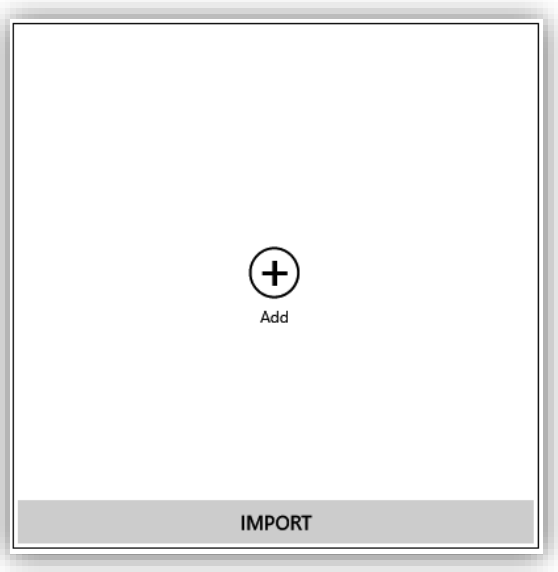
The first thing the user sees is the StartPage which is a place where the user can create , view ,import and export timetables. It shows basic information about each timetable and displays all options using a design that is very pleasing and responsive. Touching the add icon (next image) makes the card flip around to reveal the new timetable dialog.Editing a timetable provides the same experience.

All of the UI follows the metro design language which is one of the pivotal parts of windows 8 apps. The overall idea of the design style is simplicity. That means putting the content first and letting the chrome fade away.




Demo
Edit
This is a dummy timetable used to test certain parts of the app.
Math
Euclid
A201
Chemistry
Mendelev
B11
EXPORT
OPEN


Editing...
Delete
Save
Name
Demo
Description
This is a dummy timetable used to test certain parts of the app.

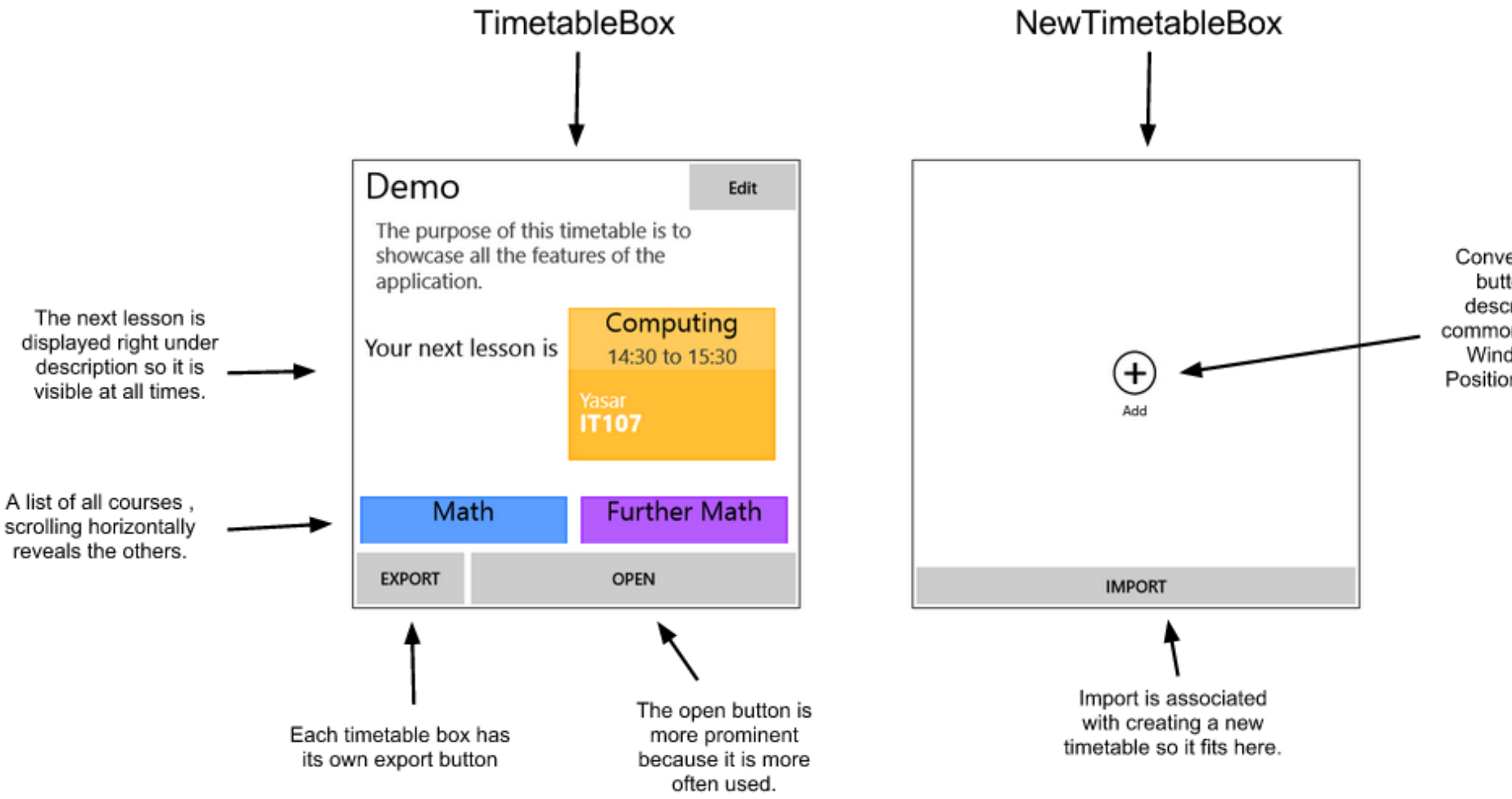

Add
IMPORT


New Timetable
Create
Name
Description

Centre Number:30415

Candidate Number:4017

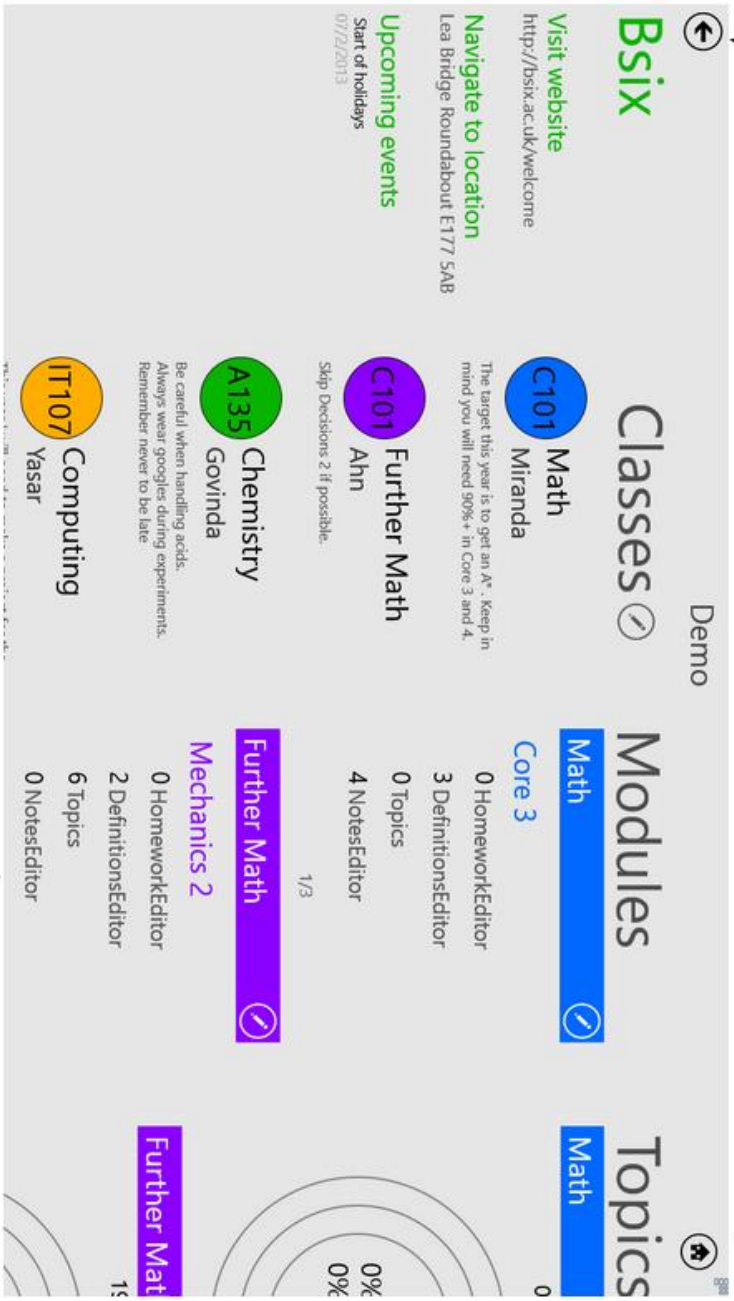
Name:Leonardo Ciocan



The iconography are standard components, this makes it easy for the user to understand what they do. The export button is smaller than the open button because it will be used less often, therefore we should remove attention from it. The open button is used because making the whole box touch enabled would lead to unintended touches. All buttons are 12 pixels away from the border for consistency purposes.

The cards are stacked horizontally and the user may scroll to reveal further timetables. However, that is not likely to happen as the user is likely to have less than 3 timetables.

Note that the vast majority of Windows 8 machines have displays that are widescreen (16:9 or 16:10) therefore it is more efficient to lay out the content horizontally.



Another important page is the MainPage. This is used for general timetable operations such as editing the schedule and classes.

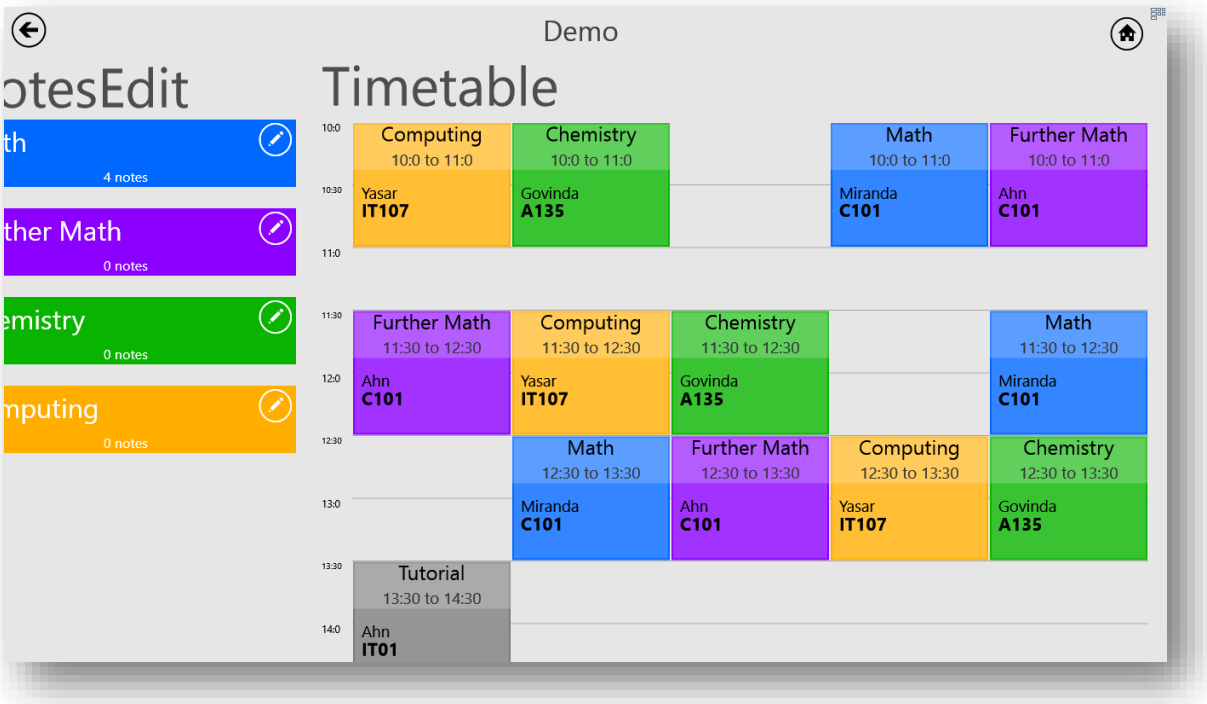
(Image on next page)

Note that the layout stretches across horizontally and the user can scroll to view more content (this is a standard navigation pattern in windows 8).

All the fonts are Segoe UI and anti-aliased . This fits well with the overall operating system and is thus beneficial for the user.

The timetable is on the right , but is easy to get to since it's at the end (so that the user can flick to it or scroll all the way). Each class is rendered in their own color with a 2px transparent black outline to make it easier on the eyes. There are 1px gray lines across each hour label to make it easy to understand at what time each lesson is or how long the breaks are.

The user can color code their lessons and this will be used throughout the UI to differentiate between lessons. The colors provided have been chosen such that a white or black text may be visible when laid out on a control with that color.



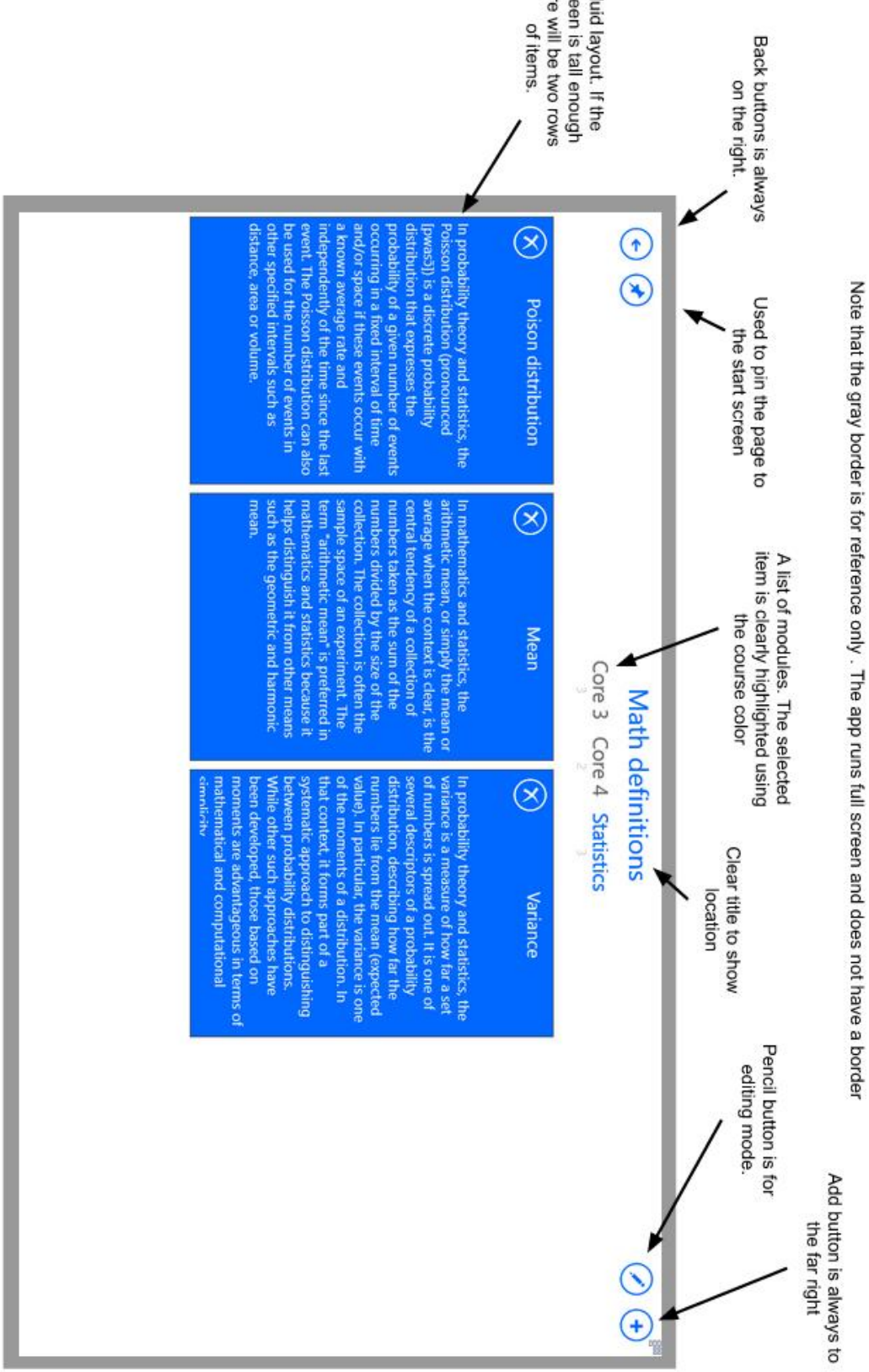
The classes headings has an edit button enclosed in a circular black outline (which is a standard UI pattern). This is easy to understand and the button is efficient using both touch and mouse.

The EventControl (the boxes used in the timetableView) is made of two parts-the header and the body.The header contains information on the class and the body is just a rectangle of the class colour which is extended to represent the timespan of the event.

The Main Page will show snapshots of information to the user. Those snapshots are called previews , there are ModulePreviews , DefinitionPreviews and so on. They are tailored show the content in the most useful way. For instance , the ModulePreview shows how many of each information type there are on each module and the TopicPreview shows how well the revision is going .

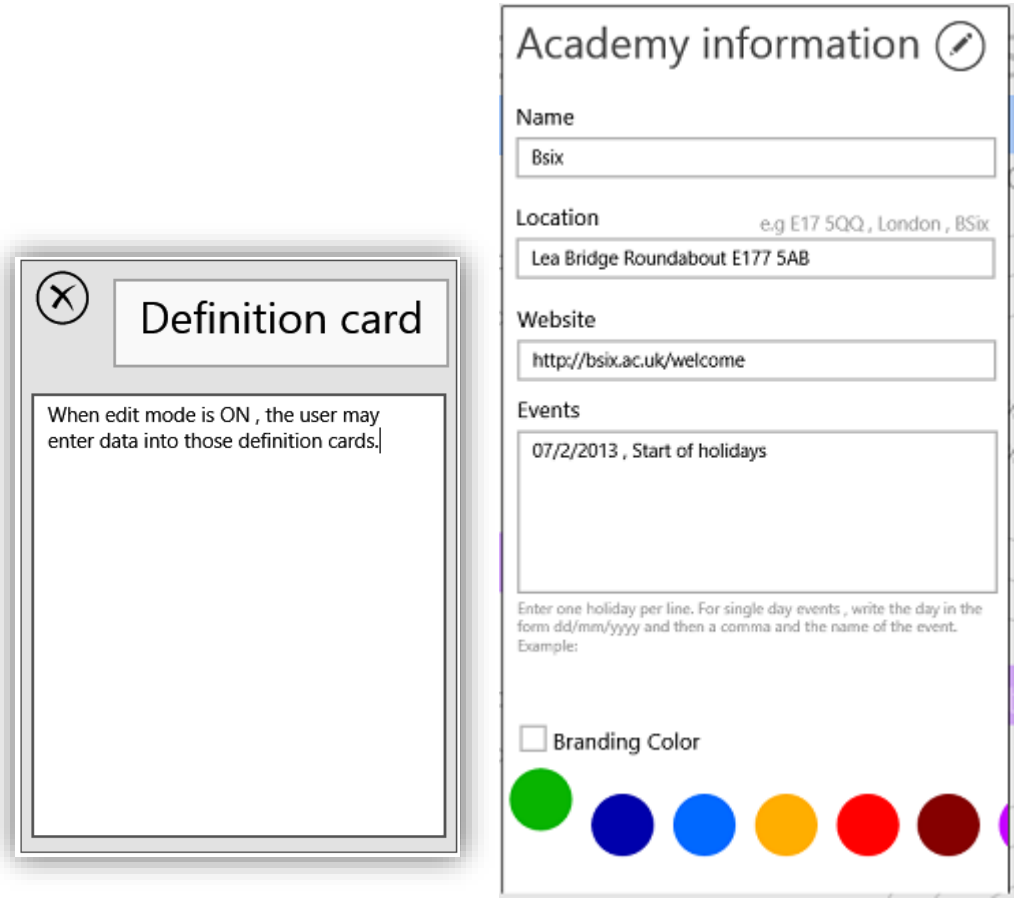
On the left the academy information is shown using the selected color , this can be used by the college to customize the timetable and allow the user to see quick links and important events. This can be edited by pressing the home button on the top right.

To edit anything the user can press the circled pencil next to any item of interest , this is a standard UI pattern that the user will be familiar with (but also is trivial to deduct)

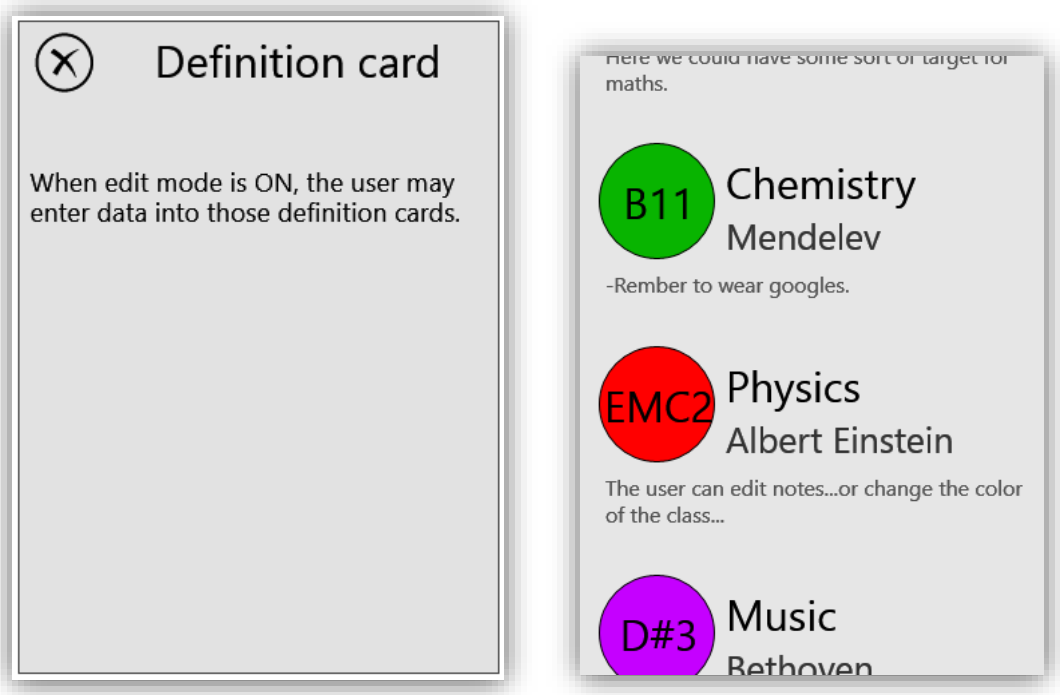

Note that the gray border is for reference only . The app runs full screen and does not have a border
Add button is always to the far right
Pencil button is for editing mode.
Clear title to show location
A list of modules. The selected item is clearly highlighted using the course color
Used to pin the page to the start screen
Back buttons is always on the right.
Fluid layout. If the screen is tall enough there will be two rows of items.
Math definitions
Core 3 Core 4 Statistics
Poison distribution
In probability theory and statistics, the Poisson distribution (pronounced [pwas3]) is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event. The Poisson distribution can also be used for the number of events in other specified intervals such as distance, area or volume.
Mean
In mathematics and statistics, the arithmetic mean, or simply the mean or average when the context is clear, is the central tendency of a collection of numbers taken as the sum of the numbers divided by the size of the collection. The collection is often the sample space of an experiment. The term "arithmetic mean" is preferred in mathematics and statistics because it helps distinguish it from other means such as the geometric and harmonic mean.
Variance
In probability theory and statistics, the variance is a measure of how far a set of numbers is spread out. It is one of several descriptors of a probability distribution, describing how far the numbers lie from the mean (expected value). In particular, the variance is one of the moments of a distribution. In that context, it forms part of a systematic approach to distinguishing between probability distributions. While other such approaches have been developed, those based on moments are advantageous in terms of mathematical and computational

The above screenshot shows an Editor , in this case the DefinitionEditor. Editors are used to edit types of data - they have the same layout. On the top left there is back button , and then a pin button which will make a shortcut to the editor on the start screen. On the right there is an edit button which will toggle the editing mode of all EditableItems in the editor. The add button creates a new one and updates the view. In the middle beneath the title there is a filter that the user can use to filter through their data by module.

UI Sample of input



UI Samples of output



The output of editing the card is the definition card above - the text is saved and non-editable - the actual definition can be collapsed. On the right you can see the input was saved and the color and description changed accordingly.

SECURITY OF SYSTEM

All Windows 8/RT apps built on the WinRT API are sandboxed. They cannot access system files and the user is not able to access any of the files within the app's isolated storage. The timetables that the user creates are stored within a folder called Data in the isolated storage , they cannot be compromised by any means except interacting with the app itself.

SECURITY AND INTEGRITY OF DATA

The user has control over what data is publicly available. Timetables that are not exported reside within the isolated storage. This storage is not accessible via a file browser and can only be used by the application.

Data is only saved while the application is running (not during suspension) , this way data will never become corrupted due to serializing interruptions. Also the de-serializer will detect any abnormality in the timetable file and the app will notify the user that the file is corrupted.

OVERALL TEST STRATEGY

A small amount of testing will be done by me. I will try to input invalid data and check if the behavior of the app is appropriate. I will also check if the app catches exceptions properly (such as files under usage or malformed xml).I will gather some more data on how the app performs by letting a few students use it and provide feedback. I will not influence those test to check if the layout is intuitive enough.

There will be validation checks, unit tests and regular functionality testing.

TECHNICAL SOLUTION

Note that I have left out all generated code and redundant code (namespace , imports).

ANIMATIONHELPER.CS

```
public class AnimationHelper
    {

        #region Opacity Animations
        /// <summary>
        /// Fade out an element to 0 opacity.
        /// </summary>
        /// <param name="target">The element to fade</param>
        /// <param name="duration">The time it should take for the animation to complete</param>
        /// <param name="autoStart">Whether the animation should start imidiatel y after the call or will be handled using the return value.</param>
        /// <param name="autoReverse">Whether the animation should loop back to the original opacity.</param>
        /// <returns></returns>
        public static Storyboard FadeTo(FrameworkElement target,double targetVal ue, double duration, bool autoStart = true, bool autoReverse = false)
        {
            //create a storyboard to handle the animation
            Storyboard sb = new Storyboard();
            //a double animation to animate opacity (of type double)
            DoubleAnimation dt = new DoubleAnimation();
            dt.Duration = new Duration(TimeSpan.FromSeconds(duration));
            dt.From = target.Opacity;
            dt.To = targetValue ;
            //set the property to Opacity and the target to target
            Storyboard.SetTargetProperty(dt, "Opacity");
            Storyboard.SetTarget(sb, target);
            //add the double animation to the storyboard
```

```
            sb.Children.Add(dt);
            sb.AutoReverse = autoReverse;
            if (autoStart) sb.Begin();
            return sb;//return the storyboard so additional operations can be ch
ained (i.e complete events)
        }
        #endregion

        /// <summary>
        /// This will move an element to a specific location in a Canvas with a
smooth animation
        /// </summary>
        /// <param name="target">The element to be moved</param>
        /// <param name="to">The Y coordonate of the wanted position</param>
        /// <param name="duration">Time for the movement to complete</param>
        /// <param name="autoStart">Whether the animation should immidiately sta
rt</param>
        /// <param name="autoReverse">Whether the animation should loop to its s
tarting values</param>
        /// <returns></returns>
        public static Storyboard SmoothMove(FrameworkElement target, double to,
double duration, bool autoStart = true, bool autoReverse = false)
        {
            Storyboard sb = new Storyboard();

                DoubleAnimation dy = new DoubleAnimation();
                dy.Duration = new Duration(TimeSpan.FromSeconds(duration));
                dy.From = Canvas.GetTop(target);
                dy.To = to;
                Storyboard.SetTargetProperty(dy, "(Canvas.Top)");
                sb.Children.Add(dy);

                Storyboard.SetTarget(sb, target);

            sb.AutoReverse = autoReverse;
            if (autoStart) sb.Begin();
            return sb;
        }
```

```
        /// <summary>
        /// This method allows to bind a small reaction animation to an element.
 The element will move when pressed to show it responds to touch.
        /// </summary>
        /// <param name="element">The element to add the animation to</param>
        public static void AddPointerAnimation(FrameworkElement element)
        {
            //animation to shrink the element when mouse is pressed
            Storyboard anim_down = new Storyboard();
            Storyboard.SetTarget(anim_down, element);
            anim_down.Children.Add(new PointerDownThemeAnimation());

            //animation to enlarge to original size when mouse is not pressed
            Storyboard anim_up = new Storyboard();
            Storyboard.SetTarget(anim_up,element);
            anim_up.Children.Add(new PointerUpThemeAnimation());

            bool down = false;
            //when the pointer enters the animation should be shrinking unless t
he mouse is not pressed.
            element.PointerEntered += (e, ev) =>
            {
                if (down)
                    anim_down.Begin();
            };

            //if mouse button is down then begin shrinking animation and set dow
n to true
            element.PointerPressed += (e, ev) =>
            {
                anim_down.Begin();
                down = true;
            };

            //if the pointer leaves the bounds of the element the released event
 will not be detected. Therefore stop the nimation and reverse value of down.
```

```
            element.PointerExited += (e, ev) =>
            {
                anim_up.Begin();
                down = false;
            };

            //if the mouse button is released resize and set down to false.
            element.PointerReleased += (e, ev) =>
            {
                anim_up.Begin();
                down = false;
            };
        }
    }
```

CONSTANTS.CS

```
public static class Constants
    {

        //Used for drawing the timetable view
        public static int EventControlWidth = 187;
        public static int EventControlHeight = 147;

        //used in controls where a list of months has to be displayed (date pick
er)
        public static string[] Months = new string[12] { "January", "February",
"March", "April", "May", "June", "July", "August", "September", "October", "Nove
mber", "December" };
    }
```

CORE.CS

```
//Represents a homework
        public class HomeworkTask
        {
            public DateTime DueOn = DateTime.Now.AddDays(7);//the time when the
homework is due
```

```
            public string Notes = ""; //description of homework
            public bool Completed = false;//whether it's completed


        }

        //represents a definition
        public class Definition
        {
            public string Caption = "";//The term defined
            public string Content = "";//Its definition
        }

        //A module which contains its own set of notes/definitions etc
        public class Module
        {
            public string Name = "";
            //Lists of items
            public List<StructureBlock> StructureBlocks = new List<StructureBloc
k>();
            public List<Definition> Definitions = new List<Definition>();
            public List<HomeworkTask> Homeworks = new List<HomeworkTask>();
            public List<Note> Notes = new List<Note>();

        }

        //A structure block is the name of a topic group but it's abstracted fro
m the user by just calling it topic
        public class StructureBlock
        {
            public string Name = "";
            public List<Topic> Topics = new List<Topic>();
        }

        //Represents a single topic
        public class Topic
        {
```

```
            public bool Studied = false;
            public bool Revised = false;
            public string Name = "Enter a topic name";
        }

        //Represents a single Note
        public class Note
        {
            public string Name = "Name this note";
            public List<NoteFragment> Fragments = new List<NoteFragment>();
        }

        //A paragraph in a note. Has its own class for later extensibility.
        public class NoteFragment
        {
            public string Content = "Turn on writing mode on the left to start e
diting this.";

            public double FontSize=13;
            public bool Italic=false;
            public bool Bold=false;
            public bool Centered=false;

        }
```

COREDATA.CS

```
 public static class CoreData
    {
        //Local folder contains the folder called Data that holds our files
        static StorageFolder LocalFolder = ApplicationData.Current.LocalFolder;
        static StorageFolder DataFolder;

        //List of timetables that is accessible from anywhere in the app
        public static List<Timetable> Timetables = new List<Timetable>();

        //Reusable serializer for writing and reading .tb files
```

```
        public static XmlSerializer Serializer = new XmlSerializer(typeof(Timeta
ble));



        /// <summary>
        /// Used to locate the data folder for further operations - must be call
ed any entry point of the app.
        /// </summary>
        /// <returns>Whether an error occured.</returns>
        public static async Task<bool> Initialize()
        {
            try
            {
                //Try to create the data folder  , or open it if it exists
                DataFolder = await LocalFolder.CreateFolderAsync("Data", Creatio
nCollisionOption.OpenIfExists);
            }
            catch
            {
                return false;
            }
            return true;
        }

        /// <summary>
        /// This will create a timetable file from a Timetable instance
        /// </summary>
        /// <param name="timetable">The timetable to save</param>
        /// <returns></returns>
        public static async Task<bool> CreateTimetable(Timetable timetable)
        {
            var found = false; //whether a timetable with the same name exists
            try
            {
                //try to get timetable file with same name
                await DataFolder.GetFileAsync(timetable.Name + ".tb");
```

```
            }
            catch
            {
                found = true;
            }

            //if it's found , then it cannot be made. Another name must be chose
n
            if (!found)
            {
                var dialog = new MessageDialog("A Timetable with that name alrea
dy exists.\nPlease choooose another name.", "Oops");
                await dialog.ShowAsync();
                return false;
            }
            else
            {
                //create the file and asynchronously serialize the data bundle t
o xml then add the object to the list
                StorageFile file = await DataFolder.CreateFileAsync(timetable.Na
me +".tb");
                Stream filestream = await file.OpenStreamForWriteAsync();
                await Task.Run(() => Serializer.Serialize(filestream, timetable)
);
                Timetables.Add(timetable);
                return true;//return success
            }
        }

        /// <summary>
        /// Edit name or description of  timetable
        /// </summary>
        /// <param name="table">The edit target</param>
        /// <param name="rename">Whether it should be renamed</param>
        /// <param name="newName">The new name (if rename = true)</param>
        /// <returns></returns>
        public static async Task<bool> EditTimetable(Timetable table,bool rename
=false , string newName = "")
```

```
        {
            try
            {
                //get the file and overwrite with new data
                    var file = await DataFolder.GetFileAsync(table.Name + ".tb")
;
                    table.Name = newName;
                    Stream dataStream = await file.OpenStreamForWriteAsync();
                    await Task.Run(() => Serializer.Serialize(dataStream, table)
);

                //if it should rename , rename it asynchrnously
                if (rename)
                    await file.RenameAsync(newName + ".tb");
                return true;

            }
            catch
            {
                return false;//error
            }

        }

        /// <summary>
        /// Fills the list with all the timetables
        /// </summary>
        /// <returns></returns>
        public static async Task<bool> GetTimetables()
        {

            Timetables.Clear();//empty it just in case it has data
            var success = true;//assume it works

            var names = await DataFolder.GetFilesAsync(); //list of all files in
 data folder
                foreach (var name in names)
```

```
                {

                    Stream st = await name.OpenStreamForReadAsync();//open a rea
d-only data stream

                    //asynchrnously deserialize each file
                    await Task.Run(() =>
                    {
                        try
                        {
                            Timetable table = Serializer.Deserialize(st) as Time
table;

                            Timetables.Add(table);
                        }
                        catch
                        {
                            success = false;
                        }
                    });

                    st.Dispose();
                }
            return success;
        }

        /// <summary>
        /// This will delete a timetable
        /// </summary>
        /// <param name="table"></param>
        /// <returns></returns>
        public static async Task<bool> DeleteTimetable(Timetable table)
        {
            //bool success = true;
            try
            {
```

```
                //finds the file with that name and deletes. Then removes the ob
ject from the list.
                StorageFile file = await DataFolder.GetFileAsync(table.Name + ".
tb");
                await file.DeleteAsync();
                Timetables.Remove(table);
            }
            catch
            {
                return false;
            }
            return true;
        }

        /// <summary>
        /// Save a timetable over the old version
        /// </summary>
        /// <param name="table">The timetable to save</param>
        /// <returns></returns>
        public static async Task<bool> SaveTimetable(Timetable table)
        {
            try
            {
                //find timetable file , open for writing and serialize it
                StorageFile file = await DataFolder.CreateFileAsync(table.Name +
 ".tb", CreationCollisionOption.ReplaceExisting);
                Stream dataStream = await file.OpenStreamForWriteAsync();
                await Task.Run(() =>
                {
                    Serializer.Serialize(dataStream, table);
                });
                dataStream.Dispose();//to avoid two streams exceptions
                return true;
            }
            catch
            {
                return false;
```

```
            }
        }

        /// <summary>
        /// Export a timetable to a .tb file
        /// </summary>
        /// <param name="table"></param>
        /// <returns></returns>
        public static async Task<bool> ExportTimetable(Timetable table)
        {
            //file picker to let the use choose a file
            FileSavePicker saver = new FileSavePicker();
            saver.DefaultFileExtension = ".tb";
            saver.SuggestedFileName = table.Name;
            saver.SuggestedStartLocation = PickerLocationId.DocumentsLibrary;
            saver.FileTypeChoices.Add("Timetable file" , new List<string>{".tb"}
);//.tb is the file extension
            StorageFile file = await saver.PickSaveFileAsync();//le the use pick
a file
            //if the use did not press Cancel , it should not be null , so write
to it
            if (file != null)
            {
                Stream stream = await file.OpenStreamForWriteAsync();
                await Task.Run(() =>
                {
                    Serializer.Serialize(stream, table);
                });
            }
            return true;
        }

        /// <summary>
        /// Imports a .tb file
        /// </summary>
        /// <returns></returns>
        public static async Task<bool> ImportTimetable()
```

```
        {
            try
            {
                //let the user pick file
                FileOpenPicker opener = new FileOpenPicker();
                opener.FileTypeFilter.Add(".tb");
                opener.CommitButtonText = "Open Timetable";
                opener.SuggestedStartLocation = PickerLocationId.DocumentsLibrar
y;
                StorageFile file = await opener.PickSingleFileAsync();
                Stream str = await file.OpenStreamForReadAsync();
                bool doRename = false;
                //if the user changed the filename it could break the app so it
must be renamed
                await Task.Run(() =>
                {
                    Timetable table=Serializer.Deserialize(str) as Timetable;
                    if (file.Name != table.Name)
                    {
                        doRename = true;
                    }
                    Timetables.Add(table);
                    str.Dispose();
                });
                //copy it to the data folder
                file = await file.CopyAsync(DataFolder, Timetables.Last().Name +
 ".tb",NameCollisionOption.FailIfExists);
                //rename appropiately
                if (doRename) await file.RenameAsync(Timetables.Last().Name + ".
tb" , NameCollisionOption.ReplaceExisting);
                return true;
            }
            catch(Exception x)
            {
                return false;
            }
        }
```

```
    }
```

CORETIME.CS

```
/// <summary>
    /// Used to deal with time operations.
    /// </summary>
    public static class CoreTime
    {
        /// <summary>
        /// Check for the earliest time for space setting the Timetable view.
        /// </summary>
        /// <param name="tb"></param>
        /// <returns>Earliest Time</returns>
        public static Time GetEarliest(Timetable tb)
        {
            //This variable has to as big as possible
            Time earliest = Time.MaximumTime;

            foreach (Course t in tb.Courses)
            {
                //iterate through each Instance linearly
                foreach (Instance instance in t.Instances)
                {
                    //if this is earlier than the current earliest found time th
en it replaces it.
                    if (instance.StartTime <= earliest) earliest = instance.Star
tTime;
                }
            }
            //if nothing is earlier than the original value , MaximumTime is ret
urned.
            return earliest;
        }

        /// <summary>
        /// Check for the latest time for space setting the Timetable view.
```

```
        /// </summary>
        /// <param name="tb"></param>
        /// <returns>Latest time</returns>
        public static Time GetLatest(Timetable tb)
        {
            Time latest = Time.MinimumTime;
            foreach (Course t in tb.Courses)
            {
                foreach (Instance instance in t.Instances)
                {
                    if (instance.EndTime >= latest) latest = instance.EndTime;
                }
            }
            return latest;

        }

        /// <summary>
        /// This method checks whether a proposed instace is overlapping another
        /// </summary>
        /// <param name="instance">The instance we are checking the overlap with
</param>
        /// <param name="table">The Timetable in which we check for overlaps</pa
ram>
        /// <returns></returns>
        public static TimeOverlapData IsOverlapping(Instance instance, Timetable
 table)
        {
            foreach (Course model in table.Courses)
            {
                foreach (Instance current in model.Instances)
                {
                    if (instance.Day == current.Day)
                    {
                        if (instance != current && !model.Instances.Contains(ins
tance) && !(current.EndTime == instance.EndTime || current.StartTime == instance
.StartTime) )
                        {
```

```
                            if (instance.StartTime == current.StartTime) return
new TimeOverlapData { isOverlapping = true, OverlappingEventName = model.Name };
                            if (instance.StartTime < current.StartTime)
                            {
                                if (instance.EndTime > current.StartTime)
                                {
                                    return new TimeOverlapData { isOverlapping =
 true, OverlappingEventName = model.Name };
                                }
                            }
                            if (instance.StartTime >= current.StartTime && insta
nce.EndTime <= current.EndTime)
                            {
                                return new TimeOverlapData { isOverlapping = tru
e, OverlappingEventName = model.Name };
                            }
                        }
                    }
                }
            }
            return new TimeOverlapData{isOverlapping=false};
        }

        /// <summary>
        /// Formats a string into a easily readable format
        /// </summary>
        /// <param name="date">The date to format</param>
        /// <returns>The formatted string</returns>
        public static string FormatTime(DateTime date)
        {
            string final="";
            final += Constants.Months[date.Month - 1];
            final += " ";
            if (date.Day.ToString().EndsWith("1"))
            {
                final += date.Day.ToString() + "st";
            }
```

```
            else if (date.Day.ToString().EndsWith("2"))
            {
                final += date.Day.ToString() + "nd";
            }
            else if (date.Day.ToString().EndsWith("3"))
            {
                final += date.Day.ToString() + "rd";
            }
            else
            {
                final += date.Day.ToString() + "th";
            }
            final += "   " + date.Year.ToString();
            return final;
        }

        /// <summary>
        /// Converts a string of the form dd/mm/yyyy
        /// </summary>
        public static DateTime ParseString(string date)
        {
            /*
             d d / m m / y y y y
             0 1 2 3 4 5 6 7 8 9
             */
            try
            {
                date = date.Substring(0, 10); //cut the string just in case
                date.Trim();
                int day = int.Parse(date.Substring(0, 2)); //from index 0 ,take
2 characters
                int month = int.Parse(date.Substring(3, 2)); //from index 3 , ta
ke 2 characters
                int year = int.Parse(date.Substring(6, 4)); //from index 6 , tak
e 4 characters
                return new DateTime(year, month, day);
            }
```

```
            catch
            {
                return DateTime.Today;
            }
        }
    }

    /// <summary>
    /// A structure that holds the result of an overlapping check
    /// </summary>
    public struct TimeOverlapData
    {
        public bool isOverlapping;
        public string OverlappingEventName; //this is null if no overlap was fou
nd
    }

    /// <summary>
    /// Room that represents a time span.
    /// </summary>
    public class Time
    {
        public int Minutes;
        public int Hours;

        //standard comparasion operators
        public static bool operator >=(Time t1, Time t2)
        {
            return (t1.TotalMinutes >= t2.TotalMinutes);
        }
        public static bool operator <=(Time t1, Time t2)
        {
            return (t1.TotalMinutes <= t2.TotalMinutes);
        }

        public static bool operator >(Time t1, Time t2)
```

```
        {
            return (t1.TotalMinutes > t2.TotalMinutes);
        }
        public static bool operator <(Time t1, Time t2)
        {
            return (t1.TotalMinutes < t2.TotalMinutes);
        }

        public static Time operator +(Time t1, Time t2)
        {
            return new Time { Hours = t1.Hours + t2.Hours, Minutes = t1.Minutes
+ t2.Minutes };
        }

        public static Time operator -(Time t1, Time t2)
        {
            return new Time { Hours = t1.Hours - t2.Hours, Minutes = t1.Minutes
- t2.Minutes };
        }

        public static bool operator ==(Time t1 , Time t2)
        {
            return t1.Hours == t2.Hours && t1.Minutes == t2.Minutes;
        }

        public static bool operator !=(Time t1, Time t2)
        {
            return t1.Hours != t2.Hours || t1.Minutes != t2.Minutes;
        }

        public int TotalMinutes
        {
            get { return Hours * 60 + Minutes; }
        }

        //from 00:00 to 23:59
        public static Time MinimumTime = new Time { Hours = 0, Minutes = 0 };
```

```
        public static Time MaximumTime = new Time { Hours = 23, Minutes = 59 };

        //Useful for formatting , returns in hh:mm format
        public override string ToString()
        {
            string result="";
            result += Hours.ToString();
            result += ":" + Minutes.ToString();
            return result;
        }
    }
```

COURSE.CS

```
/// <summary>
    /// This class represents a course
    /// </summary>
    public class Course
    {
        //default values
        public string Name="";
        public string Room="";
        public string Teacher="";
        public string Notes="";
        public Color TileColor = Color.FromArgb(255,0,104,255); //the default co
lor is the color of the app icon/theme
        public List<Instance> Instances = new List<Instance>();
        public List<Module> Modules = new List<Module>();



    }


    /// <summary>
    /// This method is used to indicate the time and duration of each lesson. It
 is saved in the respective course.
```

```
    /// </summary>
    public class Instance
    {
        //setting default values
        public Time StartTime;
        public Time EndTime;
        public int Day;// 0...5
        public string EventName="";

    }
```

EDITABLEITEM.CS

```
/// <summary>
    /// This class can be inherited by any item that needs to show both a presen
tation and editing view.
    /// </summary>
    public class EditableItem : UserControl
    {
        List<FrameworkElement> Static = new List<FrameworkElement>();
        List<FrameworkElement> Editable = new List<FrameworkElement>();

        public EditableItem()
        {
            this.DoubleTapped +=(A,b) =>
                                    {
                                        setEditable(!edit);
                                        b.Handled = true;
                                    };
        }


        /// <summary>
        /// Set all the items that will be shown in edit mode.
        /// </summary>
        /// <param name="list">The list of editable items.</param>
```

```
        public void SetEditableItems(params FrameworkElement[] list)
        {
            Editable.Clear();
            foreach (FrameworkElement e in list)
            {
                Editable.Add(e);
            }
        }

        /// <summary>
        /// Set all the items that will be shown when not in edit mode.
        /// </summary>
        /// <param name="list">The list of static items.</param>
        public void SetStaticItems(params FrameworkElement[] list)
        {
            Static.Clear();
            foreach (FrameworkElement e in list)
            {
                Static.Add(e);
            }
        }




        public bool edit = false;

        /// <summary>
        /// Set the current presentation mode.
        /// </summary>
        public virtual void setEditable(bool isEditable)
        {
            edit = isEditable;
            //fade out elements that should not be shown based on the current ed
iting mode.
            foreach (FrameworkElement element in Static)
```

```
            {
                if (isEditable)
                {

                    AnimationHelper.FadeTo(element, 0, 0.5, true).Completed += (
a, b) => element.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
                }
                else
                {
                    AnimationHelper.FadeTo(element, 1, 0.5, true).Completed += (
a, b) => element.Visibility = Windows.UI.Xaml.Visibility.Visible;
                }

            }
            foreach (FrameworkElement element in Editable)
            {
                if (isEditable)
                {
                    AnimationHelper.FadeTo(element, 1, 0.5, true).Completed += (
a, b) => element.Visibility = Windows.UI.Xaml.Visibility.Visible;
                }
                else
                {
                    AnimationHelper.FadeTo(element, 0, 0.5, true).Completed += (
a, b) => element.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
                }
            }
        }
    }
```

TIMETABLE.CS

```
/// <summary>
    /// Represents a timetable , which includes
    /// </summary>
    public class Timetable
    {
```

```
        public string Name;
        public string Description;
        public AcademyBundle AcademyInfo=new AcademyBundle();
        public List<Course> Courses = new List<Course>();


        /// <summary>
        /// This method returns the total number of instances in the timetable
        /// </summary>
        /// <returns></returns>
        public int GetTotalInstances()
        {
            int total=0;
            foreach (Course model in Courses)
            {
                foreach (Instance instance in model.Instances)
                {
                    total++;
                }
            }
            return total;
        }

        /// <summary>
        /// This method returns whether the bundle timetable has a weekend or no
t.
        /// </summary>
        /// <returns></returns>
        public bool hasWeekend()
        {
            bool result = false;
            foreach (Course model in this.Courses)
            {
                foreach (Instance instance in model.Instances)
                {
                    if (instance.Day > 4) return true;
```

```
                }
            }
            return result;
        }
    }

    public class AcademyBundle
    {
        public string Name="";
        public string Location="";
        public string Website = "";
        public Color Accent = Color.FromArgb(255,8,180,0);
        public string Events="";
    }
```

VALIDATOR.CS

```
/// <summary>
    /// This class deals with validating strings. It also contains the validatio
n constants.
    /// </summary>
    public class Validator
    {

        #region Validation Constants
        //Those constants will be used for validating input
        public static int TimetableTitleMax = 30;
        public static int TimetableDescriptionMax = 200;

        public const int EventClassMax = 7;
        public const int EventNotesMax = 300;
        public const int EventTitleMax = 20;
        public const int EventTeacherMax = 25;

        public const int HomeworkDescriptionMax = 500;
```

```
        public const int ModuleNameMax = 20;

        public const int DefinitionNameMax = 35;
        public const int DefinitionDescriptionMax = 300;

        public const int StructureBlockNameMax = 30;
        public const int TopicTitleMax = 20;

        public const int NoteNameMax = 35;
        public const int NoteContentMax = 1000;

        public const int AcademyNameMax = 15;
        public const int AcademyLocationMax = 75;
        public const int AcademyWebsiteMax = 50;
        public const int AcademyEventsMax = 2000;

        #endregion


        /// <summary>
        /// This method is used to check whether a string meets certain validati
on rules. It can also autofix it and report it to the user.
        /// </summary>
        /// <param name="target">The string to check</param>
        /// <param name="scope">The context that describes the target (i.e Name)
</param>
        /// <param name="allowDigits">Whether the string is allowed to contain n
umbers</param>
        /// <param name="allowEmpty">Whether the string can be empty</param>
        /// <param name="max">The maximum number of characteers.Including whites
pace.</param>
        /// <param name="min">The minimum number of characteers.Including whites
pace.</param>
        /// <param name="autoFix">If true , automatically fix the string (remove
 digits , remove character over the max limit)</param>
        /// <param name="fixTarget">The holder of the target.</param>
        /// <param name="autoNotify">Whether the method should display the error
s found to the user. See ValidatorResponse.NotifyUser()</param>
        /// <param name="fileName">Whether the method should check if the string
 is a valid file path </param>
```

```
        /// <returns></returns>
        public static ValidatorResponse isStringValid(
            string target,string scope, bool allowDigits , bool allowEmpty, int
max = int.MaxValue,
            int min = int.MinValue , bool autoFix = false , TextBox fixTarget=nu
ll ,bool autoNotify = false , bool fileName = false)
        {
            var errors = new List<string>();
            if(target.Length > max) errors.Add("The "+scope+" should be less tha
n " + max + " characters.");
            if (target.Length < min) errors.Add("The " + scope + " should be mor
e than " + max+ " characters.");
            if (!allowDigits) if (target.ToCharArray().Any(char.IsDigit)) errors
.Add("The " + scope + " should not contain any numbers");
            if (!allowEmpty) if (target == "") errors.Add("The " + scope + " can
't be empty");
            if (fileName)
            {
                errors.AddRange(from char c in target
                                from k in System.IO.Path.GetInvalidFileNameChars
()
                                where c == k select
                                "Character " + c + " is invalid.");
            }

            var response = new ValidatorResponse { isValid = (errors.Count == 0)
, Errors = errors };

            if (autoNotify) response.NotifyUser();
            if (autoFix && fixTarget != null)
            {
                if (target.Length > max) fixTarget.Text = fixTarget.Text.Substri
ng(0, max);
                if (!allowDigits) if (target.ToCharArray().Any(char.IsDigit))
                    {
                        fixTarget.Text = new string(target.ToCharArray().Where(x
 => !char.IsDigit(x)).ToArray());
                    }
               // if (!allowEmpty) fixTarget.Text = "Enter a "+scope;
```

```
            }
            return response;
        }



    }


    /// <summary>
    /// This is the object returned by the Validator method.
    /// </summary>
    public struct ValidatorResponse
    {
        public bool isValid;
        public List<string> Errors;

        /// <summary>
        /// Groups and outputs the errors of this response to the user.
        /// </summary>
        public async void NotifyUser()
        {
            if (Errors.Count > 0)
            {
                var errorMessage = Errors.Aggregate("The following errors have b
een fixed for you:\n", (current, s) => current + (s + "\n"));

                var dialog =
                    new MessageDialog(content: errorMessage, title: "An error oc
cured");
                await dialog.ShowAsync();
            }
        }
    }
```

CLASSBOX.XAML.CS

```
public sealed partial class ClassBox : UserControl
    {
        Course self;//reference to the model of this class box
        Timetable selfTB;//reference to parent Timetable for deletion

        public ClassBox(
            Course model, UIElementCollection parent, Timetable tb)
        {
            this.InitializeComponent();
            self = model;
            selfTB = tb;

            //setting initial values
            name.Text = model.Name;
            teacher.Text = model.Teacher;
            room.Text = model.Room;
            notes.Text = model.Notes;
            color.SelectedColor = model.TileColor;
            border.BorderBrush = new SolidColorBrush(model.TileColor);

            //lambdas to handle changes
            color.ColorChanged += (col) =>
            {
                model.TileColor = col;
                border.BorderBrush = new SolidColorBrush(model.TileColor);
                deleteBtn.Foreground = new SolidColorBrush(model.TileColor);
            };

            //when text changes , perform a validation check/fix
            name.TextChanged += (a, b) =>
            {
                Validator.isStringValid(
                    target: name.Text, scope: "name", allowDigits: true, allowEm
pty: false, max: Validator.EventTitleMax,
                    autoFix: true, fixTarget: name, autoNotify: true);
                model.Name = name.Text;
```

```
            };

            notes.TextChanged += (a, b) =>
                                 {
                                     Validator.isStringValid(
                                                                    target: notes.T
ext, scope: "note", allowDigits: true, allowEmpty: true, max: Validator.EventNot
esMax,
                                                                    autoFix: true,
fixTarget: notes, autoNotify: true);
                    model.Notes = notes.Text;
                };
            room.TextChanged += (a, b) =>
            {
                Validator.isStringValid(
                    target: room.Text, scope: "room name", allowDigits: true, al
lowEmpty: false, max: Validator.EventClassMax,
                    autoFix: true, fixTarget: room, autoNotify: true);
                model.Room = room.Text;
            };
            teacher.TextChanged += (a, b) =>
            {
                Validator.isStringValid(
                    target: teacher.Text, scope: "teacher name", allowDigits: fa
lse, allowEmpty: false, max: Validator.EventTeacherMax,
                    autoFix: true, fixTarget: teacher, autoNotify: true);
                model.Teacher = teacher.Text;
            };

            //lambdas to handle tap events (delete/add)
            deleteBtn.Tapped += async (a, b) =>
            {
                MessageDialog diag = new MessageDialog("This action cannot be un
done", "Are you sure you want to delete " + model.Name);
                diag.Commands.Add(new UICommand("Delete", (d) =>
                {
                    parent.Remove(this);
                    tb.Courses.Remove(model);
```

```
                }));
                diag.Commands.Add(new UICommand("Cancel"));
                await diag.ShowAsync();

            };



            add.Tapped += (a, b) =>
            {
                Instance inst = new Instance { Day = 0, StartTime = new Time { H
ours = 9, Minutes = 0 }, EndTime = new Time { Hours = 10, Minutes = 0 }, EventNa
me = self.Name };
                self.Instances.Add(inst);
                RefreshInstances();
            };

            RefreshInstances();
        }

        //this method recreates the content of the holder after an update to the
 data source
        void RefreshInstances()
        {
            holder.Children.Clear();
            foreach (Instance c in self.Instances)
            {
                InstanceControl cont = new InstanceControl(c, selfTB);
                holder.Children.Add(cont);
                cont.OnDelete += (a, b) =>
                {
                    holder.Children.Remove(a);
                    self.Instances.Remove(b);
                };
            }
        }
    }
```

CLASSESEDITOR.XAML.CS

```
/// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame
.
    /// </summary>
    public sealed partial class ClassesEditor : Page
    {
        public ClassesEditor()
        {
            this.InitializeComponent();
        }

        Timetable self;
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            self = e.Parameter as Timetable;//keep a reference to the timetable
            RefreshCourses();

            //go back to previous page
            back.Tapped += async (a, b) =>
            {
                await CoreData.SaveTimetable(self);
                Frame.Navigate(typeof(MainPage), self);
            };

            //create new course
            add.Tapped += (a, b) =>
            {
                self.Courses.Add(new Course { Name ="New Room" });
                RefreshCourses();
            };
        }

        //redraw all courses
```

```
        void RefreshCourses()
        {
            holder.Children.Clear();
            foreach (Course model in self.Courses)
            {
                ClassBox box = new ClassBox(model , holder.Children , self);
                box.VerticalAlignment = Windows.UI.Xaml.VerticalAlignment.Center
;
                holder.Children.Add(box);
            }
        }
    }
```

DEFINITIONCONTROL.XAML.CS

```
 public sealed partial class DefinitionControl : EditableItem
    {



        public Definition self;
        public DefinitionControl(Definition def, Windows.UI.Color c, object cont
ainer = null, Module module = null)
        {
            this.InitializeComponent();
            //set default values
            root.Background = new SolidColorBrush(c);
            self = def;
            SetEditableItems(definitionEdit, titleEdit);
            SetStaticItems(definition, title);
            title.Text = def.Caption;
            titleEdit.Text = def.Caption;
            definition.Text = def.Content;
            definitionEdit.Text = def.Content;

            //lambdas for change events
            titleEdit.TextChanged += (e, ev) =>
            {
```

```
                Validator.isStringValid(
                    target: titleEdit.Text, scope: "name of the definition", all
owDigits: true, allowEmpty: true, max: Validator.DefinitionNameMax,
                    autoFix: true, fixTarget: titleEdit, autoNotify: true);
                title.Text = titleEdit.Text;
                self.Caption = titleEdit.Text;
                //OnChange();
            };
            definitionEdit.TextChanged += (e, ev) =>
            {
                Validator.isStringValid(
                    target: definitionEdit.Text, scope: "definition", allowDigit
s: true, allowEmpty: true, max: Validator.DefinitionDescriptionMax,
                    autoFix: true, fixTarget: definitionEdit, autoNotify: true);
                definition.Text = definitionEdit.Text;
                self.Content = definitionEdit.Text;
               // OnChange();
            };

            //internal lambda to remove itself from data source
            deleteButton.Click += (e, ev) =>
            {
                //OnDelete(this);
                if (container.GetType() == typeof(UIElementCollection))
                {
                    (container as UIElementCollection).Remove(this);
                }
                else
                {
                    (container as ItemCollection).Remove(this);
                }
                module.Definitions.Remove(self);
            };
        }
    }
```

DEFINITIONSEDITOR.XAML.CS

```
/// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame
.
    /// </summary>
    public sealed partial class DefinitionsEditor : Page
    {
        public DefinitionsEditor()
        {
            this.InitializeComponent();

        }

        Course self;
        protected async override void OnNavigatedTo(NavigationEventArgs e)
        {

            self = (e.Parameter as object[])[1] as Course;//keep for reference

            //set default values
            title.Foreground = new SolidColorBrush(self.TileColor);
            back.Foreground = new SolidColorBrush(self.TileColor);
            add.Foreground = new SolidColorBrush(self.TileColor);
            edit.Foreground = new SolidColorBrush(self.TileColor);
            pin.Foreground = new SolidColorBrush(self.TileColor);
            title.Text = self.Name + " definitions";

            //Handle pinning to start screen
             bool isPinned = false;
             foreach(SecondaryTile tl in  await SecondaryTile.FindAllForPackageA
sync()){
                if (tl.TileId == self.Name.Replace(" ", "") + "definitions")
                {
                    pin.Icon = "";
                    isPinned=true;
                }
```

```
            }

            pin.Tapped += async (a, b) =>
            {
                Uri logo = new Uri("ms-appx:///Assets/TileLogo.png");//icon
                SecondaryTile tile = new SecondaryTile();
                tile.BackgroundColor = self.TileColor;//course color
                tile.TileId = self.Name.Replace(" ", "") + "definitions";//uniqu
e id
                tile.Logo = logo;
                tile.ForegroundText = ForegroundText.Light;
                tile.DisplayName = "DefinitionsEditor for " + self.Name;
                tile.ShortName = self.Name + " definitions";
                tile.TileOptions = TileOptions.ShowNameOnLogo;
                tile.Arguments = "definitions," + ((e.Parameter as object[])[0]
as Timetable).Name + "," + self.Name;//used to direct to appropiate page
                if (!isPinned)
                {
                    isPinned = await tile.RequestCreateAsync();
                    pin.Icon = "";
                }
                else
                {
                    isPinned = !await tile.RequestDeleteAsync();
                    pin.Icon = "";
                }
            };



            back.Tapped += async(a, b) =>
            {
                await CoreData.SaveTimetable((e.Parameter as object[])[0] as Tim
etable);
                Frame.Navigate(typeof(MainPage), (e.Parameter as object[])[0]);
            };
```

```
            //this will enable filtering by module
            moduleHolder.fill(self,typeof(DefinitionsEditor));
            moduleHolder.OnChosen += (module) =>
            {
                RefreshDefinitions();
            };
            RefreshDefinitions();



            add.Tapped += (q, b) =>
            {
                Definition def = new Definition { Caption="Title here" , Content
="Definition or explanation goes here!" };
                moduleHolder.Chosen.Definitions.Add(def);
                RefreshDefinitions();
            };

            //change the editable mode of each box
            edit.Tapped += (a, b) =>
            {
                if ((edit.Foreground as SolidColorBrush).Color == Windows.UI.Col
ors.Black)
                {
                    foreach (EditableItem box in holder.Children) box.setEditabl
e(false);
                    edit.Foreground = new SolidColorBrush(self.TileColor);
                }
                else
                {
                    foreach (EditableItem box in holder.Children) box.setEditabl
e(true);
                    edit.Foreground = new SolidColorBrush(Windows.UI.Colors.Blac
k);
                }
            };
        }
```

```
        //redraw all boxes
        void RefreshDefinitions()
        {
            if (moduleHolder.Chosen != null)
            {
                holder.Children.Clear();
                foreach (Definition def in moduleHolder.Chosen.Definitions)
                {
                    DefinitionControl dc = new DefinitionControl(def,self.TileCo
lor, holder.Children, moduleHolder.Chosen);
                    holder.Children.Add(dc);
                }
                if (holder.Children.Count == 0) holder.Children.Add(new Image {
HorizontalAlignment = Windows.UI.Xaml.HorizontalAlignment.Center , VerticalAlign
ment = Windows.UI.Xaml.VerticalAlignment.Top, Width = 600 , Height = 600 , Sourc
e = new Windows.UI.Xaml.Media.Imaging.BitmapImage(new Uri("ms-
appx:///Assets/empty.png"))});
            }
        }
    }
```

HOMEWORKEDITOR.XAML.CS

```
/// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame
.
    /// </summary>
    public sealed partial class HomeworkEditor : Page
    {
        public HomeworkEditor()
        {
            this.InitializeComponent();
        }

        Course self;
        protected async override void OnNavigatedTo(NavigationEventArgs e)
        {
            self = (e.Parameter as object[])[1] as Course;
```

```
            title.Foreground = new SolidColorBrush(self.TileColor);
            back.Foreground = new SolidColorBrush(self.TileColor);
            add.Foreground = new SolidColorBrush(self.TileColor);

            title.Text = self.Name + " homework";

            bool isPinned = false;
            foreach (SecondaryTile tl in await SecondaryTile.FindAllForPackageAs
ync())
            {
                if (tl.TileId == self.Name.Replace(" ", "") + "homework")
                {
                    pin.Icon = "";
                    isPinned = true;
                }
            }
            pin.Tapped += async (a, b) =>
            {
                Uri logo = new Uri("ms-appx:///Assets/TileLogo.png");
                SecondaryTile tile = new SecondaryTile();
                tile.BackgroundColor = self.TileColor;
                tile.TileId = self.Name.Replace(" ", "") + "homework";
                tile.Logo = logo;
                tile.ForegroundText = ForegroundText.Light;
                tile.DisplayName = "HomeworkEditor for " + self.Name;
                tile.ShortName = self.Name + " homework";
                tile.TileOptions = TileOptions.ShowNameOnLogo;
                tile.Arguments = "homework," + ((e.Parameter as object[])[0] as
Timetable).Name + "," + self.Name;
                if (!isPinned)
                {
                    isPinned = await tile.RequestCreateAsync();
                    pin.Icon = "";
                }
                else
                {
```

```
                isPinned = !await tile.RequestDeleteAsync();
                pin.Icon = "";
            }
        };
        back.Tapped += async (a, b) =>
        {
            await CoreData.SaveTimetable((e.Parameter as object[])[0] as Tim
etable);
            Frame.Navigate(typeof(MainPage), (e.Parameter as object[])[0]);
        };

        moduleHolder.fill(self,typeof(HomeworkEditor));
        moduleHolder.OnChosen += (module) =>
        {
            RefreshHomework();
        };
        RefreshHomework();



        add.Tapped += (q, b) =>
        {
            HomeworkTask bl = new HomeworkTask();
            moduleHolder.Chosen.Homeworks.Add(bl);
            RefreshHomework();
        };
    }

    void RefreshHomework()
    {
        if (moduleHolder.Chosen != null)
        {
            holder.Children.Clear();
            foreach (HomeworkTask def in moduleHolder.Chosen.Homeworks)
            {
                HomeworkItem dc = new HomeworkItem(def, self.TileColor, hold
er.Children, moduleHolder.Chosen);
```

```
                    holder.Children.Add(dc);
                }
                if (holder.Children.Count == 0) holder.Children.Add(new Image {
HorizontalAlignment = Windows.UI.Xaml.HorizontalAlignment.Center, VerticalAlignm
ent = Windows.UI.Xaml.VerticalAlignment.Top, Width = 600, Height = 600, Source =
 new Windows.UI.Xaml.Media.Imaging.BitmapImage(new Uri("ms-
appx:///Assets/empty.png")) });

            }
        }
    }
```

HOMEWORKITEM.XAML.CS

```
public sealed partial class HomeworkItem : UserControl
    {
        public HomeworkTask self;
        public HomeworkItem(HomeworkTask task , Windows.UI.Color c, UIElementCol
lection container = null, Module module = null)
        {
            this.InitializeComponent();
            self = task;

            //initial values
            root.Background = new SolidColorBrush(c);
            _description.Text = self.Notes;
            completedSwitch.IsOn = self.Completed;
            dateButton.Content = CoreTime.FormatTime(task.DueOn);



            deleteButton.Tapped += (e, ev) =>
            {
                //OnDelete(this);
                container.Remove(this);
                module.Homeworks.Remove(self);
            };
            completedSwitch.Toggled += (e, ev) =>
```

```
            {
                self.Completed = completedSwitch.IsOn;
               // OnChange();
            };
            _description.TextChanged += (e, ev) =>
            {
                Validator.isStringValid(
                    target: _description.Text, scope: "description", allowDigits
: true, allowEmpty: true, max: Validator.HomeworkDescriptionMax,
                    autoFix: true, fixTarget: _description, autoNotify: true);
                self.Notes = _description.Text;
                //OnChange();
            };

            dateButton.Click += (e, ev) =>
            {
                introDate.Begin();
            };
            datePicker.BackPressed += (time) =>
            {
                exitDate.Begin();
                dateButton.Content = CoreTime.FormatTime(time);
                self.DueOn = time;
                //OnChange();
            };
        }
    }
```

SIMPLEDATEPICKER.XAML.CS

```
public sealed partial class SimpleDatePicker : UserControl
    {
        public delegate void BackPressedEventHandler(DateTime time);
        public event BackPressedEventHandler BackPressed;
```

```
        public DateTime PickedTime = new DateTime();
        public SimpleDatePicker()
        {
            this.InitializeComponent();
            this.Loaded += (e, ev) =>
            {
                //Realistically , the user will not set homework that is more th
an 1 year apart (i.e in december 2013 , he may set January 2014 homework)
                year.Items.Add(new ListBoxItem {Content = "This year ("+ DateTim
e.Now.Year.ToString() +")"});
                year.Items.Add(new ListBoxItem { Content = "Next year (" + DateT
ime.Now.Year.ToString() + ")" });
            };

            //fill the month boxes with true for next year or false for last yea
r
            year.SelectionChanged += (e, ev) => fillBoxes((year.SelectedIndex !=
 0));

            //return chosen date
            backButton.Click += (e, ev) =>
            {
                int yr = (year.SelectedIndex == 0) ? DateTime.Now.Year : DateTim
e.Now.AddYears(1).Year;
                PickedTime = new DateTime(yr,Month , Day);
                BackPressed(PickedTime);
            };
        }

        public void fillBoxes(bool nextYear)
        {
            month.Items.Clear();

                //if next year , start from January , else start from this month
            //add the month labels to the box
                for (int x = (nextYear)?0:DateTime.Now.Month-1; x < 12; x++)
                {
```

```
                    month.Items.Add(new ListBoxItem { Content = Constants.Months
[x] });
                }
            //when a month is selected
                month.SelectionChanged += (e, ev) =>
                {
                    day.Items.Clear();
                    //if this year , start from tomorrow , else from 1st day of
month
                    //go up to the last day in the month
                    for (int y = (Month == DateTime.Now.Month) ? DateTime.Now.Da
y + 1 : 1; y <= DateTime.DaysInMonth(DateTime.Now.AddYears(1).Year, Month); y++)
                    {
                        day.Items.Add(new ListBoxItem { Content = y.ToString() }
);
                    }
                };



        }

        //standard properties for getting values
        int Day
        {
            get { try { return int.Parse((day.SelectedItem as ListBoxItem).Conte
nt as string); } catch { return 1; } }
        }

        int Month
        {
            get {
                for(int x= 0; x< Constants.Months.Count();x++)
                {
                    if (Constants.Months[x] == (month.SelectedItem as ListBoxIte
m).Content as string) return x + 1;
                }
                return 1;
            }
```

```
        }
    }
```

INTRO.XAML.CS

```
/// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame
.
    /// </summary>
    public sealed partial class Intro : Page
    {
        public Intro()
        {
            this.InitializeComponent();
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            start.Tapped += (a, b) =>
            {
                Frame.Navigate(typeof(StartPage));
            };
        }
    }
```

DEFINITIONPREVIEW.XAML.CS

```
public sealed partial class DefinitionPreview : UserControl
    {

        public delegate void OnOpenHandler(Course model);
        public event OnOpenHandler OnOpen;


        public DefinitionPreview(Course model)
        {
```

```
            this.InitializeComponent();
            this.Loaded += (a, b) => { if (model.Modules.Count == 0) (this.Paren
t as StackPanel).Children.Remove(this); };
            //add each definition to the list for random picking
            foreach (Module m in model.Modules)
            {
                foreach (Definition def in m.Definitions) defs.Add(def);
            }



            name.Text = model.Name;
            header.Background = new SolidColorBrush(model.TileColor);
            edit.Tapped += (a, b) => OnOpen(model);
            PickRandomDefinition(model);
            refresh.Tapped += (a, b) => PickRandomDefinition(model);
        }

        //this method will return a random definition to preview
        List<Definition> defs = new List<Definition>();
        Random rnd = new Random();//random generator
        void PickRandomDefinition(Course model)
        {
            //only there are any definitions
            if (defs.Count > 0)
            {
                Definition def = defs[rnd.Next(0, defs.Count)];
                title.Text = def.Caption;
                content.Text = def.Content;
            }
            else
            {
                title.Text = "";
                content.Text = "None of the modules have any definitions yet.\n
Try adding some by tapping the pencil icon on the top right of this box";
            }
        }
```

```
    }
```

HOMEWORKPREVIEW.XAML.CS

```
public sealed partial class HomeworkPreview : UserControl
    {
        public delegate void OnOpenHandler(Course model);
        public event OnOpenHandler OnOpen;

        public HomeworkPreview(Course model)
        {
            this.InitializeComponent();
            this.Loaded += (a, b) => { if (model.Modules.Count == 0) (this.Paren
t as StackPanel).Children.Remove(this); };
            name.Text = model.Name;
            header.Background = new SolidColorBrush(model.TileColor);
            edit.Tapped += (a, b) => OnOpen(model);

            //The user might want to see which module the homework is for
            //therefore we must create a list of homework with the module name a
ttached
            List<HomeworkData> data = new List<HomeworkData>();
            foreach (Module m in model.Modules) foreach (HomeworkTask tk in m.Ho
meworks)
                {
                    data.Add(new HomeworkData { homework = tk, ModuleName = m.Na
me });
                }

            //perform bubble sort to list them from the closest
            //while the list is unordered , perform the algorithm
            while (!isOrdered(data))
            {
                //one complete for loop is one complete pass
                for (int i = 0; i < data.Count -1; i++)
                {
                    //if a date is later than one on its right then they must be
 swapped
```

```
                    if (data[i].homework.DueOn > data[i+1].homework.DueOn)
                    {
                        swap(data, i, i + 1);
                    }
                }
            }

            //10 homeworks is enough for the preview:
            for (int i = 0; i < ((data.Count>10 ) ? 10: data.Count); i++)
            {
                HomeworkPreviewComponent cont = new HomeworkPreviewComponent(dat
a[i].homework, data[i].ModuleName , model.TileColor);
                holder.Children.Add(cont);
            }
        }

        //handy struct to associate homework with the module
        struct HomeworkData
        {
            public HomeworkTask homework;
            public string ModuleName;
        }

        //this will return true when the homeworks are sorted
        bool isOrdered(List<HomeworkData> hw)
        {
            //check each adjecent pair to see if they are in order
            for (int i = 0; i < hw.Count -1 ; i++)
            {
                var current = hw[i];//the current HomeworkData
                var next = hw[i+1];//the adjecent one
                if (current.homework.DueOn > next.homework.DueOn) return false;
            }
            //this statement will only be reached if false is not returned befor
e the checking is complete
            return true;
        }
```

```
        //swaps two indexes using a temporary value
        void swap(List<HomeworkData> hw, int a, int b)
        {
            //we need a temporary value to hold one of the values while switchin
g
            HomeworkData temp = hw[a];
            hw[a] = hw[b];
            hw[b] = temp;
        }
    }
```

HOMEWORKPREVIEWCOMPONENT.XAML.CS

```
public sealed partial class HomeworkPreviewComponent : UserControl
    {
        public HomeworkPreviewComponent(HomeworkTask hw,string module_name , Col
or c)
        {
            this.InitializeComponent();
            //initial values
            module.Text = module_name;
            content.Text = hw.Notes;
            module.Foreground = new SolidColorBrush(c);
            border.Background = new SolidColorBrush(c);
            int left = (hw.DueOn - DateTime.Now).Days;
            if (left == 0) time.Text = "Today";
            else time.Text = left.ToString() + " days left";

            if (left < 0) time.Text = "OVERDUE"; //negative left means due date
passed.

            //set colors to indicate proximity of date
            if (left < 5)
            {
                time.Foreground = new SolidColorBrush(Colors.Red);
            }
```

```
            else if (left < 14)
            {
                time.Foreground = new SolidColorBrush(Colors.Orange);
            }
            else
            {
                time.Foreground = new SolidColorBrush(Colors.Green);
            }
        }
    }
```

MODULEPREVIEW.XAML.CS

```
public sealed partial class ModulePreview : UserControl
    {
        public delegate void OnOpenHandler(Course model);
        public event OnOpenHandler OnOpen;

        public ModulePreview(Course model)
        {
            this.InitializeComponent();

            //initial values
            name.Text = model.Name;
            header.Background = new SolidColorBrush(model.TileColor);
            edit.Tapped += (a, b) => OnOpen(model);


            holder.SelectionChanged += (a, b) =>
            {
                count.Text = (holder.SelectedIndex + 1).ToString() + "/" + holde
r.Items.Count.ToString();
            };

            //add each module to the flipview as a modulepreviewcomponent
            //so the user can flick through them
```

```
            foreach (Module m in model.Modules)
            {
                ModulePreviewComponent mpc = new ModulePreviewComponent(m , mode
l.TileColor);
                FlipViewItem item = new FlipViewItem { Content = mpc };
                holder.Items.Add(item);
            }
            count.Text = (holder.SelectedIndex + 1).ToString() + "/" + holder.It
ems.Count.ToString();
        }
    }
```

MODULEPREVIEWCOMPONENT.XAML.CS

```
public ModulePreviewComponent(Module m , Windows.UI.Color col)
        {
            this.InitializeComponent();
            //set the shade color
            name.Foreground = new SolidColorBrush(col);
            name.Text = m.Name;//set the name
            //show the amount of each info type this module has
            homework.Text = m.Homeworks.Count.ToString();
            definitions.Text = m.Definitions.Count.ToString();
            notes.Text = m.Notes.Count.ToString();
            topics.Text = m.StructureBlocks.Count.ToString();
        }
```

NOTEPREVIEW.XAML.CS

```
public sealed partial class NotePreview : UserControl
    {
        public delegate void OnOpenHandler(Course model);
        public event OnOpenHandler OnOpen;

        public NotePreview(Course model)
        {
            this.InitializeComponent();
```

```
            //hide if no modules are found
            this.Loaded += (a, b) => { if (model.Modules.Count == 0) (Parent as
StackPanel).Children.Remove(this); };
           //color setting
            header.Background = new SolidColorBrush(model.TileColor);
            edit.Tapped += (a, b) => OnOpen(model);
            //this LINQ expression will get the number of all notes
            int c = model.Modules.SelectMany(mod => mod.Notes).Count();
            count.Text = c.ToString() + " notes";
            name.Text = model.Name;
        }
    }
```

PIECHART.XAML.CS

```
public sealed partial class PieChart : UserControl
    {
        //the outer rings
        Polyline curve = new Polyline { StrokeThickness =30 };
        Polyline curve2 = new Polyline { StrokeThickness = 30 };

        //small rings to show the outline
        Polyline back1 = new Polyline { StrokeThickness = 2 };
        Polyline back2 = new Polyline { StrokeThickness = 2 };
        Polyline back3 = new Polyline { StrokeThickness = 2 };

        static double unit = 340; // the width /height

        public PieChart()
        {
            this.InitializeComponent();
            //add everything to the canvas
            root.Children.Add(curve);
            root.Children.Add(curve2);
            root.Children.Add(back1);
```

```
            root.Children.Add(back2);
            root.Children.Add(back3);

            this.Loaded += (a, b) =>
            {
                //initialize
                back1.Stroke = new SolidColorBrush(Colors.Gray);
                back2.Stroke = new SolidColorBrush(Colors.Gray);
                back3.Stroke = new SolidColorBrush(Colors.Gray);

                //set the outlines in the back
                Canvas.SetZIndex(back1, -3);
                Canvas.SetZIndex(back2, -3);
                Canvas.SetZIndex(back3, -3);

                Point center = new Point(unit / 2, unit / 2);

                double radius = 0;
                //go through the whole circle
                for (int i = 0; i <= 360; i++)
                {
                    //convert to radians
                    double angle = i * Math.PI / 180;

                    //use formula y = centerY+rsin(theta) , x = centerX + rcos(t
heta)
                    radius = unit / 2 -15;
                    Point p = new Point(center.X + radius * Math.Cos(angle), cen
ter.Y + radius * Math.Sin(angle));
                    back1.Points.Add(p);

                    radius = unit / 2 - 45;
                    Point p2 = new Point(center.X + radius * Math.Cos(angle), ce
nter.Y + radius * Math.Sin(angle));
                    back2.Points.Add(p2);

                    radius = unit / 2 - 75;
```

```
                    Point p3 = new Point(center.X + radius * Math.Cos(angle), ce
nter.Y + radius * Math.Sin(angle));
                    back3.Points.Add(p3);
                }
            };
        }

        //note:angle 0 is on the right so we will do angleD -
90 to start at the top and going clockwise

        //draws the outer circle
        public void DrawWithAngle(double angleD , Color c){
            c.R = (byte)(c.R * 0.8);
            c.B = (byte)(c.B * 0.8);
            c.G = (byte)(c.G * 0.8);
                curve.Stroke = new SolidColorBrush(c);
                Point center = new Point(unit / 2, unit / 2);

                double radius = unit / 2 - 30;

                for (int x = -90; x < angleD-90; x++)
                {
                    double angle = x * Math.PI / 180;
                    Point p = new Point(center.X + radius * Math.Cos(angle), cen
ter.Y + radius * Math.Sin(angle));
                    curve.Points.Add(p);

                }
        }

        //draws the inner circle
        public void DrawInnerWithAngle(double angleD, Color c)
        {

            curve2.Stroke = new SolidColorBrush(c);
            Point center = new Point(unit / 2, unit / 2);
```

```
            double radius = unit / 2-60;

            for (int x = -90; x < angleD - 90; x++)
            {
                double angle = x * Math.PI / 180;
                Point p = new Point(center.X + radius * Math.Cos(angle), center.
Y + radius * Math.Sin(angle));
                curve2.Points.Add(p);
            }
        }
    }
```

STRUCTUREPREVIEW.XAML.CS

```
public sealed partial class StructurePreview : UserControl
    {
        public delegate void OnOpenHandler(Course model);
        public event OnOpenHandler OnOpen;
        public StructurePreview(Course model)
        {
            this.InitializeComponent();
            this.Loaded += (a, b) => { if (model.Modules.Count == 0) (this.Paren
t as StackPanel).Children.Remove(this); };

            name.Text = model.Name;
            edit.Tapped += (a, b) => OnOpen(model);
            root.Background = new SolidColorBrush(model.TileColor);
            int total =0 , s = 0, r = 0; //s is the number studied and r is the
number revised
            //iterate through every topic and increment accordingly
            foreach (Module mod in model.Modules)
            {
                foreach (StructureBlock block in mod.StructureBlocks)
                {
                    foreach (Topic tp in block.Topics)
                    {
                        total++;
```

```
                    if (tp.Studied) s++;
                    if (tp.Revised) r++;
                }
            }
        }
        this.Loaded += (a, b) =>
        {
            topics.Text = total.ToString() + " topics";
            //if studied >0
            if (s != 0)
            {
                studied.Text = (s * 100 / total).ToString() + "%";
                Color c = model.TileColor;
                c.R = (byte)(c.R * 0.8);//dime the color a bit for nice cont
rast
                c.B = (byte)(c.B * 0.8);
                c.G = (byte)(c.G * 0.8);
                studied.Foreground = new SolidColorBrush(c);
                //draw the arc to symbolize the percentage studied , the ang
le is the same as the ratio of studied to total
                studiedPie.DrawWithAngle((double)s / total * 360 , model.Til
eColor);

            }
            if (r != 0)
            {
                revised.Foreground = new SolidColorBrush(model.TileColor);
                revised.Text = (r * 100 / total).ToString() + "%";
                studiedPie.DrawInnerWithAngle(((double)r / total) * 360 , mo
del.TileColor);
            }


        };
    }
}
```

TIMETABLEVIEW.XAML.CS

```
public sealed partial class TimetableView : UserControl
    {
        public TimetableView()
        {
            this.InitializeComponent();
            this.Loaded += (E, EV) =>
            {
                //draw the day name labels
                for (int x = 0; x < 7; x++)
                {
                    TextBlock tb = new TextBlock { FontSize = 15 };
                    switch (x)
                    {
                        case 0:
                            tb.Text = "Monday";
                            break;
                        case 1:
                            tb.Text = "Tuesday";
                            break;
                        case 2:
                            tb.Text = "Wendsday";
                            break;
                        case 3:
                            tb.Text = "Thursday";
                            break;
                        case 4:
                            tb.Text = "Friday";
                            break;
                        case 5:
                            tb.Text = "Saturday";
                            break;
                        case 6:
                            tb.Text = "Sunday";
                            break;
                    }
```

```
                    labelContainer.Children.Add(tb);//this holds the labels
                    //place the label at the appropiate position
                    Canvas.SetLeft(tb, x * Constants.EventControlWidth + Constan
ts.EventControlWidth / 2 - tb.ActualWidth);
                    Canvas.SetTop(tb, 0);
                }
            };




        }

        //line definition
        Rectangle TimeLine = new Rectangle { Height = 5, Width = Constants.Event
ControlWidth, Fill = new SolidColorBrush(Colors.Red) };
        public void DrawCurrentTime()
        {
            //add it if it's not already there
           if(!root.Children.Contains(TimeLine)) root.Children.Add(TimeLine);



            var time = new Time {Hours = DateTime.Now.Hour, Minutes = DateTim
e.Now.Minute};
            //if current time is within the bounds of the timetable view , dr
aw it in
            if (lowerBound < time && upperBound > time)
            {
                double ratio = (double) (time.TotalMinutes - lowerBound.Total
Minutes)/(upperBound.TotalMinutes - lowerBound.TotalMinutes);
                Canvas.SetLeft(TimeLine, DateTime.Now.Day * Constants.EventCo
ntrolWidth);
                Canvas.SetTop(TimeLine, ratio *  ((upperBound.TotalMinutes -
lowerBound.TotalMinutes) / 60 * Constants.EventControlHeight + 43));
                Canvas.SetZIndex(TimeLine , 3);
            }



        }
```

```
        //list for each day
        List<EventControl>[] instances = new List<EventControl>[]{
        new List<EventControl>(),
        new List<EventControl>(),
        new List<EventControl>(),
        new List<EventControl>(),
        new List<EventControl>(),
         new List<EventControl>(),
        new List<EventControl>()};

        //earliest and latest time
        private Time lowerBound;
        private Time upperBound;
        public void FillWith(Timetable timetable)
        {
            timeLabels.Children.Clear();
            root.Children.Clear();
            if (timetable.Courses.Count > 0 && timetable.GetTotalInstances() > 0
)
            {
                lowerBound = CoreTime.GetEarliest(timetable);
                upperBound = CoreTime.GetLatest(timetable);

                //set size based on whether there is a weekedend or not
                labelContainer.Width = ((timetable.hasWeekend()) ? 7 : 5) * Cons
tants.EventControlWidth;
                this.Width = Constants.EventControlWidth * ((timetable.hasWeeken
d()) ? 7 : 5) + 43;

                this.Height = (upperBound.TotalMinutes - lowerBound.TotalMinutes
) / 60 * Constants.EventControlHeight + 43;


                for (int x = 0; x < timetable.Courses.Count; x++)
                {
```

```
                    Course model = timetable.Courses[x];
                    foreach (Instance instance in model.Instances)
                    {
                        //foreach instance we
                        EventControl cont = new EventControl(model);//create a c
ontrol
                        cont.setTime(instance.StartTime, instance.EndTime);//set
 the text
                        instances[instance.Day].Add(cont);//add it to the approp
iate list
                        root.Children.Add(cont);//add it to the view
                        Canvas.SetLeft(cont, Constants.EventControlWidth * insta
nce.Day);//set it at the x coord and y coord
                        Canvas.SetTop(cont, ((instance.StartTime.TotalMinutes -
lowerBound.TotalMinutes) / 60.0) * Constants.EventControlHeight);//take lenght i
n account (i.e 2 hour lesson)
                        double dt = (instance.EndTime.TotalMinutes - instance.St
artTime.TotalMinutes);
                        double height = dt / 60;
                        cont.Height = height * Constants.EventControlHeight; //s
et the height proportionately to the duration
                    }
                }

                //create the time labels and the outline lines
                for (int x = lowerBound.TotalMinutes; x <= upperBound.TotalMinut
es; x += 30)
                {
                    TextBlock block = new TextBlock { FontSize = (x % 60 == 0) ?
 12 : 10 };
                    //block.Text = string.Format("{0}:{1}", ((int)x / 60).ToStri
ng(), ((x % 60)).ToString());
                    // block.Text = (x / 60).ToString() + ":00";
                    double hours = x / 60.0;
                    block.Text = Math.Truncate(hours).ToString() + ":" + ((hours
 - Math.Truncate(hours)) * 60).ToString();
                    timeLabels.Children.Add(block);
                    Canvas.SetLeft(block, 0);
                    Canvas.SetTop(block, Constants.EventControlHeight * ((x - lo
werBound.TotalMinutes) / 60.0) - block.ActualHeight);
```

```
                    Line indicator = new Line { Stroke = new SolidColorBrush(Col
ors.DarkGray), StrokeThickness = 1 };
                    root.Children.Add(indicator);
                    indicator.X1 = 0;
                    indicator.Y1 = Constants.EventControlHeight * ((x - lowerBou
nd.TotalMinutes) / 60.0);
                    indicator.X2 = Constants.EventControlWidth * ((timetable.has
Weekend()) ? 7 : 5) + timeLabels.ActualWidth;
                    indicator.Y2 = indicator.Y1;
                    Canvas.SetZIndex(indicator, -1);
                }



            }
            //draw a red line which shows the current time
            var timer = new DispatcherTimer { Interval = TimeSpan.FromMinutes(1)
 };
            timer.Start();
            timer.Tick += (a, b) => DrawCurrentTime();
            DrawCurrentTime();
        }

    }
```

EVENTPREVIEWBOX

```
public sealed partial class EventPreviewBox : UserControl
    {
        public EventPreviewBox()
        {
            this.InitializeComponent();
        }


        public void FillWith(Timetable table)
        {
```

```
            //assume there is a next lesson , so make it visible
            nextPanel.Visibility = Windows.UI.Xaml.Visibility.Visible;
            description.Text = table.Description;
            //placeholder name to check if it was overriden
            Course earliest = new Course {Name="null" };
            //pick biggest time for use in linear search
            Instance earliestInstance = new Instance { StartTime = Time.MaximumT
ime , EndTime = Time.MaximumTime };
            foreach (Course model in table.Courses)
            {
                //we will have a list of all courses
                EventControl control = new EventControl(model);
                control.Margin = new Thickness(5, 0, 5, 0);
                eventHolder.Children.Add(control);
                //iterate through all instances
                foreach (Instance instance in model.Instances)
                {
                    //check if it's the same day as today
                    if (instance.Day == (int)DateTime.Now.DayOfWeek-1)
                    {
                        //current time
                        Time now = new Time { Hours = DateTime.Now.Hour, Minutes
 = DateTime.Now.Minute };
                        //if it's in the future
                        if (now < instance.StartTime)
                        {
                            //if we passed the original set instance then discar
d it
                            if (earliestInstance.StartTime < now)
                            {
                                earliestInstance = instance;
                                earliest = model;
                            }
                            else
                            {
                                //check if the current instance is closer to the
 present than the second
```

```
                                if (instance.StartTime - now < earliestInstance.
StartTime - now)
                                {
                                    earliestInstance = instance;
                                    earliest = model;
                                }
                            }
                        }
                    }
                }
                //if it has been changed , fill with new info
                if (earliest.Name != "null")
                {
                    eventControl.FillWith(earliest);
                    eventControl.setTime(earliestInstance.StartTime, earliestInstanc
e.EndTime);
                }
                else
                {
                    //else hide , there is no next lesson today
                    nextPanel.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
                }
            }
        }
```

NEWTIMETABLEBOX.XAML.CS

```
public sealed partial class NewTimetableBox : UserControl
    {
        public NewTimetableBox()
        {
            this.InitializeComponent();
        }

        public delegate void CreateHandler();
        public event CreateHandler OnCreate;
```

```
        private void add_tapped(object sender, Windows.UI.Xaml.Input.TappedRoute
dEventArgs e)
        {
            //when the add button is tapped go to the fron view
            //and toggle the visibility to avoid unintended touches
            AnimationHelper.FadeTo(front, 0,0.5, true).Completed += (a, b) =>
                {
                    back.Visibility = Windows.UI.Xaml.Visibility.Visible;
                    AnimationHelper.FadeTo(back,1, 0.5, true).Completed += (c, d
) =>
                        {
                            front.Visibility = Windows.UI.Xaml.Visibility.Collap
sed;
                        };
                };
        }

        private void back_clicked(object sender, Windows.UI.Xaml.RoutedEventArgs
 e)
        {
            //do the opposite of add_clicked
            AnimationHelper.FadeTo(front, 0, 0.5, true).Completed += (a, b) =>
            {
                front.Visibility = Windows.UI.Xaml.Visibility.Visible;
                AnimationHelper.FadeTo(front, 1, 0.5, true).Completed += (c, d)
=>
                {
                    back.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
                };
            };
        }

        private async void create_clicked(object sender, Windows.UI.Xaml.RoutedE
ventArgs e)
        {

            //validate inputs
```

```
            ValidatorResponse TitleErrors = Validator.isStringValid(
                target: _name.Text, scope: "title", allowDigits: true, allowEmpt
y: false, max: Validator.TimetableTitleMax , fileName:true);

            ValidatorResponse DescriptionErrors = Validator.isStringValid(
                target: _description.Text, scope: "description", allowDigits: tr
ue, allowEmpty: true, max: Validator.TimetableDescriptionMax);

            //if there are errors for either field give an error.
            if (!TitleErrors.isValid || !DescriptionErrors.isValid)
            {
                //this will join the errors of both inputs
                string errorMessage = TitleErrors.Errors.Aggregate("Cannot creat
e timetable because of the following errors:\n", (current, s) => current + (s +
"\n"));
                errorMessage = DescriptionErrors.Errors.Aggregate(errorMessage,
(current, s) => current + (s + "\n"));

                var dialog = new MessageDialog(content: errorMessage
                                                        , title: "An error occ
ured");
                await dialog.ShowAsync();
            }
            else
            {
                //create a new timetable asynchrnously
                var tb = new Timetable { Name = _name.Text, Description = _descr
iption.Text };
                if (await CoreData.CreateTimetable(tb))
                {
                    // CoreData.Timetables.Add(tb);
                    OnCreate();
                    toFront.Begin();
                    _name.Text = "";
                    _description.Text = "";
                }
                else
                {
```

```
                    var diag = new MessageDialog("There was an error creating it
");
                    await diag.ShowAsync();
                }
            }
        }

        private async void import_clicked(object sender, Windows.UI.Xaml.RoutedE
ventArgs e)
        {
            if (await CoreData.ImportTimetable())
            {
                OnCreate();//after import request redraw of all timetables
            }
            else
            {
                var diag = new MessageDialog("The file might be corrupted or you
 might have another timetable with the same name.Please check there isn't anothe
r timetable with the same name", "There was a problem");
                await diag.ShowAsync();
            }
        }
    }
```

TIMETABLEBOX.XAML.CS

```
public sealed partial class TimetableBox : UserControl
    {
        Timetable self;
        public TimetableBox(Timetable table)
        {
            this.InitializeComponent();
            //set initial values
            _name.Text = table.Name;
            _description.Text = table.Description;
            _nameLabel.Text = table.Name;
            self=table;
```

```
            preview.FillWith(self);
        }

        public delegate void DeletedEventHandler();
        public event DeletedEventHandler OnDeleted;

        public delegate void OpenHandler(Timetable table);
        public event OpenHandler OnOpen;

        private void edit_clicked(object sender, Windows.UI.Xaml.RoutedEventArgs
 e)
        {
            //switch to back view
            AnimationHelper.FadeTo(front, 0, 0.5, true).Completed += (a, b) =>
            {
                back.Visibility = Windows.UI.Xaml.Visibility.Visible;
                AnimationHelper.FadeTo(back, 1, 0.5, true).Completed += (c, d) =
>
                {
                    front.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
                };
            };
        }

        private async void save_clicked(object sender, Windows.UI.Xaml.RoutedEve
ntArgs e)
        {
            //rename is true if the text is changed AND not empty
            bool rename = _name.Text != self.Name && _name.Text != "";
            self.Description = _description.Text;
            //validate inputs
            ValidatorResponse TitleErrors = Validator.isStringValid(
                target: _name.Text, scope: "title", allowDigits: true, allowEmpt
y: false, max: Validator.TimetableTitleMax, fileName: true);

            ValidatorResponse DescriptionErrors = Validator.isStringValid(
                target: _description.Text, scope: "description", allowDigits: tr
ue, allowEmpty: true, max: Validator.TimetableDescriptionMax);
```

```
            //if everything is valid
            if (TitleErrors.isValid && DescriptionErrors.isValid)
            {
                //edit the timetable
                if (await CoreData.EditTimetable(self, rename: rename, newName:
_name.Text))
                {
                    _nameLabel.Text = _name.Text;
                    ToFront();
                }
                preview.FillWith(self);
            }
            else
            {
                //join all errors toghether
                //this LINQ statement appends the last statement with the new er
ror and a \n newline character
                string errorMessage = TitleErrors.Errors.Aggregate("Cannot save
timetable because of the following errors:\n", (current, s) => current + (s + "\
n"));
                errorMessage = DescriptionErrors.Errors.Aggregate(errorMessage,
(current, s) => current + (s + "\n"));


                var dialog = new MessageDialog(content: errorMessage
                                                        , title: "An error occ
ured");
                await dialog.ShowAsync();
            }
        }


        private async void delete_clicked(object sender, Windows.UI.Xaml.RoutedE
ventArgs e)
        {
            //ask for confirmation
            var dialog = new MessageDialog("This action cannot be undone.", "Are
 you sure you want to delete " + self.Name);
            var delete = new UICommand("Delete", (com) => {
```

```
            });
            var cancel = new UICommand("Cancel");
            dialog.Commands.Add(delete);
            dialog.Commands.Add(cancel);
            var response =  await dialog.ShowAsync();
            //check response and perform appropiate action.
            if (response.Label == "Delete")
            {
                if (await CoreData.DeleteTimetable(self))
                {
                    OnDeleted();
                }
                else
                {
                    var diag = new MessageDialog("Try to restart the app.", "Thi
s timetable could not be deleted");
                    await diag.ShowAsync();

                }
            }

        }

        private void back_clicked(object sender, Windows.UI.Xaml.RoutedEventArgs
 e)
        {
            ToFront();

        }

        private void open_clicked(object sender, Windows.UI.Xaml.RoutedEventArgs
 e)
        {
            OnOpen(self);
        }
```

```
        private async void export_clicked(object sender, Windows.UI.Xaml.RoutedE
ventArgs e)
        {
            await CoreData.ExportTimetable(self);
        }

        private void ToFront()
        {
            AnimationHelper.FadeTo(back, 0, 0.5, true).Completed += (a, b) =>
            {
                front.Visibility = Windows.UI.Xaml.Visibility.Visible;
                AnimationHelper.FadeTo(front, 1, 0.5, true).Completed += (c, d)
=>
                {
                    back.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
                };
            };
        }
    }
```

COLORCHOOSER.XAML.CS

```
/// <summary>
    /// Used to choose the color of the class.
    /// </summary>
    public sealed partial class ColorChooser : UserControl
    {
        /// <summary>
        /// This event is triggered when the user presses a rectangle other than
 the selected one.
        /// </summary>
        /// <param name="newColor">The new color selected.</param>
        public delegate void ColorChangedHandler(Color newColor);
        public event ColorChangedHandler ColorChanged;
        public ColorChooser()
        {
            this.InitializeComponent();
```

```
            foreach (Rectangle r in root.Children)
            {
                AnimationHelper.AddPointerAnimation(r);
                r.Tapped += (e, ev) =>
                {
                    //when tapped move all elements to the starting line
                    //and then move e - the sender rectangle
                    foreach (Rectangle rr in root.Children)
                    {
                        AnimationHelper.SmoothMove(rr, 30, 0.5);
                    }
                    AnimationHelper.SmoothMove(e as FrameworkElement, 10, 0.5);
                    SelectedElement = e as Rectangle;
                    ColorChanged(SelectedColor);
                };
            }
        }
        /// <summary>
        /// Points to the rectangle currently selected by the user (or defaulted
by the constructor)
        /// </summary>
        public Rectangle SelectedElement;

        /// <summary>
        /// Property for setting and returning the current color
        /// </summary>
        public Color SelectedColor
        {
            get
            {
                try
                {
                    return (SelectedElement.Fill as SolidColorBrush).Color;
                }
                catch {return new Color();}
```

```
            }
            set
            {
                foreach (Rectangle r in root.Children)
                {
                    if ((r.Fill as SolidColorBrush).Color == value)
                    {
                        AnimationHelper.SmoothMove(r, 10, 0.5);
                    }
                    else
                    {
                        AnimationHelper.SmoothMove(r, 30, 0.5);
                    }
                }
            }
        }
    }
```

EVENTCONTROL.XAML.CS

```
public sealed partial class EventControl : UserControl
    {
        public Course self;
        public EventControl(Course model)
        {
            this.InitializeComponent();
            //set initial values
            SubjectName.Text = model.Name;
            TeacherName.Text = model.Teacher;
            ClassName.Text = model.Room;
            self = model;
            root.Fill = new SolidColorBrush(model.TileColor);
        }

        public EventControl() {
            this.InitializeComponent();
```

```
        }

        /// <summary>
        /// This method is used to change the target to
        /// something other than the value passed to the constructor
        /// </summary>
        /// <param name="model"></param>
        public void FillWith(Course model)
        {
            self = model;
            SubjectName.Text = model.Name;
            TeacherName.Text = model.Teacher;
            ClassName.Text = model.Room;
            root.Fill = new SolidColorBrush(model.TileColor);
        }

        //sets the time labels
        public void setTime(Time start , Time end)
        {
            _time.Text = start.ToString() + " to " + end.ToString();
        }


    }
```

EVENTPREVIEW.XAML.CS

```
public sealed partial class EventPreview : UserControl
    {
        public EventPreview(Course model)
        {
            this.InitializeComponent();
            //fill with given values
            _class.Text = model.Room;
            _teacher.Text = model.Teacher;
            _name.Text = model.Name;
```

```
            _notes.Text = model.Notes;
            rect.Fill = new SolidColorBrush(model.TileColor);
            if (model.Notes == "") _notes.Visibility = Windows.UI.Xaml.Visibilit
y.Collapsed;
        }
    }
```

INSTANCECONTROL.XAML.CS

```
public sealed partial class InstanceControl : UserControl
    {
        //if the time is invalid for some reason this will show an error icon
        public bool isValid
        {
            set { error_flag.Visibility = (value) ? Visibility.Collapsed : Visib
ility.Visible; }
            get { return (error_flag.Visibility == Windows.UI.Xaml.Visibility.Co
llapsed) ? true : false; }
        }

        Instance self;
        Timetable selfTB;
        public InstanceControl(Instance instance , Timetable timetable)
        {
            this.InitializeComponent();
            this.Loaded += (a, b) =>
                                {

                                    start1.Items.Clear();
                                    start2.Items.Clear();
                                    end1.Items.Clear();
                                    end2.Items.Clear();
                                    //add the hour values
                                    for (int x = 1; x < 24; x++)
                                    {
                                        start1.Items.Add(new ComboBoxItem {Conten
t = x.ToString()});
```

```
                                    end1.Items.Add(new ComboBoxItem {Content
= x.ToString()});
                                }
                                //add the minute values ( interval of 5)
                                for (int x = 0; x <= 55; x += 5)
                                {
                                    start2.Items.Add(new ComboBoxItem {Conten
t = (x == 0) ? x.ToString() + "0" : x.ToString()});
                                    end2.Items.Add(new ComboBoxItem {Content
= (x == 0) ? x.ToString() + "0" : x.ToString()});
                                }
                                start1.SelectedIndex = 7;
                                end1.SelectedIndex = 8;

                                start2.SelectedIndex = 0;
                                end2.SelectedIndex = 0;

                                day.SelectedIndex = self.Day;


                                //fill with given values
                                day.SelectedIndex = self.Day;
                                start1.SelectedIndex = self.StartTime.Hours -
 1;
                                start2.SelectedIndex = self.StartTime.Minutes
/15;
                                end1.SelectedIndex = self.EndTime.Hours - 1;
                                end2.SelectedIndex = self.EndTime.Minutes/15;

                                start1.SelectionChanged += changedEvent;
                                start2.SelectionChanged += changedEvent;
                                end1.SelectionChanged += changedEvent;
                                end2.SelectionChanged += changedEvent;
                                day.SelectionChanged += changedEvent;
                            };
            self = instance;
            selfTB = timetable;
        }
```

```
        public delegate void InstanceChangedHandler();
        public event InstanceChangedHandler InstanceChanged;
        async void changedEvent(object sender, SelectionChangedEventArgs e)
        {

            //when somthing has been changed , update the Instance with the new
values
            self.StartTime.Hours = int.Parse(((start1.SelectedItem as ComboBoxIt
em).Content as string));
            self.StartTime.Minutes = int.Parse(((start2.SelectedItem as ComboBox
Item).Content as string));

            self.EndTime.Hours = int.Parse(((end1.SelectedItem as ComboBoxItem).
Content as string));
            self.EndTime.Minutes = int.Parse(((end2.SelectedItem as ComboBoxItem
).Content as string));

            self.Day = day.SelectedIndex;
            try
            {
                InstanceChanged();
            }
            catch { }

            // if the start time is after the end time then show an error
            if (int.Parse((start1.SelectedValue as ComboBoxItem).Content as stri
ng) * 60 + int.Parse((start2.SelectedValue as ComboBoxItem).Content as string) >
= int.Parse((end1.SelectedValue as ComboBoxItem).Content as string) * 60 + int.P
arse((end2.SelectedValue as ComboBoxItem).Content as string))
            {
                MessageDialog error_dialog = new MessageDialog("The class start
time must be before the end time", "We have a problem , Watson");
                error_dialog.Commands.Add(new UICommand("Okay"));
                await error_dialog.ShowAsync();
                error_flag.Visibility = Windows.UI.Xaml.Visibility.Visible;
            }
            else
            {
```

```
                error_flag.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }

            //Check for overlaps
            TimeOverlapData result = CoreTime.IsOverlapping(self, selfTB);
            if (result.isOverlapping)
            {
                MessageDialog error_dialog = new MessageDialog("Classes cannot o
verlap.Please choose another time.", "This class time overlaps with your " + res
ult.OverlappingEventName + " class");
                error_dialog.Commands.Add(new UICommand("Okay"));
                await error_dialog.ShowAsync();
                isValid = false;
            }
            else
            {
                isValid = true;
            }
        }


        public delegate void DeletedHandler(InstanceControl control , Instance i
nstance);
        public event DeletedHandler OnDelete;
        private void delete_clicked(object sender, Windows.UI.Xaml.RoutedEventAr
gs e)
        {
            OnDelete(this,self);
        }
    }
```

MODULESEDITOR.XAML.CS

```
/// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame
.
    /// </summary>
    public sealed partial class ModulesEditor : Page
```

```
    {
        public ModulesEditor()
        {
            this.InitializeComponent();
        }

        //self will hold the current Course
        Course self;
        protected async override void OnNavigatedTo(NavigationEventArgs e)
        {
            self = (e.Parameter as object[])[1] as Course;

            title.Foreground = new SolidColorBrush(self.TileColor);
            back.Foreground = new SolidColorBrush(self.TileColor);
            add.Foreground = new SolidColorBrush(self.TileColor);
            edit.Foreground = new SolidColorBrush(self.TileColor);

            title.Text = self.Name + " modules";

            back.Tapped += async(a, b) =>{
                await CoreData.SaveTimetable((e.Parameter as object[])[0] as Tim
etable);
                Frame.Navigate(typeof(MainPage), (e.Parameter as object[])[0]);
            };


            foreach (Module mod in self.Modules)
            {
                ModuleBox box = new ModuleBox(mod, self.TileColor, holder.Childr
en, self);
                holder.Children.Add(box);
            }

            add.Tapped += (q, b) =>
            {
                Module m = new Module { Name = "New" };
                self.Modules.Add(m);
```

```
                ModuleBox box = new ModuleBox(m,self.TileColor,holder.Children ,
 self);

                holder.Children.Add(box);
            };

            edit.Tapped += (a,b) =>
            {
                if ((edit.Foreground as SolidColorBrush).Color == Windows.UI.Col
ors.Black)
                {
                    foreach (ModuleBox box in holder.Children) box.setEditable(f
alse);
                    edit.Foreground = new SolidColorBrush(self.TileColor);
                }
                else
                {
                    foreach (ModuleBox box in holder.Children) box.setEditable(t
rue);
                    edit.Foreground = new SolidColorBrush(Windows.UI.Colors.Blac
k);
                }
            };
        }
    }
```

NOTEBOX.XAML.CS

```
public sealed partial class NoteBox : EditableItem
    {
        Note self;
        Color color;
        public NoteBox(Note note , Color col)
        {

            this.InitializeComponent();
            //set initial values
            color = col;
```

```
            header.Background = new SolidColorBrush(col);
            add.Foreground = new SolidColorBrush(Colors.White);
            edit.Foreground = new SolidColorBrush(Colors.White);
            self = note;

            //set the items shown during edit mode
            SetEditableItems(TitleEdit,delete);
            SetStaticItems(Title);

            this.VerticalAlignment = Windows.UI.Xaml.VerticalAlignment.Stretch;
           //toggle edit mode
            edit.Tapped += (a, b) =>
            {
                if ((edit.Foreground as SolidColorBrush).Color == Windows.UI.Col
ors.Black)
                {
                    foreach (EditableItem box in holder.Children) box.setEditabl
e(false);
                    edit.Foreground = new SolidColorBrush(Colors.White);
                }
                else
                {
                    foreach (EditableItem box in holder.Children) box.setEditabl
e(true);
                    edit.Foreground = new SolidColorBrush(Windows.UI.Colors.Blac
k);
                }
                setEditable((edit.Foreground as SolidColorBrush).Color == Window
s.UI.Colors.Black);
            };

            Title.Text = self.Name;
            TitleEdit.Text = self.Name;

            TitleEdit.TextChanged += (a, b) =>
                                     {
                                         //validate string and fix if possib
le
```

```
                                            Validator.isStringValid(
                                                                    target: Tit
leEdit.Text, scope: "title", allowDigits: true, allowEmpty: true, max: Validator
.NoteNameMax,
                                                                    autoFix: tr
ue, fixTarget: TitleEdit, autoNotify: true);
                                                Title.Text = TitleEdit.Text;
                                                self.Name = TitleEdit.Text;
                                            };

            //create a new paragraph and add it to both UI and model
            add.Tapped += (a, b) =>
            {
                NoteFragment frag = new NoteFragment { Content = "Turn on editin
g to add text" };
                self.Fragments.Add(frag);
                refresh();
            };

            this.Loaded += (c, d) =>
            {
                refresh();
                //holder.SizeChanged += (a, b) => this.Height = holder.ActualHei
ght + 68;
            };



        }

        void refresh()
        {
            holder.Children.Clear();
            bool shade = true;
            foreach (NoteFragment fragment in self.Fragments)
            {
                NoteFragmentControl cont = new NoteFragmentControl(fragment, hol
der.Children, self , color,shade);
                holder.Children.Add(cont);
```

```
                shade = !shade;
            }
            //this.Height = holder.ActualHeight + 68;
        }
    }
```

NOTEFRAGMENTCONTROL.XAML.CS

```
public sealed partial class NoteFragmentControl : EditableItem
    {
        public NoteFragment self;
        public NoteFragmentControl(NoteFragment fragment, UIElementCollection co
ntainer , Note note , Color col , bool shouldShade)
        {
            this.InitializeComponent();
            self = fragment;
            content.Text = fragment.Content;
            editBox.Text = content.Text;

            SetEditableItems(editBox,deleteButton,edit);
            SetStaticItems(content);

            back.Background = new SolidColorBrush(col);
            //the paragraphs occur in an alternating pattern of brighter and dar
ker colors (like a table)
            shade.Visibility = (shouldShade) ? Visibility.Visible : Visibility.C
ollapsed;

            deleteButton.Tapped += (e, ev) =>
            {
                container.Remove(this);
                note.Fragments.Remove(self);
            };
            //validate
            editBox.TextChanged += (e, ev) =>
            {
                Validator.isStringValid(
```

```
                    target: editBox.Text, scope: "content", allowDigits: true, a
llowEmpty: true, max: Validator.NoteContentMax,
                    autoFix: true, fixTarget: editBox, autoNotify: true);
                content.Text = editBox.Text;
                self.Content = editBox.Text;
            };

            //expand size when more text is added
            editBox.SizeChanged += (_e, _ev) =>
            {
                this.Height = editBox.ActualHeight + 10;
            };

            this.Loaded+=(_e,_ev)=>this.Width = (this.Parent as FrameworkElement
).ActualWidth;

            Update();

            edit.Tapped += (a, b) =>
            {
                //when edit is tapped , show the options in a pop up
                NotesToolbox ns = new NotesToolbox(self);
                Flyout fl = new Flyout { Placement = PlacementMode.Top , Placeme
ntTarget = edit };
                fl.Content = ns;
                fl.IsOpen = true;
                ns.OnChange += () => Update();
            };


        }

        //redraw all formatting after a change
        public void Update()
        {
            content.FontSize = self.FontSize;
```

```
            content.FontWeight = (self.Bold) ? FontWeights.Bold : FontWeights.No
rmal;
            content.FontStyle = (self.Italic) ? FontStyle.Italic : FontStyle.Nor
mal;
            content.TextAlignment = (self.Centered) ? TextAlignment.Center : Tex
tAlignment.Left;


            editBox.FontSize = self.FontSize;
            editBox.FontWeight = (self.Bold) ? FontWeights.Bold : FontWeights.No
rmal;
            editBox.FontStyle = (self.Italic) ? FontStyle.Italic : FontStyle.Nor
mal;
            editBox.TextAlignment = (self.Centered) ? TextAlignment.Center : Tex
tAlignment.Left;

        }


    }
```

NOTESEDITOR.XAML.CS

```
/// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame
.
    /// </summary>
    public sealed partial class NotesEditor : Page
    {
        public NotesEditor()
        {
            this.InitializeComponent();
        }

        Course self;
        protected async override void OnNavigatedTo(NavigationEventArgs e)
        {
            self = (e.Parameter as object[])[1] as Course;
```

```
            title.Foreground = new SolidColorBrush(self.TileColor);
            pin.Foreground = new SolidColorBrush(self.TileColor);
            back.Foreground = new SolidColorBrush(self.TileColor);
            add.Foreground = new SolidColorBrush(self.TileColor);

            title.Text = self.Name + " notes";


            bool isPinned = false;
            //iterate through all tiles , if found then make the icon to unpin i
t
            foreach (SecondaryTile tl in await SecondaryTile.FindAllForPackageAs
ync())
            {
                if (tl.TileId == self.Name.Replace(" ", "") + "notes")
                {
                    pin.Icon = "";
                    isPinned = true;
                }
            }

            pin.Tapped += async (a, b) =>
            {
                //get image and ID
                Uri logo = new Uri("ms-appx:///Assets/TileLogo.png");
                SecondaryTile tile = new SecondaryTile();
                tile.BackgroundColor = self.TileColor;
                tile.TileId = self.Name.Replace(" ", "") + "notes";
                tile.Logo = logo;
                tile.ForegroundText = ForegroundText.Light;
                tile.DisplayName = "NotesEditor for " + self.Name;
                tile.ShortName = self.Name + " notes";
                tile.TileOptions = TileOptions.ShowNameOnLogo;
                tile.Arguments = "notes," + ((e.Parameter as object[])[0] as Tim
etable).Name + "," + self.Name;
                //create or remove tile based on whether it already exists.
                if (!isPinned)
```

```
            {
                isPinned = await tile.RequestCreateAsync();
                pin.Icon = "";
            }
            else
            {
                isPinned = !await tile.RequestDeleteAsync();
                pin.Icon = "";
            }
        };


        back.Tapped += async (a, b) =>
        {
            await CoreData.SaveTimetable((e.Parameter as object[])[0] as Tim
etable);
            Frame.Navigate(typeof(MainPage), (e.Parameter as object[])[0]);
        };

        moduleHolder.fill(self, typeof(NotesEditor));
        moduleHolder.OnChosen += (module) => RefreshNotes();
        RefreshNotes();



        add.Tapped += (q, b) =>
        {
            var note = new Note { Name = "Click edit to enter a name" };
            moduleHolder.Chosen.Notes.Add(note);
            RefreshNotes();
        };

        holder.SelectionChanged += (a, b) =>
                                   {
                                       //make the current edited note ti
tle bold
```

```
                                                (TitleHolder.Children[holder.Sele
ctedIndex] as TextBlock).FontWeight = FontWeights.Bold;
                                                foreach (TextBlock c in TitleHold
er.Children)
                                                {
                                                    //make all the other gray and
 normal sized
                                                    c.Foreground = Foreground = n
ew SolidColorBrush(Colors.Gray);
                                                    c.FontWeight = FontWeights.No
rmal;
                                                }
                                                //color it with the accent color
                                                (TitleHolder.Children[holder.Sele
ctedIndex] as TextBlock).Foreground = new SolidColorBrush(self.TileColor);

                                            };
        }

        void RefreshNotes()
        {
            holder.Items.Clear();
            TitleHolder.Children.Clear();
            for (int index = 0; index < moduleHolder.Chosen.Notes.Count; index++
)
            {
                Note note = moduleHolder.Chosen.Notes[index];
                var nb = new NoteBox(note, self.TileColor);
                nb.VerticalAlignment = Windows.UI.Xaml.VerticalAlignment.Top;
                //add the note to the FlipView , which lets the user swipe throu
gh them
                holder.Items.Add(new FlipViewItem {VerticalAlignment = Windows.U
I.Xaml.VerticalAlignment.Top, Content = nb});

                //create title button for choosing the note
                var item = new TextBlock
                                {
                                    Text = note.Name,
                                    FontSize = 25,
```

```
                                    Foreground = new SolidColorBrush(Colors.Gray)
,
                                    Tag = index,
                                    Margin = new Thickness(0,10,0,10)
                                };

                AnimationHelper.AddPointerAnimation(item);
                //when tapped sync to the FlipView
                item.Tapped += (a, b) =>
                                    {
                                        holder.SelectedIndex = (int) ((a as TextB
lock).Tag);
                                        (a as TextBlock).FontWeight = FontWeights
.Bold;
                                        foreach (TextBlock c in TitleHolder.Child
ren)
                                        {
                                            c.Foreground = Foreground = new Solid
ColorBrush(Colors.Gray);
                                            c.FontWeight = FontWeights.Normal;
                                        }
                                        (a as TextBlock).Foreground = new SolidCo
lorBrush(self.TileColor);

                                    };

                TitleHolder.Children.Add(item);
                (TitleHolder.Children[holder.SelectedIndex] as TextBlock).FontWe
ight = FontWeights.Bold;
                (TitleHolder.Children[holder.SelectedIndex] as TextBlock).Foregr
ound = new SolidColorBrush(self.TileColor);
            }
        }
    }
```

NOTETOOLBOX.XAML.CS

```
public sealed partial class NotesToolbox : UserControl
    {
```

```
        public delegate void OnChangeHandler();
        public event OnChangeHandler OnChange;

        public NotesToolbox()
        {

        }

        public NotesToolbox(NoteFragment fragment)
        {
            this.InitializeComponent();
            //set initial values
            sizeSlider.Value = fragment.FontSize;
            italic.IsChecked = fragment.Italic;
            bold.IsChecked = fragment.Bold;
            center.IsChecked = fragment.Centered;

            //these lambdas change the instance and trigger the change event.
            sizeSlider.ValueChanged += (a, b) =>
                { fragment.FontSize = sizeSlider.Value; OnChange(); };
            italic.Tapped += (a, b) =>
                {
                    fragment.Italic = italic.IsChecked.Value;
                    OnChange();
                };
            bold.Tapped += (a, b) => {
                fragment.Bold = bold.IsChecked.Value;
                OnChange();
            } ;
            center.Tapped += (a, b) => {
                fragment.Centered = center.IsChecked.Value;
                OnChange();
            } ;
        }
    }
```

MODULELIST.XAML.CS

```
public sealed partial class ModuleList : UserControl
    {
        public ModuleList()
        {
            this.InitializeComponent();

        }

        public delegate void ChosenEventHandler(object target);
        public event ChosenEventHandler OnChosen;

        public Module Chosen;

        public void fill(Course model , Type context)
        {
            foreach (Module mod in model.Modules)
            {
                var c = new ModuleListItem(mod, context,model.TileColor);
                holder.Children.Add(c);
                c.Tapped += (a, b) =>
                {
                    //if an item is tapped , deselect everything and then select
 that object
                    Chosen = (a as ModuleListItem).self;
                    foreach (ModuleListItem item in holder.Children) item.setSel
ected(false);
                    OnChosen((a as ModuleListItem).self);
                    (a as ModuleListItem).setSelected(true);

                };
            }
            //at the start select the first item
            (holder.Children[0] as ModuleListItem).setSelected(true);
            Chosen = (holder.Children[0] as ModuleListItem).self;
```

```
        }


    }
```

MODULELISTITEM.XAML.CS

```
public sealed partial class ModuleListItem : UserControl
    {
        public Module self;
        Color col;
        public ModuleListItem(Module module , Type context , Color c)
        {
            this.InitializeComponent();
            col = c;
            self = module;
            name.Text = module.Name;
            this.HorizontalAlignment = Windows.UI.Xaml.HorizontalAlignment.Cente
r;
            this.Loaded += (a, b) => this.Width = name.ActualWidth;
            //each editor wants to know the count of a different info type
            //therefore a Type is passed
            if (context == typeof(TopicsEditor))
            {
                count.Text = module.StructureBlocks.Count.ToString();
            }
            if (context == typeof(NotesEditor))
            {
                count.Text = module.Notes.Count.ToString();
            }
            if (context == typeof(DefinitionsEditor))
            {
                count.Text = module.Definitions.Count.ToString();
            }
            if (context == typeof(HomeworkEditor))
            {
```

```
                count.Text = module.Homeworks.Count.ToString();
            }
            setSelected(false);
        }

        public void setSelected(bool selected)
        {
            name.Foreground = new SolidColorBrush((selected) ? col : Colors.DimG
ray);
        }
    }
```

TOPICSEDITOR.XAML.CS

```
public sealed partial class TopicsEditor : Page
    {
        public TopicsEditor()
        {
            this.InitializeComponent();
        }

        Course self;
        protected async override void OnNavigatedTo(NavigationEventArgs e)
        {
            self = (e.Parameter as object[])[1] as Course;

            title.Foreground = new SolidColorBrush(self.TileColor);
            back.Foreground = new SolidColorBrush(self.TileColor);
            add.Foreground = new SolidColorBrush(self.TileColor);
            edit.Foreground = new SolidColorBrush(self.TileColor);
            pin.Foreground = new SolidColorBrush(self.TileColor);

            title.Text = self.Name + " definitions";

            //this code will pin or unpin a shortcut to the editor
            bool isPinned = false;
```

```
            foreach (SecondaryTile tl in await SecondaryTile.FindAllForPackageAs
ync())
            {
                if (tl.TileId == self.Name.Replace(" ", "") + "topics")
                {
                    pin.Icon = "";
                    isPinned = true;
                }
            }

            pin.Tapped += async (a, b) =>
            {
                Uri logo = new Uri("ms-appx:///Assets/TileLogo.png");
                SecondaryTile tile = new SecondaryTile();
                tile.BackgroundColor = self.TileColor;
                tile.TileId = self.Name.Replace(" ", "") + "topics";
                tile.Logo = logo;
                tile.ForegroundText = ForegroundText.Light;
                tile.DisplayName = "Topics for " + self.Name;
                tile.ShortName = self.Name + " topics";
                tile.TileOptions = TileOptions.ShowNameOnLogo;
                tile.Arguments = "topics," + ((e.Parameter as object[])[0] as Ti
metable).Name + "," + self.Name;
                if (!isPinned)
                {
                    isPinned = await tile.RequestCreateAsync();
                    pin.Icon = "";
                }
                else
                {
                    isPinned = !await tile.RequestDeleteAsync();
                    pin.Icon = "";
                }
            };
            back.Tapped += async (a, b) =>
            {
```

```
                await CoreData.SaveTimetable((e.Parameter as object[])[0] as Tim
etable);
                Frame.Navigate(typeof(MainPage), (e.Parameter as object[])[0]);
            };

            moduleHolder.fill(self,typeof(TopicsEditor));
            //when a new module is chosen , redraw everything
            moduleHolder.OnChosen += (module) =>
            {
                RefreshTopics();
            };
            RefreshTopics();



            add.Tapped += (q, b) =>
            {
                StructureBlock bl = new StructureBlock();
                moduleHolder.Chosen.StructureBlocks.Add(bl);
                RefreshTopics();
            };

            //toggle edit
            edit.Tapped += (a, b) =>
            {
                if ((edit.Foreground as SolidColorBrush).Color == Windows.UI.Col
ors.Black)
                {
                    foreach (EditableItem box in holder.Children) box.setEditabl
e(false);
                    edit.Foreground = new SolidColorBrush(self.TileColor);
                }
                else
                {
                    foreach (EditableItem box in holder.Children) box.setEditabl
e(true);
                    edit.Foreground = new SolidColorBrush(Windows.UI.Colors.Blac
k);
```

```
                }
            };
        }

        void RefreshTopics()
        {
            if (moduleHolder.Chosen != null)
            {
                holder.Children.Clear();
                foreach (StructureBlock def in moduleHolder.Chosen.StructureBloc
ks)
                {
                    StructureBlockControl dc = new StructureBlockControl(def, se
lf.TileColor, holder.Children, moduleHolder.Chosen);
                    holder.Children.Add(dc);
                }
                if (holder.Children.Count == 0) holder.Children.Add(new Image {
HorizontalAlignment = Windows.UI.Xaml.HorizontalAlignment.Center, VerticalAlignm
ent = Windows.UI.Xaml.VerticalAlignment.Top, Width = 600, Height = 600, Source =
 new Windows.UI.Xaml.Media.Imaging.BitmapImage(new Uri("ms-
appx:///Assets/empty.png")) });

            }
        }
    }
```

TOPICCONTROL.XAML.CS

```
public sealed partial class TopicControl : EditableItem
    {
        public delegate void ChangedEventHandler();
        public event ChangedEventHandler OnChange;

        public delegate void DeletedEventHandler(TopicControl topicC);
        public event DeletedEventHandler OnDelete;

        public Topic self;
        public TopicControl(Topic topic)
```

```
        {
            this.InitializeComponent();
            //set edit mode items
            SetEditableItems(_titleEdit , deleteButton);
            SetStaticItems(_title , _revised , _studied);
            //set initial values
            self = topic;
            _title.Text = topic.Name;
            _titleEdit.Text = topic.Name;
            _revised.IsChecked = topic.Revised;
            _studied.IsChecked = topic.Studied;

            //when the checkboxes are tapped , update the Topic
            _revised.Tapped += (e, ev) =>
            {
                self.Revised = _revised.IsChecked.Value;
            };
            _studied.Tapped += (e, ev) =>
            {
                self.Studied = _studied.IsChecked.Value;
            };

            //validate as the text changes.
            _titleEdit.TextChanged += (e, ev) =>
            {
                Validator.isStringValid(
                    target: _titleEdit.Text, scope: "title", allowDigits: false,
allowEmpty: true, max: Validator.TopicTitleMax,
                    autoFix: true, fixTarget: _titleEdit, autoNotify: true);
                _title.Text = _titleEdit.Text;
                self.Name = _titleEdit.Text;
                //OnChange();
            };

            deleteButton.Click += (e, ev) => OnDelete(this);
        }
```

```
    }
```

STRUCTUREBLOCKBOX.XAML.CS

```
public sealed partial class StructureBlockControl : EditableItem
    {
        public delegate void ChangedEventHandler();
        public event ChangedEventHandler OnChange;

        public delegate void DeletedEventHandler(StructureBlockControl structure
);
        public event DeletedEventHandler OnDelete;

        public StructureBlock self;
        public StructureBlockControl(StructureBlock block,Windows.UI.Color c, UI
ElementCollection container = null, Module module = null)
        {
            this.InitializeComponent();
            //set initial values
            root.Background = new SolidColorBrush(c);
            SetEditableItems(nameEdit);
            SetStaticItems(name);
            self = block;
            name.Text = self.Name;
            nameEdit.Text = self.Name;
            //add topics
            foreach (Topic topic in block.Topics)
            {
                TopicControl cont = new TopicControl(topic);
                topicsHolder.Children.Add(cont);
                cont.OnChange += () => OnChange();
                cont.OnDelete += (tpC) =>
                {
                    self.Topics.Remove(tpC.self);
                    topicsHolder.Children.Remove(tpC);
                };
            }
```

```
            //create a new topic and add it
            addButton.Tapped += (e, ev) =>
            {
                Topic topic = new Topic();
                TopicControl cont = new TopicControl(topic);
                if (edit) cont.setEditable(true);
                topicsHolder.Children.Add(cont);
                self.Topics.Add(topic);

                cont.OnDelete += (tpC) =>
                {
                    self.Topics.Remove(tpC.self);
                    topicsHolder.Children.Remove(tpC);
                };
            };
            deleteButton.Tapped += (e, ev) =>
            {
                container.Remove(this);
                module.StructureBlocks.Remove(self);
            };
            nameEdit.TextChanged += (e, ev) =>
            {
                Validator.isStringValid(
                    target: nameEdit.Text, scope: "name", allowDigits: false, al
lowEmpty: true, max: Validator.StructureBlockNameMax,
                    autoFix: true, fixTarget: nameEdit, autoNotify: true);
                name.Text = nameEdit.Text;
                self.Name = nameEdit.Text;
            };
        }

        //overriden because the behavious is slightly different
        public override void setEditable(bool isEditable)
        {
            //first set the editing mode to all controls
            foreach (TopicControl cont in topicsHolder.Children)
```

```
            {
                cont.edit = edit;
            }
            base.setEditable(isEditable);//continue to toggle own edit mode.
        }
    }
```

ACADEMYINFOCONTROL.XAML.CS

```
public sealed partial class AcademyInfoControl : UserControl
    {
        public AcademyInfoControl()
        {
            this.InitializeComponent();
        }

        public void FillWith(AcademyBundle bundle , MainPage page)
        {
            txtName.Text = bundle.Name;
            txtLocation.Text = bundle.Location;
            txtEvents.Text = bundle.Events;
            txtWebsite.Text = bundle.Website;

            //validates as the user types
            txtName.TextChanged += (a, b) =>
                                    {
                                        Validator.isStringValid(
                                                                target: txtNa
me.Text, scope: "name", allowDigits: true, allowEmpty: true, max: Validator.Acad
emyNameMax,
                                                                autoFix: true
, fixTarget: txtName, autoNotify: true);
                                        bundle.Name = txtName.Text;
                                        page.updateBundle(bundle);
                                    };
            //validate before putting it into the class instance
            txtLocation.TextChanged += (a, b) =>
```

```
            {
                Validator.isStringValid(
                    target: txtLocation.Text, scope: "location", allowDigits: tr
ue, allowEmpty: true, max: Validator.AcademyLocationMax,
                    autoFix: true, fixTarget: txtLocation, autoNotify: true);
                bundle.Location = txtLocation.Text;
                page.updateBundle(bundle);
            };

            txtWebsite.TextChanged += (a, b) =>
            {
                Validator.isStringValid(
                    target: txtWebsite.Text, scope: "website", allowDigits: true
, allowEmpty: true, max: Validator.AcademyWebsiteMax,
                    autoFix: true, fixTarget: txtWebsite, autoNotify: true);
                bundle.Website = txtWebsite.Text;
                page.updateBundle(bundle);
            };

            txtEvents.TextChanged += (a, b) =>
            {
                Validator.isStringValid(
                    target: txtEvents.Text, scope: "events", allowDigits: true,
allowEmpty: true, max: Validator.AcademyEventsMax,
                    autoFix: true, fixTarget: txtEvents, autoNotify: true);
                bundle.Events = txtEvents.Text;
                page.updateBundle(bundle);
            };

            ColorChooser.SelectedColor = bundle.Accent;
            ColorChooser.ColorChanged += (c) =>
                                        {
                                            bundle.Accent = c;
                                            page.updateBundle(bundle);
                                        };
        }
    }
```

UNIVERSALBUTTON.XAML.CS

```
public sealed partial class UniversalButton : UserControl
    {
        public UniversalButton()
        {
            this.InitializeComponent();
            //add animation to the Grid
            AnimationHelper.AddPointerAnimation(LayoutRoot);
        }

        //the text symbol that will be drawn inside the button
        public string Icon
        {
            get {return root.Content as string; }
            set { root.Content=value;}
        }


    }
```

APP.CS | ONLAUNCHED | CUSTOM PART ONLY

```
            //if this was launched without parameters (i,e from a tile)
            if (string.IsNullOrEmpty(args.Arguments))
            {
                //check if this is the first time the app is run
                bool first = false;
                if (!ApplicationData.Current.LocalSettings.Values.ContainsKey("f
irst"))
                {
                    ApplicationData.Current.LocalSettings.Values.Add(new KeyValu
ePair<string, object>("first", true));
                    //this key in the settings is created on the first run so it
 does not exist before that
```

```
                    first = true;
                }
                else {}
                //go to the Intro page if it is the first time
                Type tp = (first) ? typeof(Intro) : typeof(StartPage);
                if (!rootFrame.Navigate(tp))
                {
                    throw new Exception("Failed to create initial page");
                }
            }
            else{
                //if this is launched from a tile , load all timetables
                if (await CoreData.Initialize())
                {
                    //iterate through the folder and get all timetables
                    if (await CoreData.GetTimetables())
                    {

                    }
                }
                else
                {
                    //if something happened then suggest a fix for the user
                    MessageDialog dialog = new MessageDialog("There was a proble
m initializing the data storage.\nTry closing and reopening the app.", "Oops");
                    await dialog.ShowAsync();
                }
                //if there are aguments go to the appropiate page
                if (args.Arguments.StartsWith("definitions"))
                {
                    string targetTB = args.Arguments.Split(',')[1];//name of tar
get timetable
                    string target = args.Arguments.Split(',')[2];//name of targe
t course
                    //do a linear search
                    foreach (Timetable tb in CoreData.Timetables)
                    {
```

```
                        if (tb.Name == targetTB)
                        {
                            foreach (Course model in tb.Courses)
                            {
                                if(model.Name == target) rootFrame.Navigate(type
of(DefinitionsEditor), new object[2]{tb,model});
                                //open the editor
                            }
                        }
                    }
                }
                //the following code works the same way
                if (args.Arguments.StartsWith("homework"))
                {
                    string targetTB = args.Arguments.Split(',')[1];
                    string target = args.Arguments.Split(',')[2];
                    foreach (Timetable tb in CoreData.Timetables)
                    {
                        if (tb.Name == targetTB)
                        {
                            foreach (Course model in tb.Courses)
                            {
                                if (model.Name == target) rootFrame.Navigate(typ
eof(HomeworkEditor), new object[2] { tb, model });
                            }
                        }
                    }
                }

                if (args.Arguments.StartsWith("topics"))
                {
                    string targetTB = args.Arguments.Split(',')[1];
                    string target = args.Arguments.Split(',')[2];
                    foreach (Timetable tb in CoreData.Timetables)
                    {
                        if (tb.Name == targetTB)
                        {
```

```
                            foreach (Course model in tb.Courses)
                            {
                                if (model.Name == target) rootFrame.Navigate(typ
eof(TopicsEditor), new object[2] { tb, model });
                            }
                        }
                    }
                }

                if (args.Arguments.StartsWith("notes"))
                {
                    string targetTB = args.Arguments.Split(',')[1];
                    string target = args.Arguments.Split(',')[2];
                    foreach (Timetable tb in CoreData.Timetables)
                    {
                        if (tb.Name == targetTB)
                        {
                            foreach (Course model in tb.Courses)
                            {
                                if (model.Name == target) rootFrame.Navigate(typ
eof(NotesEditor), new object[2] { tb, model });
                            }
                        }
                    }
                }
            }
```

MAINPAGE.XAML.CS

```
public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            this.Loaded += MainPage_Loaded;

        }
```

```
public async void Save()
{
    await CoreData.SaveTimetable(self);
}

Timetable self;
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    //a Timetable must be passed in order to reach this page
    if (e.Parameter.GetType() == typeof(Timetable))
    {
        //self will contain a reference to the current Timetable
        self = e.Parameter as Timetable;
        //fill the Timetable preview with the Timetable
        table.FillWith(self);


    }
    else
    {

    }
}



void MainPage_Loaded(object sender, RoutedEventArgs e)
{

    //clicking on the eventsButton will go to the class editor
    eventsButton.Click += async (es, ev) =>
                                {
```

```
                                        Frame.Navigate(typeof (ClassesEditor),
 self);

                                    };



            //iterate through each class/course
            foreach (Course model in self.Courses)
            {
                //for each course we will add each type of preview item
                //the preview items will be added to their respective containers
                //the OnOpen event will go to the appropiate editor
                //the Navigation parameters are the current Timetable and the cu
rrent Course , passed in an array

                EventPreview pr = new EventPreview(model);
                classes_holder.Children.Add(pr);

                var c = new StructurePreview(model);
                topicsHolder.Children.Add(c);
                c.OnOpen += (mod) => Frame.Navigate(typeof(TopicsEditor), new ob
ject[] { self, mod });



                var p = new ModulePreview(model);
                moduleHolder.Children.Add(p);
                p.OnOpen += (m) => Frame.Navigate(typeof(ModulesEditor), new obj
ect[] { self, m });



                DefinitionPreview defp = new DefinitionPreview(model);
                definitionsHolder.Children.Add(defp);
                defp.OnOpen += (m) => Frame.Navigate(typeof(DefinitionsEditor),
new object []{self,m });

                HomeworkPreview pw = new HomeworkPreview(model);
                homeworkHolder.Children.Add(pw);
```

```
                pw.OnOpen += (m) => Frame.Navigate(typeof(HomeworkEditor), new o
bject[] { self, m });


                NotePreview np = new NotePreview(model);
                notesHolder.Children.Add(np);
                np.OnOpen += (m) => Frame.Navigate(typeof(NotesEditor), new obje
ct[] { self, m });
            }


            Title.Text = self.Name;


          //if the app is resized , it enters Snap Mode , which is a mode where
 it is snapped to ~20% of the screen with , so we will resize everything to fit
            Window.Current.SizeChanged += (a, b) =>
            {
                scrollViewer.VerticalScrollBarVisibility  = ((Windows.UI.ViewMan
agement.ApplicationView.Value == ApplicationViewState.Snapped)) ? ScrollBarVisib
ility.Visible:ScrollBarVisibility.Disabled;


                scrollViewer.VerticalScrollMode = ((Windows.UI.ViewManagement.Ap
plicationView.Value == ApplicationViewState.Snapped)) ? ScrollMode.Enabled : Scr
ollMode.Disabled;
                upperHolder.Orientation = (Windows.UI.ViewManagement.Application
View.Value != ApplicationViewState.Snapped) ? Orientation.Horizontal : Orientati
on.Vertical;
                upperHolder.Visibility =(Windows.UI.ViewManagement.ApplicationVi
ew.Value == ApplicationViewState.Snapped)? Visibility.Collapsed:Visibility.Visib
le;
                snappedView.Visibility = (Windows.UI.ViewManagement.ApplicationV
iew.Value == ApplicationViewState.Snapped) ? Visibility.Visible : Visibility.Col
lapsed;



                //We will take each collumn and add it to a FlipView (for easy s
wiping since dragging the page is inefficient with a small width)
                for (int x = 0; x < 7; x++)
                {


                    if (Windows.UI.ViewManagement.ApplicationView.Value != Appli
cationViewState.Snapped)
                    {
```

```
                        var tb = upperHolder.Children[0] as UIElement;
                        upperHolder.Children.Remove(tb);
                        UIElement elem = (snappedHolder.Items[x] as FlipViewItem
).Content as UIElement;
                        (snappedHolder.Items[x] as FlipViewItem).Content = null;
                        upperHolder.Children.Add(elem);
                        upperHolder.Children.Add(tb);
                    }
                    else
                    {
                        UIElement elem = upperHolder.Children[0];
                        upperHolder.Children.Remove(elem);
                        (snappedHolder.Items[x] as FlipViewItem).Content = elem;
                    }
                }

                foreach (UIElement element in classes_holder.Children)
                {
                    if(element.RenderTransform.GetType() != typeof(CompositeTran
sform))
                        element.RenderTransform = new CompositeTransform{ScaleX
=1 , ScaleY=1};
                    (element.RenderTransform as CompositeTransform).ScaleX = (Wi
ndows.UI.ViewManagement.ApplicationView.Value != ApplicationViewState.Snapped) ?
 1 : 0.8;
                    (element.RenderTransform as CompositeTransform).ScaleY = (Wi
ndows.UI.ViewManagement.ApplicationView.Value != ApplicationViewState.Snapped) ?
 1 : 0.8;
                }

                foreach (UIElement element in topicsHolder.Children)
                {
                    if (element.RenderTransform.GetType() != typeof(CompositeTra
nsform))
                        element.RenderTransform = new CompositeTransform { Scale
X = 1, ScaleY = 1 };
                    (element.RenderTransform as CompositeTransform).ScaleX = (Wi
ndows.UI.ViewManagement.ApplicationView.Value != ApplicationViewState.Snapped) ?
 1 : 0.8;
```

```
                    (element.RenderTransform as CompositeTransform).ScaleY = (Wi
ndows.UI.ViewManagement.ApplicationView.Value != ApplicationViewState.Snapped) ?
 1 : 0.8;
                }
                  };

            //set intial values for the academy info
            AcademyInfo.FillWith(self.AcademyInfo , this);
            //start inline academy information editor (and close)
            AcademyInfoButton.Tapped += (a, b) =>
                                        {
                                            if (AcademyInfo.Visibility == Vi
sibility.Collapsed)
                                            {
                                                AcademyInfo.Opacity = 0;
                                                AcademyInfo.Visibility = Vis
ibility.Visible;
                                                AnimationHelper.FadeTo(Acade
myInfo, 1, 0.5);
                                                AnimationHelper.FadeTo(grid,
 0.3, 0.5);
                                            }
                                            else
                                            {
                                                AcademyInfo.Opacity = 1;
                                                AnimationHelper.FadeTo(Acade
myInfo, 0, 0.5).Completed += (c, d) => AcademyInfo.Visibility = Visibility.Colla
psed; ;
                                                AnimationHelper.FadeTo(grid,
1, 0.5);
                                            }
                                        };
            updateBundle(self.AcademyInfo);
            AnimationHelper.AddPointerAnimation(gridLocation);
            AnimationHelper.AddPointerAnimation(gridWebsite);

        }
```

```
        private async void backToStartClicked(object sender, Windows.UI.Xaml.Inp
ut.TappedRoutedEventArgs e)
        {
            //go back to the previous page
            await CoreData.SaveTimetable(self);
            Frame.Navigate(typeof(StartPage));



        }

        public void updateBundle(AcademyBundle bundle)
        {
            //set values
            txtAcademyName.Text = bundle.Name;
            txtAcademyWebsite.Text = bundle.Website;
            txtAcademyLocation.Text = bundle.Location;
            spEventsHolder.Children.Clear();

            //set color to the chosen Accent
            txtAcademyName.Foreground = new SolidColorBrush(bundle.Accent);
            lblLocation.Foreground = new SolidColorBrush(bundle.Accent);
            lblWebsite.Foreground = new SolidColorBrush(bundle.Accent);
            lblEvents.Foreground = new SolidColorBrush(bundle.Accent);

            //split all event lines
            List<string> lines = bundle.Events.Split('\n').ToList();
            //Perform bubble sort
            while (!isChrnological(lines))
            {
                for (int i = 0; i < lines.Count - 1; i++)
                {
                    if (CoreTime.ParseString(lines[i]) > CoreTime.ParseString(li
nes[i + 1]))
```

```
                    {
                        string temp = "";
                        temp = lines[i];
                        lines[i] = lines[i + 1];
                        lines[i + 1] = temp;
                    }
                }

            }

            foreach (string line in bundle.Events.Split('\n'))
            {
                try
                {
                    TextBlock title = new TextBlock { FontSize = 16.5, Foregroun
d = new SolidColorBrush(Colors.Black), Text = line.Split(',')[1] };
                    spEventsHolder.Children.Add(title);
                    TextBlock date = new TextBlock { FontSize = 16.5, Foreground
 = new SolidColorBrush(Colors.DarkGray), Margin = new Thickness(0, 0, 0, 15), Te
xt = line.Split(',')[0] };
                    spEventsHolder.Children.Add(date);
                }
                catch {}
            }
        }
        //returns whether the events are in the correct order.
        public bool isChrnological(List<string> ls)
        {
            for (int i = 0; i < ls.Count-1; i++)
            {
                if (CoreTime.ParseString(ls[i]) > CoreTime.ParseString(ls[i + 1]
))
                {
                    return false;
                }
            }
            return true;
```

```
        }
    }
```

SYSTEM MAINTANANCE

SYSTEM OVERVIEW

This application is for students who want to better manage all their educational information. It allows them to store their schedule, notes, definitions, homework and course structure. Each piece of information is neatly presented in the most efficient way on the Main Page which aggregates all information that is glanced at. The user or academy can also add academy information which contains the name, location, website and a list of events of the college - the academy may write a simple script which creates .tb files that have embedded their information.

Each timetable has its own XML file, which is the Data Folder, which in turn is located in the app's isolated storage. When creating a timetable, a new instance of Timetable class is made with the given validated name and description and serialized to an XML file with the same name and a .tb extension (for file association purposes). When moving to the MainPage , a Timetable is sent as a parameter and used to fill all the information.

The TimetableView - the visual representation of the timetable is drawn by filtering all instances by their Day property and then drawing each Instance by multiplying the standard size by the difference between the end time and the start time which gives their relative height.

The MainPage also has previews which the user can glance at , he can then jump straight from preview to the editor by pressing the edit button. In this case , an array of size 2 and data type object is sent as a parameter (all classes derive from object so anything can be

sent) , the first one is the timetable which is used to save data and the Course which is the relevant course the preview was portraying.
All changes are stored in the memory unless they are major. Saves occur on page transitions. The timetables are store in a list inside the static CoreData class and can be accessed from everywhere. The method CoreData.SaveTimetable(table) can be used to save the timetable to it's appropiate XML file.
Making the tile shortcuts for each editor is done using the platform's live tile API , a transparent image is overlayed on top of the course color and the use may choose the name (although it is already suggested to an appropiate name).

Algorithms
One of the algorithms I implemented is Bubble Sort , a standard sorting algorithm. It is used twice but I have not abstracted it to a general class because it does not sort the same kind of data.I use it to sort through Homework and Events. The complexity of creating a bridge class that could be inherited to make the same bubble sort work for both is uneeded complexity which is not appropiate for this task. Furthermore the Homework will need to be wrapped in a struct for association purposes. Here is how it works:

A struct Homework Data is used to bind the name of a module to a homework.

```
struct HomeworkData
{
    public HomeworkTask homework;
    public string ModuleName;
}
```

Two additional methods are used to modularize (split into parts) the process : isOrdered and Swap

```
//this will return true when the homeworks are sorted
bool isOrdered(List<HomeworkData> hw)
{
    //check each adjecent pair to see if they are in order
    for (int i = 0; i < hw.Count -1 ; i++)
    {
        var current = hw[i];//the current HomeworkData
```

```
        var next = hw[i+1];//the adjecent one
        if (current.homework.DueOn > next.homework.DueOn) return false;
    }
    //this statement will only be reached if false is not returned before the checking is
complete
    return true;
}
```

Note about the swap method uses a temporary value. It is possible to perform a swap using two variables and simple addition however I have refrained to do so because it appears the compiler optimizes this general statement and makes it faster then any clever method would , this is a general case of premature optimization.

```
//swaps two indexes using a temporary value
void swap(List<HomeworkData> hw, int a, int b)
{
    //we need a temporary value to hold one of the values while switching
    HomeworkData temp = hw[a];
    hw[a] = hw[b];
    hw[b] = temp;
}
```

Here is the actual bubble sort as written in the constructor - first the target list is made:

```
//The user might want to see which module the homework is for
//therefore we must create a list of homework with the module name attached
List<HomeworkData> data = new List<HomeworkData>();
foreach (Module m in model.Modules) foreach (HomeworkTask tk in m.Homeworks)
    {
        data.Add(new HomeworkData { homework = tk, ModuleName = m.Name });
    }

//perform bubble sort to list them from the closest
//while the list is unordered , perform the algorithm
while (!isOrdered(data))
{
    //one complete for loop is one complete pass
    for (int i = 0; i < data.Count -1; i++)
```

```
    {
        //if a date is later than one on its right then they must be swapped
        if (data[i].homework.DueOn > data[i+1].homework.DueOn)
        {
            swap(data, i, i + 1);
        }
    }
}
```

Bubble sort is easy enough to implement and understand which makes it efficient for other programmers to contribute , the list is fairly small so it should not cause any performance issues.

Another algorithm used is Linear Search, I use it to find the earliest/latest instance time. For the earliest it check which Instance has the smallest StartTime and for the latest it checks which instance has the latest EndTime.

```
public static Time GetEarliest(Timetable tb)
{
    //This variable has to as big as possible
    Time earliest = Time.MaximumTime;

    foreach (Course t in tb.Courses)
    {
        //iterate through each Instance linearly
        foreach (Instance instance in t.Instances)
        {
            //if this is earlier than the current earliest found time then it replaces it.
            if (instance.StartTime <= earliest) earliest = instance.StartTime;
        }
    }
    //if nothing is earlier than the original value , MaximumTime is returned.
    return earliest;
}
```

PROCEDURE LIST

Name	Where	Purpose

FadeTo	AnimationHelper	Used to fade an element in or out (but can be used to fade at any percentage point)
SmoothMove	AnimationHelper	Animate a translation within a canvas.
AddPointerAnimation	AnimationHelper	Add an animation that makes an object smaller when pressed to give the user visual feedback on his action.
Initialize	CoreData	Create or open the data folder.
CreateTimetable	CoreData	Serializes a timetable to XML and returns an error if the name is already used.
EditTimetable	CoreData	Used to change the name or description of a timetable.
GetTimetables	CoreData	Used to retrieve all the timetables in the data folder.
DeleteTimetable	CoreData	Deletes the given timetable
SaveTimetable	CoreData	Saves the timetable over the old version
ExportTimetable	CoreData	Opens a file picker and serializes the timetable at that location.
ImportTimetable	CoreData	Open a file picker and copies the picked tb file.
GetEarliest	CoreTime	Gets the earliest instance in a timetable.
GetLatest	CoreTime	Gets the latest instance in a timetable.
IsOvelapping	CoreTime	Finds whether a proposed Instance time will overlap with another.
FormatTime	CoreTime	Used to format the time using standard notation (1st , 2nd , 3rd etc)
ParseString	CoreTime	Will convert a dd/mm/yyyy string to a DateTime object.
ToString	Time	Overriden to return the time in hh:mm format.
SetEditableItems	EditableItem	Sets the items that are shown when isEditable = true

SetStaticItems	EditableItem	Sets the items that are shown when isEditable is not true
SetEditable	EditableItem	Sets whether the use can edit this control.
GetTotalInstances	Timetable	Returns the total number of instances in the timetable.
hasWeekend	Timetable	Returns whether the timetable has instances on saturday or sunday.
isStringValid	Validator	Returns whether a string matches the given validation rules.
NotifyUser	ValidatorResponse	Shows the user a message box with the validation error.
RefreshInstances	ClassBox	Redraws all the instances in the timetable.
RefreshCourses	ClassesEditor	Redraws all the classes/courses in the timetable.
RefreshDefinitions	DefinitionsEditor	Redraws all the definitions in the timetable.
RefreshHomework	HomeworkEditor	Redraws all the homework in the timetable.
fillBoxes	SimpleDatePicker	Fills the day , month and year combo boxes.
PickRandomDefinition	DefinitionPreview	Finds a random definition and shows it in the preview.
isOrdered	HomeworkPreview	Returns true if the homework is sorted correctly.
swap	HomeworkPreview	Changes the positions of two homeworks in a list.
DrawWithAngle	PieChart	Draws the outer circle of the pie chart used in the Topic preview representing the studied progress
DrawInnerWithAngle	PieChart	Draws the smaller , darker circle inside of the bigger one - representing the revision progress.
FillWith	TimetableView	Draws the timetable view using the given timetable.
DrawCurrentTime	TimetableView	Draws a red line on the timetable to show the current time of the day.
FillWith	EventPreviewBox	Calculates the closest lesson and adds courses to the list in the TimetableBox

add_tapped	NewTimetableBox	Fade the front view and show the text boxes for data entry.
back_clicked	NewTimetableBox	Fade the back view and go back to the main view of the control
create_clicked	NewTimetableBox	Validates the name and description of the timetable and creates the timetable if valid.
import_clicked	NewTimetableBox	Calls the CoreData.ImportTimetable method
edit_clicked	TimetableBox	Fades to the back view for editing.
ToFront	TimetableBox	Fades to the fron view from saving or canceling
save_clicked	TimetableBox	Validates and attempts to save changes.
delete_clicked	TimetableBox	Prompts the user for confirmation and deletes the timetable
back_clicked	TimetableBox	Returns to the front view
open_clicked	TimetableBox	Triggers event that the timetable has been opened.
FillWith	EventControl	Sets the color , name and teacher name of the event box
setTime	EventControl	Sets the time displayed on this control.
changeEvent	InstanceControl	Check for overlapping errors when the user selects another time
delete_clicked	InstanceControl	Triggers delete event.
refresh	NoteBox	Reloads and redraws all NoteFragmentControls after a change.
Update	NoteFragmentControl	Updates the formatting after a change.
RefreshNotes	NoteEditor	Redraws all the notes in the timetable.
fill	ModuleList	Shows all the modules in the given course
setSelected	ModuleListItem	Can highlight or dim the control text.
RefreshTopics	TopicsEditor	Redraws all the definitions in the module.
FillWith	AcademyInfoControl	Updates the control with the current info and wires it to the Main Page for instant changes.
Fill	StartPage	Draws all the TimetableBoxes

Save	MainPage	Saves the current timetable
backToStartClicked	MainPage	Saves and goes back to the previews page
updateBundle	MainPage	Updates the information regarding the academy
isChrnological	MainPage	Used for the bubble sort where I sort the academy events.

VARIABLE LIST

Name	Data Type	Where	Purpose
EventControlHeight	int	Constants	Represents the height of one EventControl (used as a unit for the TimetableView)
EventControlWidth	int	Constants	Represent the width of one EventControl
Months	string[]	Constants	An array of length 12 containing all the names of the months.
DueOn	DateTime	HomeworkTask	When the homework is due.
Notes	string	HomeworkTask	Description of homework.
Completed	book	HomeworkTask	Whether it is completed or not.
Caption	string	Definition	The term defined
Content	string	Definition	The actual definition

Name	string	Module	The name of the module
StructureBlock	List<StructureBlock>	Module	List of all structure blocks (topics) in the module
Definitions	List<Definition>	Module	List of all definitions in the module
Notes	List<Note>	Module	List of all notes in the module
Homeworks	List<HomeworkTask>	Module	List of all notes in the module
Name	string	StructureBlock	The name of the structure block
Topics	List<Topic>	StructureBlock	The topics within the block
Name	string	Note	The name of the note
Fragments	List<NoteFragment>	Note	List of all paragraphs
Content	string	string	The text of the paragraph
FontSize	double	NoteFragment	The size of the text in this paragraph
Italic	bool	NoteFragment	Whether the text is italic
Bold	bool	NoteFragment	Whether the text is bold
Centered	bool	NoteFragment	Whether the text is bold
LocalFolder	StorageFolder	CoreData	The local folder in the

			isolated storage
DataFolder	StorageFolder	CoreData	The data folder where the xml files will be saved
Timetables	List<Timetable>	CoreData	This list contains all the timetable objects.
Serializer	XmlSerializer	CoreData	This is used to save and load timetables from .tb files.
isOverlapping	bool	TimeOverlapData	Whether an overlap has been found between the instance and any other.
OverlappingEventName	string	TimeOverlapData	The name of the event the conflicting instance belongs to.
Minutes	int	Time	The minute component of this time
Hours	int	Time	The hour component of this time
Name	string	Course	The name of the course
Room	string	Course	The ID/name of the room
Teacher	string	Course	The name of the teacher of this course
Notes	string	Course	A small note or reminder

			attached to the course.
TileColor	Color	Course	The accent color that will be used to color code this course
Instances	List<Instance>	Course	A list of all the instances (lessons) of the course.
Modules	List<Module>	Course	A list of all modules of this course.
StartTime	DateTime	Instance	The beginning time of the lesson
EndTime	DateTime	Instance	The end time of the lesson
Day	int	Instance	The day on which this lesson takes place.
EventName	string	Instance	The name of the Course it is in.
Static	List<FrameworkElement>	EditableItem	A list of elements to be shown when editing mode is off
Editable	List<FrameworkElement>	EditableItem	A list of elements to be shown when editing mode is on
edit	bool	EditableItem	Whether editing mode is on or off
Name	string	Timetable	The name of the timetable.

Description	string	Timetable	The description of the timetable.
AcademyInfo	AcademyBundle	Timetable	Contains information about the related academy
Courses	List<Course>	Timetable	A list of all courses in a timetable
Name	string	AcademyBundle	The name of the academy
Location	string	AcademyBundle	The adress of the academy
Website	string	AcademyBundle	Their website
Accent	Color	AcademyBundle	A color that is used to brand them
Events	string	AcademyBundle	A string containing all the events happening
TimetableTitleMax **TimetableDescriptionMax** **EventClassMax** **EventNotesMax** **EventTitleMax** **EventTeacherMax** **HomeworkDescriptionMax** **ModuleNameMax** **DefinitionNameMax** **DefinitionDescriptionMax** **StructureBlockNameMax** **TopicTitleMax** **NoteNameMax** **NoteContentMax**	int	Validator	Validation constants show the maximum length for their respective fields.

AcademyNameMax **AcademyLocationMax** **AcademyWebsiteMax** **AcademyEventsMax**			
isValid	bool	ValidatorResponse	Whether the string met the validation requirements
Errors	List<string>	ValidatorResponse	A list of all validation error messages , if any.
self	Course	ClassBox	A reference to the course being edited
selfTB	Timetable	ClassBox	A reference to the timetable that contains that course
self	Timetable	ClassesEditor	A reference to the current timetable.
self	Definition	DefinitionControl	A reference to the definition being edited
self	Course	DefinitionsEditor	A reference to the course containing the definitions being edited.
self	Course	HomeworkEditor	A reference to the course containing the homework being edited.
self	HomeworkTask	HomeworkItem	A reference to the homework being edited.
PickedTime	DateTime	SimpleDatePicker	Represents the date the

			user has selected.
Month	int	SimpleDatePicker	Property to safely retrieve the month selected
Day	int	SimpleDatePicker	Property to safely retrieve the day selected
defs	List<Definition>	DefinitionPreview	Unified collection of all the definitions within a module.
rnd	Random	DefinitionPreview	Genrates random numbers
curve	Polyline	PieChart	The outer ring of the revised/studied chart
curve2	Polyline	PieChart	The inner ring of the revised/studied chart
back1	Polyline	PieChart	The background outline from the inner ring to the center
back2	Polyline	PieChart	The background outline from the inner ring to the outer ring
back3	Polyline	PieChart	The background outline from

			the outer ring to the outside of the shape
unit	double	PieChart	The width and height of the PieChart control.
TimeLine	Rectangle	TimetableView	A rectangle that reasembles a thick line that is overlayed on the timetable to show the current time location.
instances	List<EventControl>[]	TimetableView	An array of lists , each list representing a day of the week and containing the event controls for that day.
lowerBound	Time	TimetableView	The earliest instance start time
upperBound	Time	TimetableView	The latest instance end time
self	Timetable	TimetableBox	A reference to the timetable this box represents.
SelectedElement	Rectangle	ColorChooser	The currently chosen element (color filled rectangle)
SelectedColor	Color	ColorChooser	Property to get and set

			the current color.
self	Course	EventControl	A reference to the course this represents.
self	Instance	InstanceControl	A reference to the current instance
selfTB	Timetable	InstanceControl	A reference to the timetable the instance is in (used for checking overlaps)
self	Module	ModuleBox	A reference to the module it represents
self	Course	ModulesEditor	A reference to the course containing the modules to edit.
self	Note	NoteBox	The note being edited
color	Color	NoteBox	The color of the note being edited
self	NoteFragment	NoteFragmentControl	The current paragraph object being edited
self	Course	NotesEditor	The course containing the notes being edited.
Chosen	Module	ModuleList	The module selected in a module list.
self	Module	ModuleListItem	The module that this

			option item represents
col	Color	ModuleListItem	The color of the course the module is in.
self	Course	TopicsEditor	The course containing the topics being edited
self	StructureBlock	StructureBlockControl	The StructureBlock that is represented by this control.
self	Topic	TopicControl	The topic represented by this control.
Icon	string	UniversalButton	The text symbol that is the icon of this button.
self	Timetable	MainPage	The timetable currently in use.
query	string	SearchResultsPage	The string to search for.

AUTOMATICALLY GENERATED COMPONENTS

Visual Studio automatically generates .xaml files (markup) from my UI design , it also provides the skeleton of the application in the file App.cs , which includes event handlers for thing such as application suspension or file opening. A third party control called Flyout is used to provide with a floating panel for the note editing - this library is endorsed by Microsft.

None of the above have been included as I have not written them.

SYSTEM TESTING

VALIDATION TESTING

Test Number	Purpose	Expected result	Actual result	Passed
1	Check whether the app correctly rejects titles that are too long.	Message box informing the user that it is not a valid title.	Message box shown. See figure 1.	YES
3	Check whether the app correctly rejects empty titles for timetables.	Message box informing the user that it is not a valid title.	Message box shown. See figure 2.	YES
3	Check whether the app correctly rejects descriptions that are too long.	Message box informing the user that it is not a valid description.	Message box shown. See figure 3.	YES
4	Checks the length of both the title and description.	Message box informing the user that BOTH fields are not valid.	Message box shown. See figure 4.	YES
5	Checks title for illegal filename.	Message box informing the user of which characters are not allowed in filenames.	Message box shown. See figure 5.	YES

6	Checks whether title is valid.	Creates timetable and shows it in the list.	Timetable created. See figure 6.	YES
7	Check whether a class name is empty.	Informs the user that it cannot be empty.	Message box shown. See figure 7.	YES
8	Checks whether a class name is too long.	Informs the user it cannot be over 20 characters and fixes it.	Message box shown , last character deleted. See figure 8.	YES
9	Checks whether a class description is too long.	Informs the user that the description has to be under 300 characters.	Message box shown , last character deleted. See figure 9.	YES
10	Checks whether the teacher name is too long.	Informs the user that the name has to be under 25 characters.	Message box shown,last character deleted. See figure 10.	YES
11	Checks whether the room name is too long.	Informs the user that the room name has to be under 7 characters.	Message box shown, last character deleted. See figure 11.	YES
12	Checks whether the module name is too long.	Informs the user that the module name has to be under 25 characters.	Message box shown, last character deleted. See figure 12.	YES
13	Checks whether the topic title is too long.	Informs the user that the title has to be under 20 characters.	Message box shown, last character deleted. See figure 13.	YES
14	Checks whether the structure block title is too long.	Informs the user that the title has to be under 30 characters.	Message box shown, last character deleted. See figure 14.	YES
15	Checks whether the description of a homework is too long.	Informs the user that the description has to be under 500 characters	Message box shown, last character deleted. See figure 15.	YES
16	Checks whether the title of a definition is too long.	Informs the user that the title has to be under 35 characters	Message box shown, last character deleted. See figure 16.	YES
17	Checks whether the definition is too long.	Informs the user that the title has to be under 300 characters	Message box shown, last character deleted. See figure 17.	YES

18	Checks whether the timetable name is valid.	Informs the user of the single invalid character.	Message box shown. See figure 18.	YES
19	Combined test 18 with a description of over 200 characters	Informs the user that the description has to be under 200 characters AND the title is invalid.	Combined message box shown , last character of description deleted. See figure 19.	YES

Figure 1
Rejecting title that is too long.

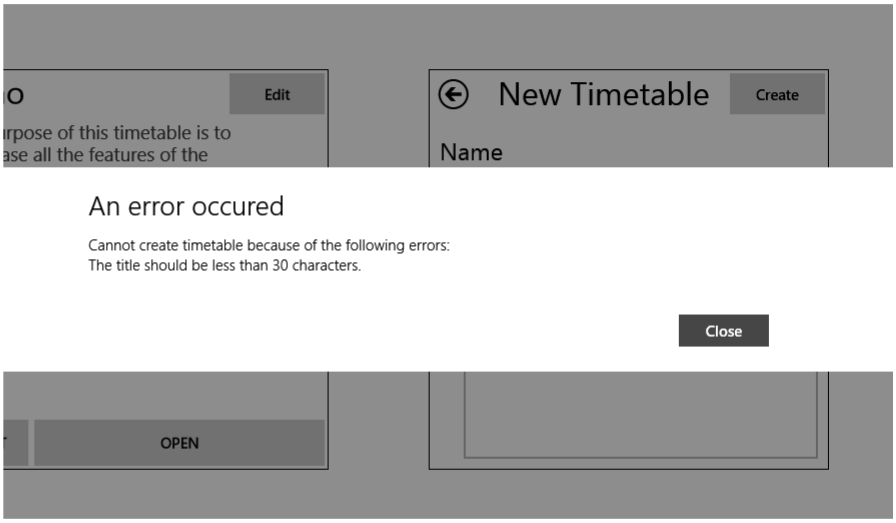


Figure 2
Rejecting title that is too long.

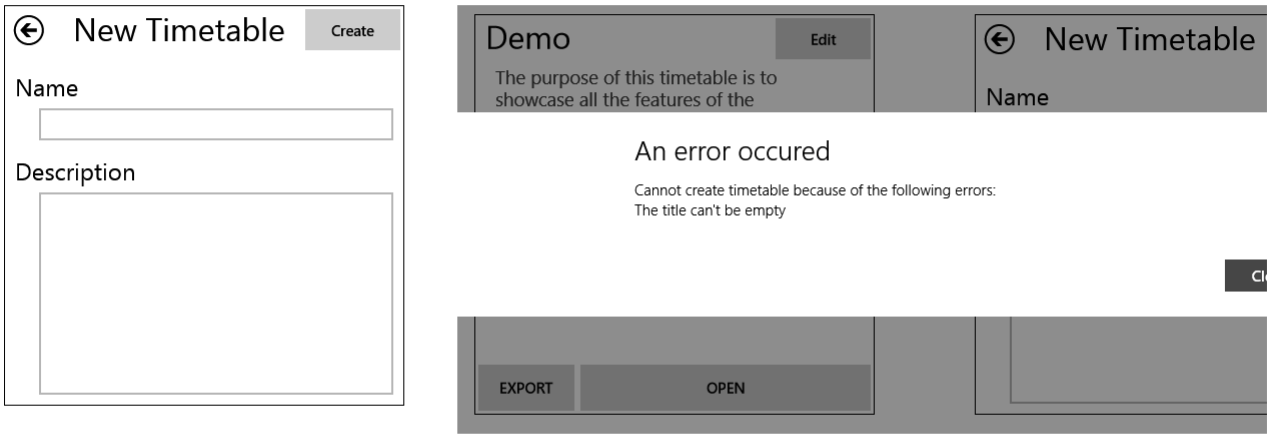


Figure 3
Description too long.

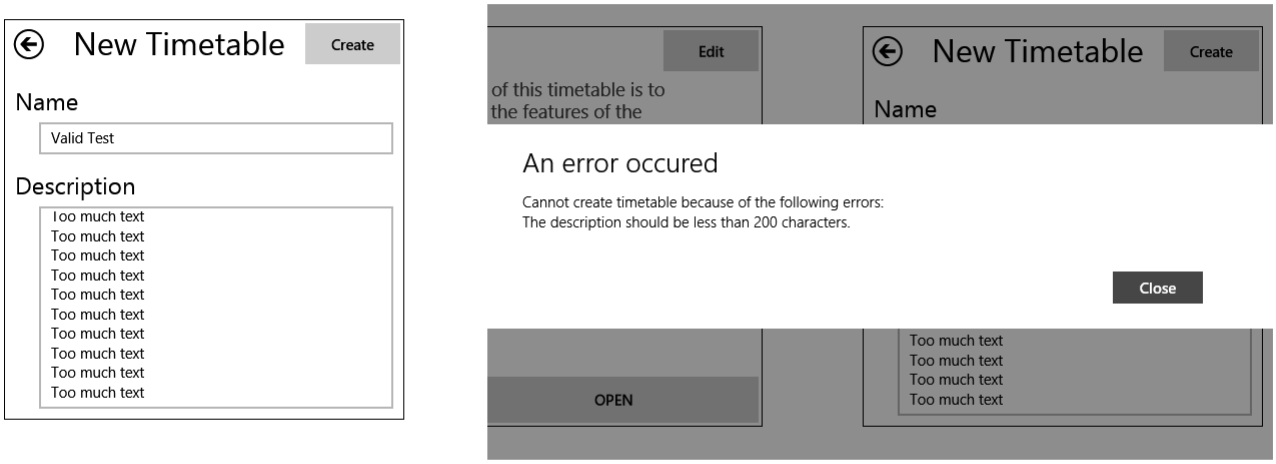


Figure 4
Multiple lenght checks.

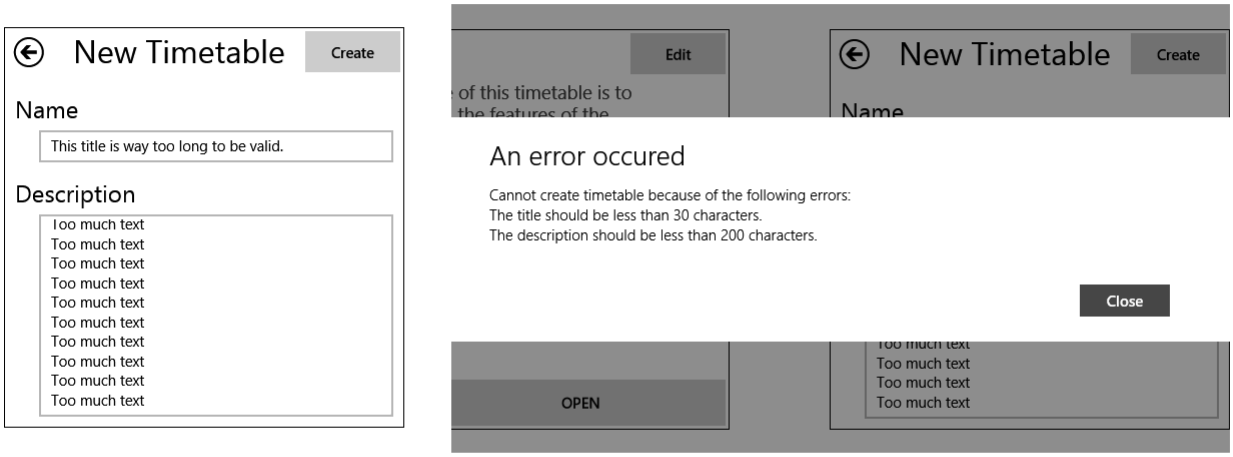


Figure 5
Checking illegal filename

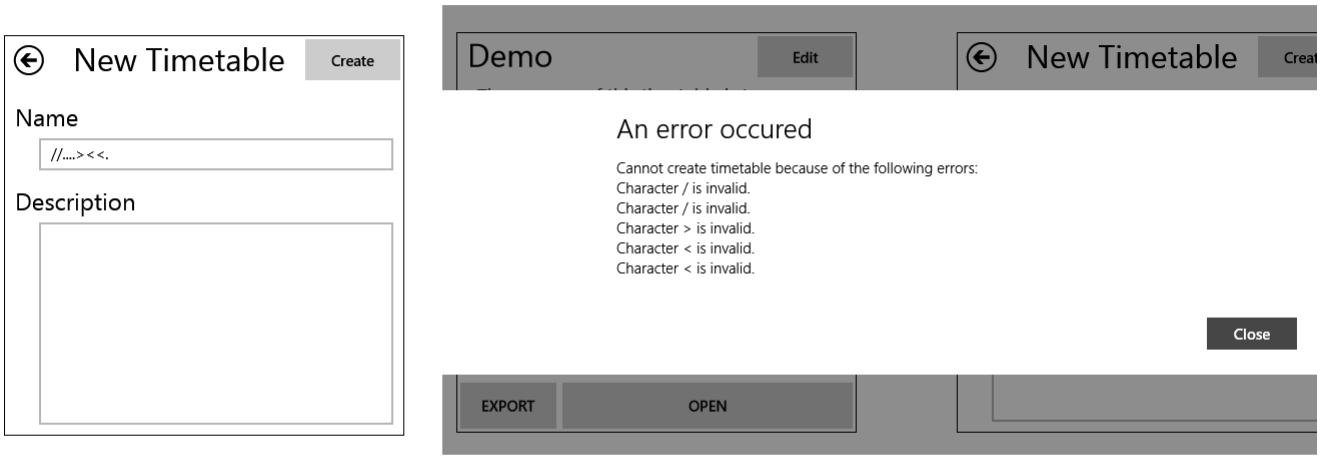


Figure 6
Valid title+valid description.

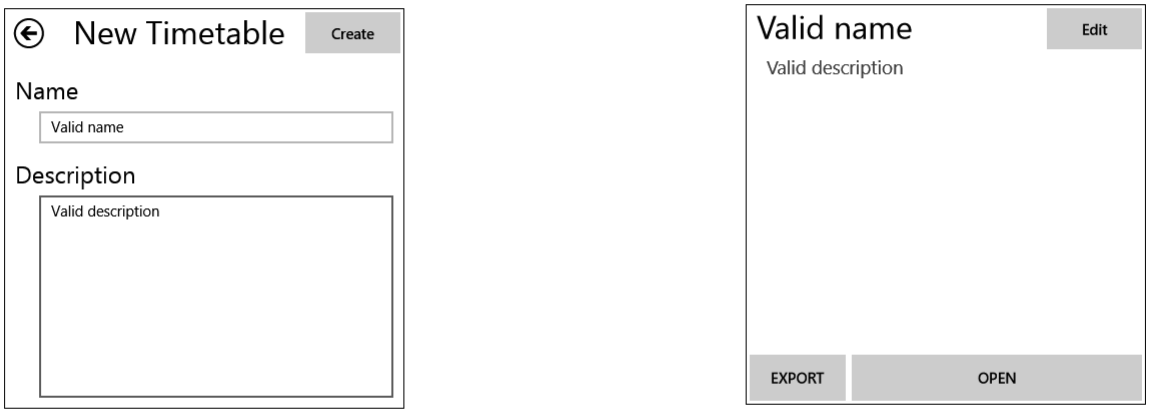


Figure 7
Auto-fix empty name

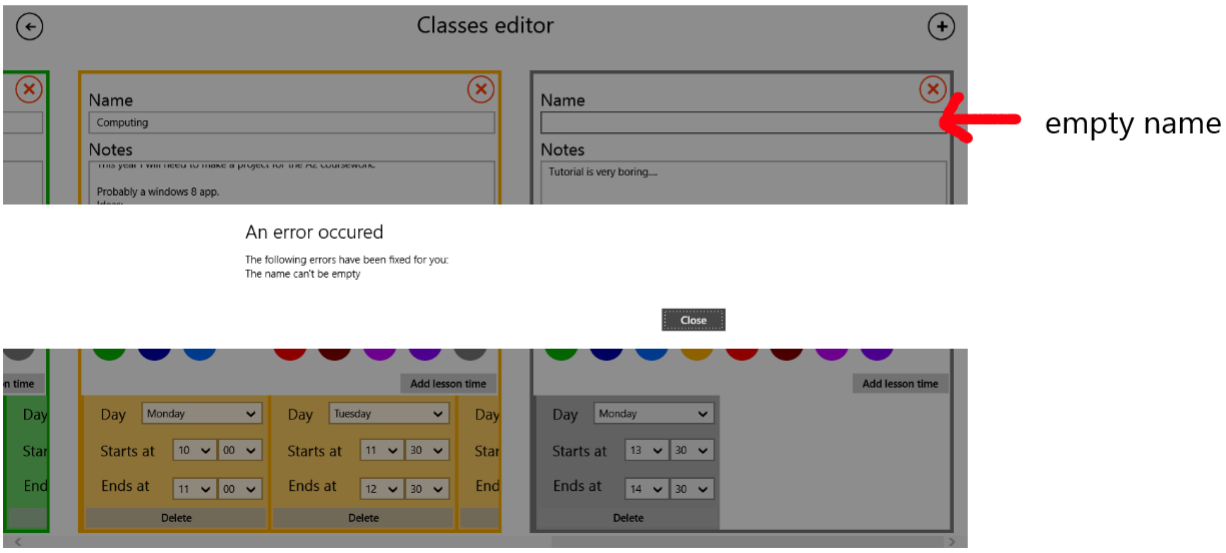


Figure 8
Auto-fix when name is too long.

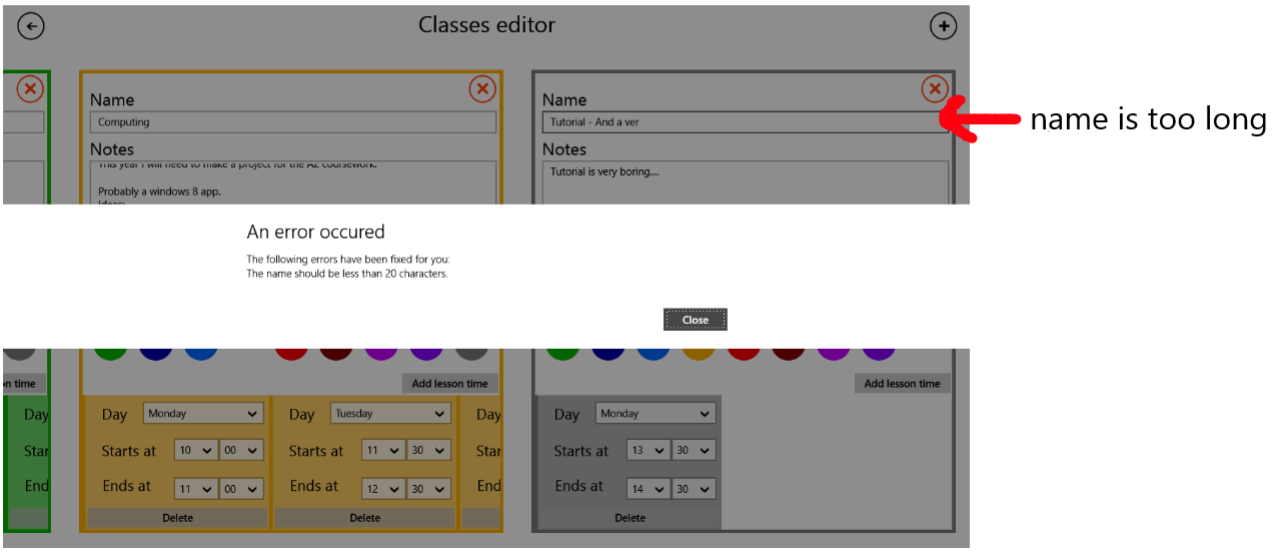


Figure 9
Auto-fix long description

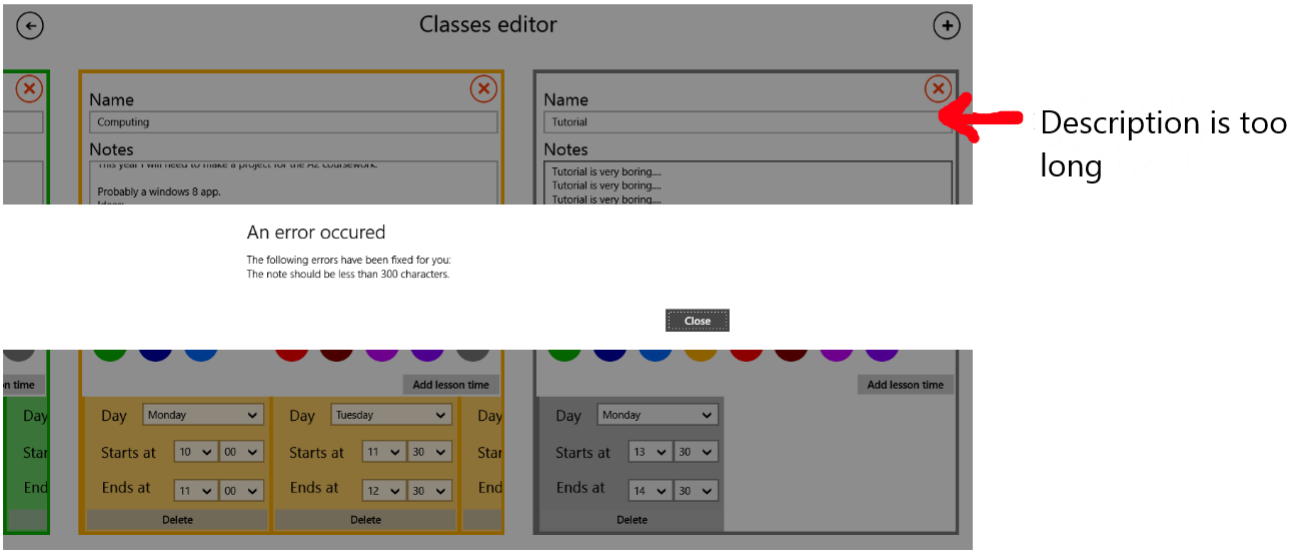


Figure 10
Teacher name too long.

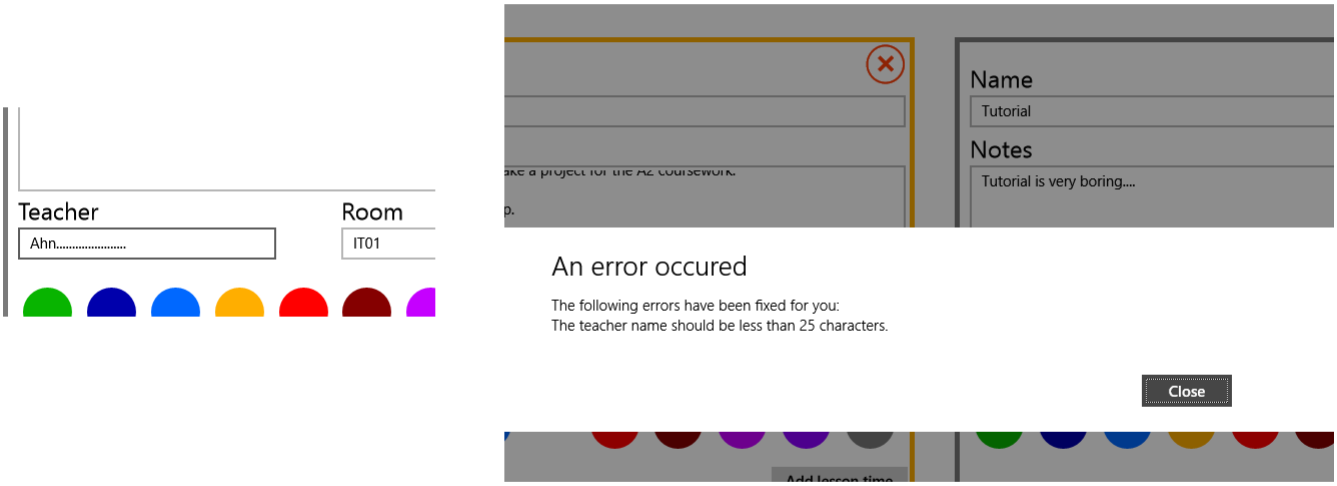


Figure 11
Room name too long

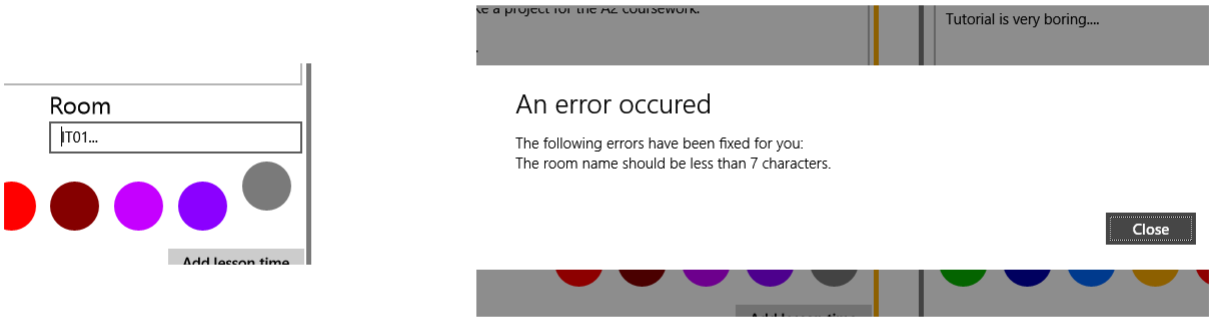


Figure 12
Module name too long.

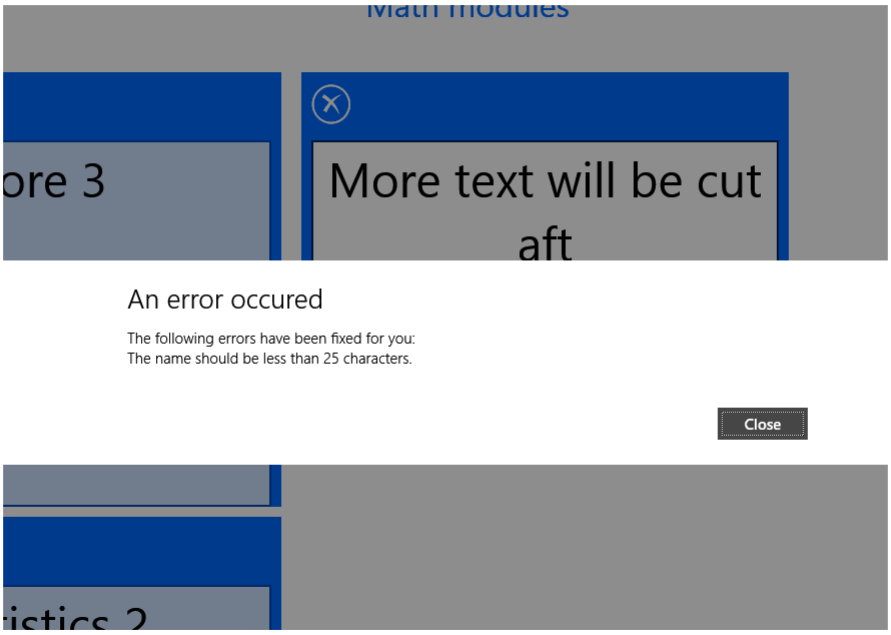


Figure 13
Inner title too long

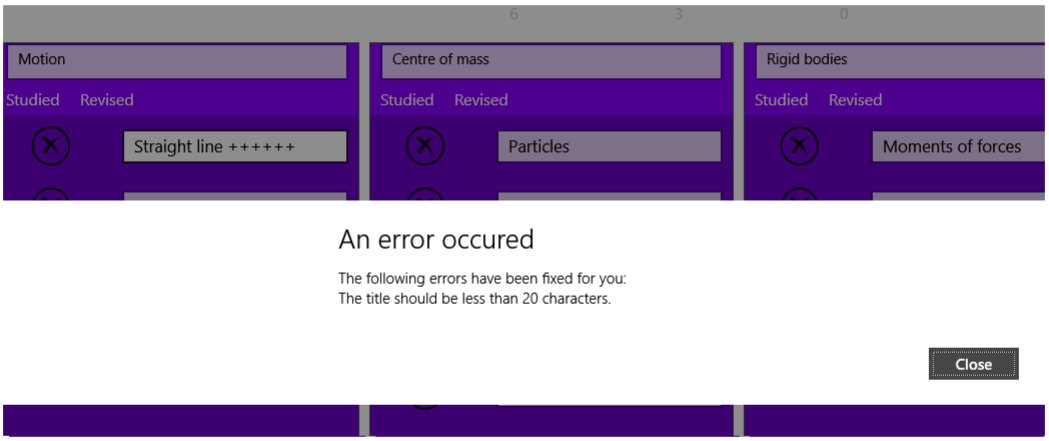


Figure 15
Homework description is too long

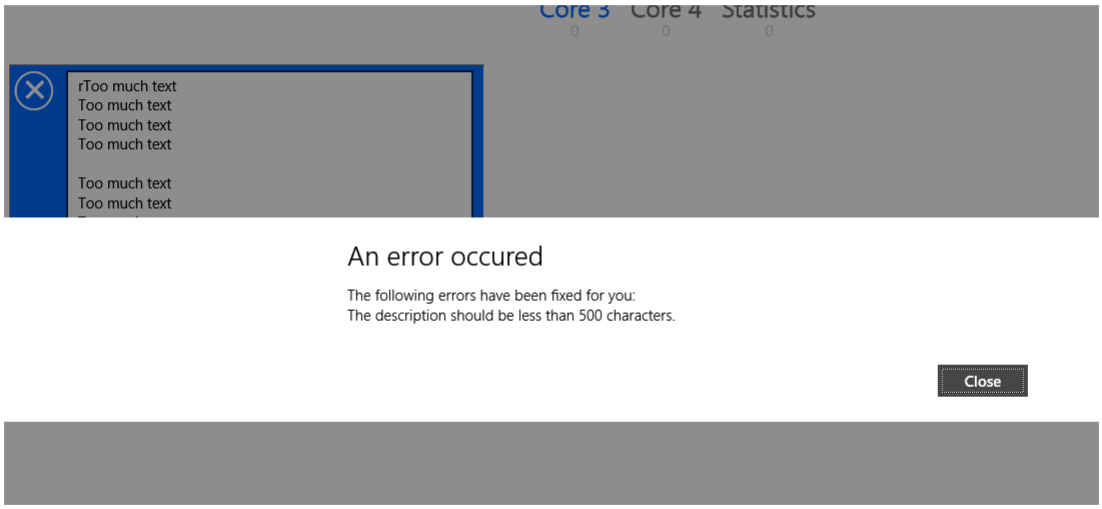


Figure 16
Definition title too long.

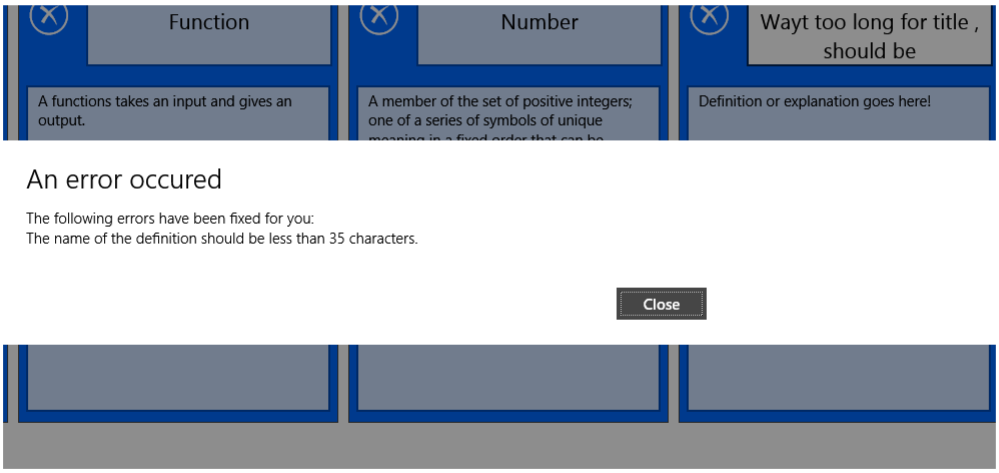


Figure 17
Definition description is too long.

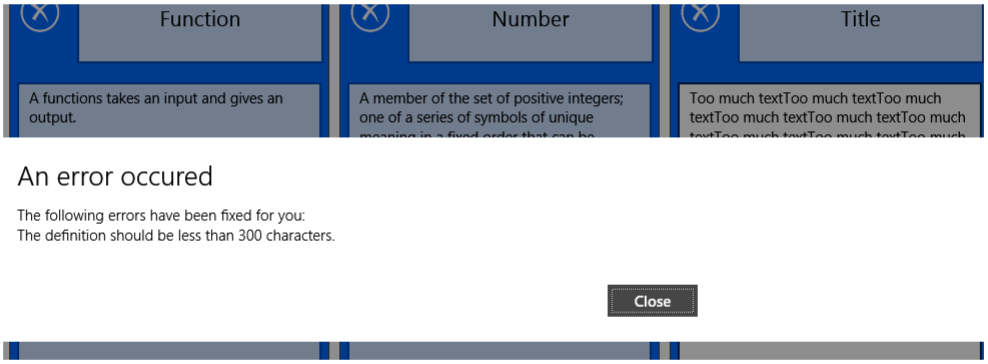


Figure 18
Single invalid character for timetable name.

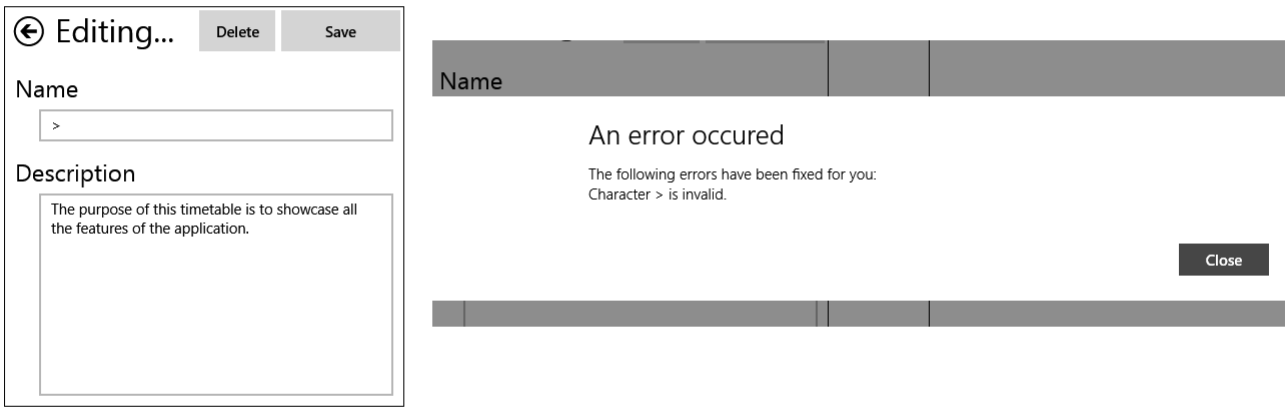
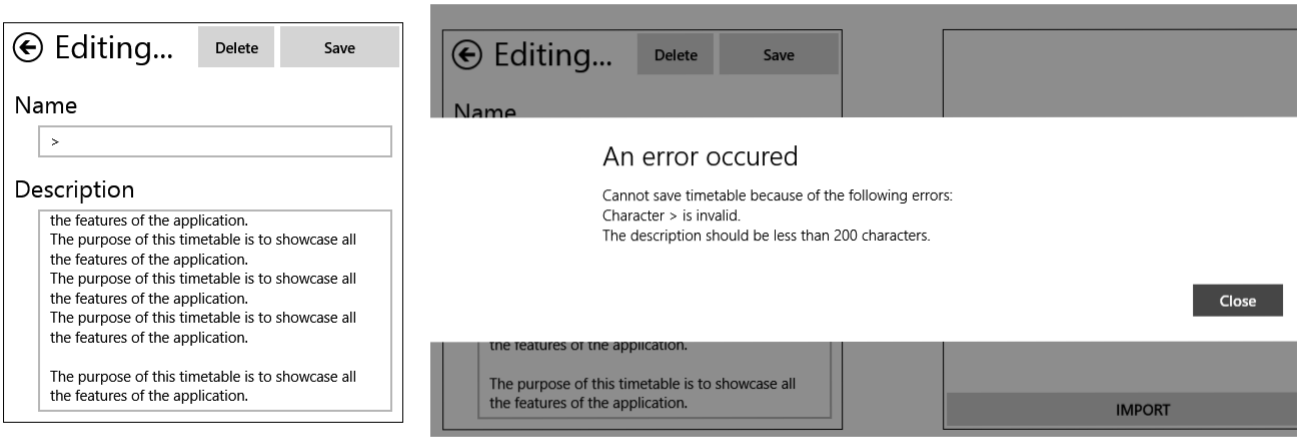


Figure 19
Invalid chacter + description too long.



UNIT TESTING

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using Microsoft.VisualStudio.TestPlatform.UnitTestFramework;
using TimetableApp;

namespace Tests
{
    [TestClass]
    public class GeneralTests
    {


        //This method will check whether the property
        //TotalMinutes of the Time class returns the correct amount of minutes.
        [TestMethod]
        public void TimeTotalMinutes()
        {
            //creating two Time instances
            var time1 = new Time();
            var time2 = new Time();

            //time 1 is at 8:30
            time1.Hours = 8;
            time1.Minutes = 30;

            //time 2 is at 8:31 , very close to time1
            time2.Hours = 8;
            time2.Minutes = 31;

            //Calculating the values we sh
            double actualTotal1 = 8 * 60 + 30;
            double actualTotal2 = 8 * 60 + 31;

            //compare whether the TotalMinutes property return the value we predicted
            Assert.AreEqual(actualTotal1, time1.TotalMinutes);
            Assert.AreEqual(actualTotal2, time2.TotalMinutes);
        }

        //Two equal times that will be used in a couple of
        //test methods. Both are at 8:09
        Time timeA = new Time { Hours = 8, Minutes = 9 };
        Time timeB = new Time { Hours = 8, Minutes = 9 };


        //This method will check whether the > operator works as intended
        [TestMethod]
        public void TimeBigger()
        {
            //timeA is not bigger than timeB , so the answer should be false
            bool actual = false;
```

```
            //timeA>timeB should return false
            Assert.AreEqual(actual, timeA > timeB);
        }

        //This method will check whether the < operator works as intended
        [TestMethod]
        public void TimeSmaller()
        {
            //timeA is not smaller than timeB , so the answer should be false
            bool actual = false;
            //timeA<timeB should return false
            Assert.AreEqual(actual, timeA < timeB);
        }

        //This method will check whether the >= operator works as intended
        [TestMethod]
        public void TimeBiggerEqual()
        {
            //timeA is not bigger than timeB but is equal to it , so the answer should be
 true
            bool actual = true;
            //timeA>=timeB should return true
            Assert.AreEqual(actual, timeA >= timeB);
        }

        //This method will check whether the <= operator works as intended
        [TestMethod]
        public void TimeSmallerEqual()
        {
            //timeA is not smaller than timeB but is equal to it , so the answer should b
e true
            bool actual = true;
            //timeA<=timeB should return true
            Assert.AreEqual(actual, timeA <= timeB);
        }

        //this method will compare whether the = operator has been implemented successful
ly
        [TestMethod]
        public void TimeEqual()
        {
            //we will make a few Instances to check boundaries
            Time a = new Time { Hours = 9, Minutes = 30 };
            Time b = new Time { Hours = 9, Minutes = 30 };
            Time c = new Time { Hours = 9, Minutes = 31 };

            Assert.AreEqual(true, a == b); //a and b are equal
            Assert.AreEqual(false, b == c);//b anc are not equal(1 minute difference)
        }



        //this method will check whether the method GetEarliest
        //from the class CoreTime works as intended.
        //The purpose of the method is to find the earliest Time of any instance in the t
imetable
        [TestMethod]
```

```
        public void TimetableEarliest()
        {
            //testTable will contain two Courses X and Y
            //Course X will contain Instances A (starts at 10:30) and B (starts at 12:30)
            //Course Y will contain Instances C (starts at 14:15) and D (starts at 15:45)
            Timetable testTable = new Timetable();

            Instance A = new Instance { StartTime = new Time { Hours = 10, Minutes = 30 }
};
            Instance B = new Instance { StartTime = new Time { Hours = 12, Minutes = 30 }
};
            Course X = new Course();
            X.Instances.Add(A);
            X.Instances.Add(B);

            Instance C = new Instance { StartTime = new Time { Hours = 14, Minutes = 15 }
};
            Instance D = new Instance { StartTime = new Time { Hours = 15, Minutes = 45 }
};
            Course Y = new Course();
            Y.Instances.Add(C);
            Y.Instances.Add(D);


            testTable.Courses.Add(X);
            testTable.Courses.Add(Y);

            //out of all the instances A is the earliest at 10:30
            Time ActualLowest = new Time { Hours = 10, Minutes = 30 };
            //We already know from method TimeEqual that the == operator works
            //so we can use it to evaluate this equality
            Assert.AreEqual(true, ActualLowest == CoreTime.GetEarliest(testTable));

        }

        //this method will check whether the method GetLatest
        //from the class CoreTime works as intended.
        //The purpose of the method is to find the latest Time of any instance in the tim
etable
        [TestMethod]
        public void TimetableLatest()
        {
            //testTable will contain two Courses X and Y
            //Course X will contain Instances A (ends at 10:30) and B (ends at 12:30)
            //Course Y will contain Instances C (ends at 14:15) and D (ends at 15:45)
            Timetable testTable = new Timetable();

            Instance A = new Instance { EndTime = new Time { Hours = 10, Minutes = 30 } }
;
            Instance B = new Instance { EndTime = new Time { Hours = 12, Minutes = 30 } }
;
            Course X = new Course();
            X.Instances.Add(A);
            X.Instances.Add(B);

            Instance C = new Instance { EndTime = new Time { Hours = 14, Minutes = 15 } }
;
```

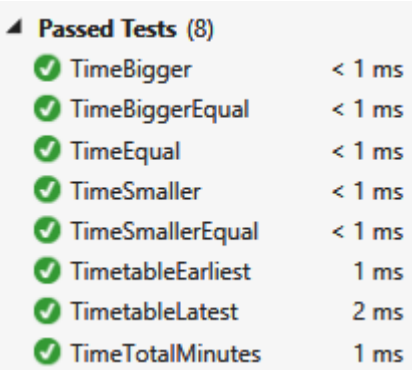
```
            Instance D = new Instance { EndTime = new Time { Hours = 15, Minutes = 45 } }
;
            Course Y = new Course();
            Y.Instances.Add(C);
            Y.Instances.Add(D);


            testTable.Courses.Add(X);
            testTable.Courses.Add(Y);

            //out of all the instances D is the latest at 15:45
            Time ActualHighest = new Time { Hours = 15, Minutes = 45 };
            //We already know from method TimeEqual that the == operator works
            //so we can use it to evaluate this equality
            Assert.AreEqual(true, ActualHighest == CoreTime.GetLatest(testTable));

        }
    }
}
```

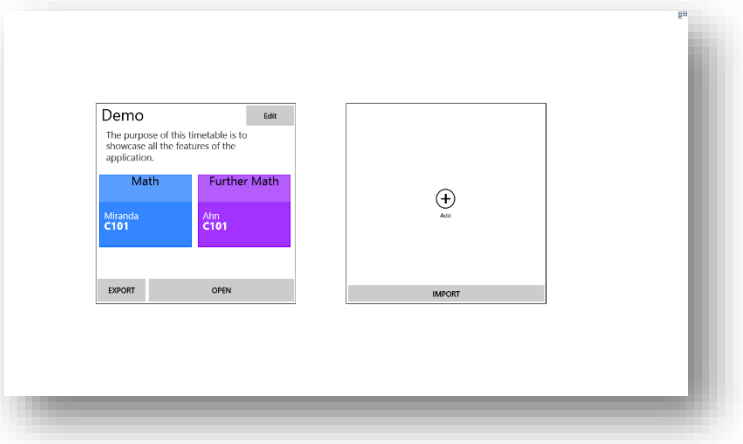
All the test passed:

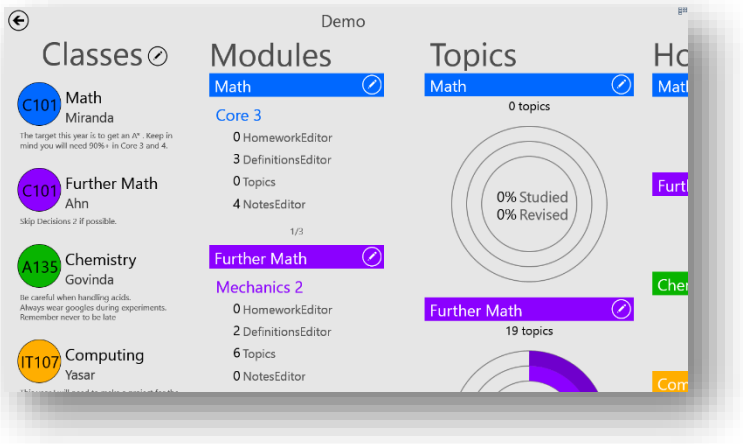


GENERAL TESTING

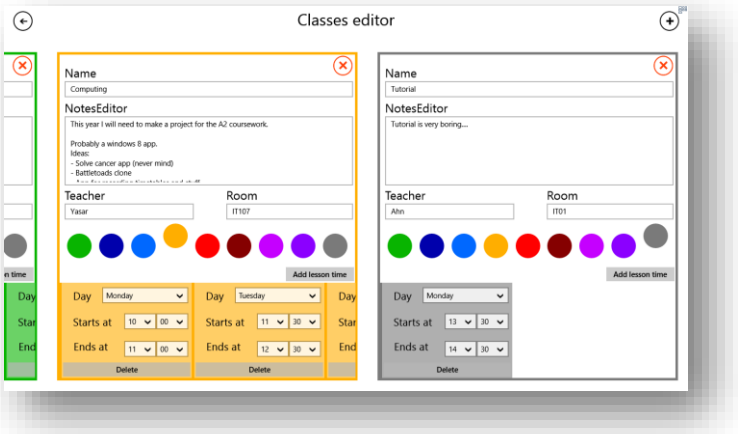
Test Number	Purpose	Expected result	Actual result	Passed
1	Show that opening a timetable works.	The MainPage is shown	The MainPage is shown, see figures 1.	YES
2	Show that adding a new course/class works.	The ClassBox is added.	The ClassBox is added, see figures 2.	YES
3	Show that deleting a course works.	The ClassBox is deleted.	The ClassBox is deleted, see figures 3.	YES
4	Show that deleting a definition control works by pressing the delete button.	The definition control is removed from the holder.	It is removed, see figure 4.	YES
5	Show that pinning the definitions page to the start screen works.	A shortcut with the course color appears on the start screen.	It is created, see figures 5.	YES
6	Show that pressing the edit button will toggle edit mode for all the controls in the holder.	The TextBlocks(labels) will be hidden and the editable TextBoxes will be shown.	Editing mode is enabled, see figures 6.	YES
7	Show that pressing the import button bring up the file picker.	A full-screen file picker should show up.	It is shown, see figures 7.	YES
8	Show that pressing the edit button in a DefinitionPreview will bring up the definitions page for that course	The page should show up with a blue color scheme and heading Math definitions.	The page is shown, see figures 8.	YES

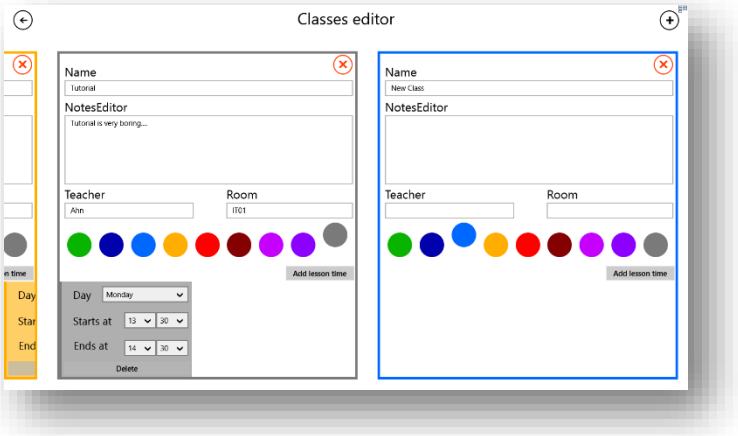
FIGURES 1


Demo
Edit
The purpose of this timetable is to showcase all the features of the application.
Math
Further Math
Miranda
C101
Ahn
C101
EXPORT
OPEN
Add
IMPORT

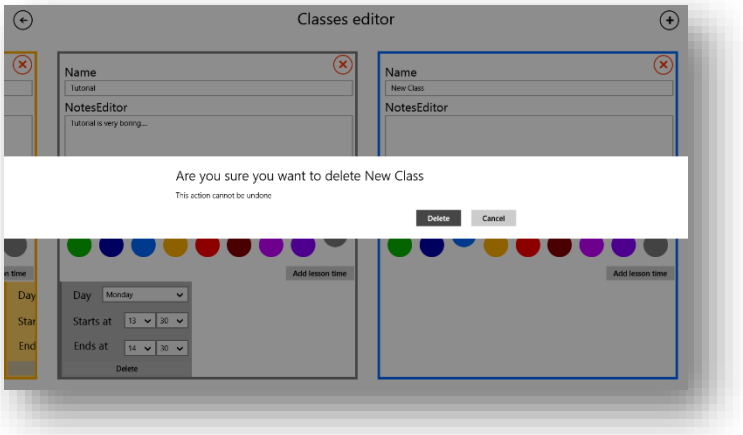

Demo
Classes
Modules
Topics
C101
Math
Miranda
C101
Further Math
Ahn
A135
Chemistry
Govinda
IT107
Computing
Yasar
Math
Core 3
0 HomeworkEditor
3 DefinitionsEditor
0 Topics
4 NotesEditor
1/3
Further Math
Mechanics 2
0 HomeworkEditor
2 DefinitionsEditor
6 Topics
0 NotesEditor
Math
0 topics
0% Studied
0% Revised
Further Math
19 topics

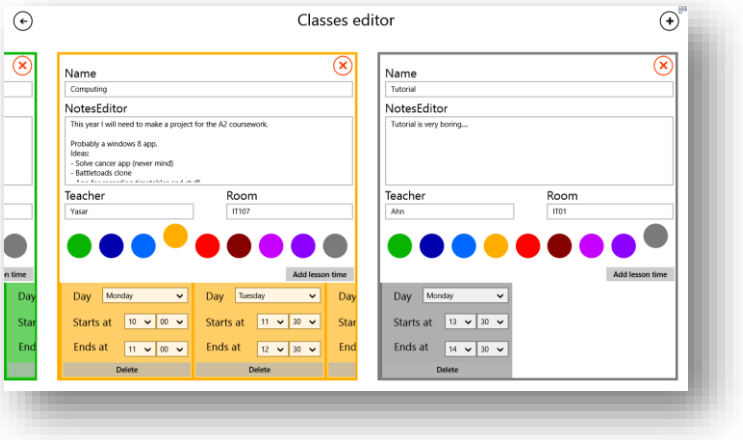
FIGURES 2


Classes editor
Name
Computing
NotesEditor
This year I will need to make a project for the A2 coursework.
Probably a windows 8 app.
Ideas:
- Solve cancer app (never mind)
- Battletoads clone
Teacher
Yasar
Room
IT107
Add lesson time
Day
Monday
Starts at
10
00
Ends at
11
00
Delete
Day
Tuesday
Starts at
11
30
Ends at
12
30
Delete
Name
Tutorial
NotesEditor
Tutorial is very boring....
Teacher
Ahn
Room
IT01
Add lesson time
Day
Monday
Starts at
13
30
Ends at
14
30
Delete

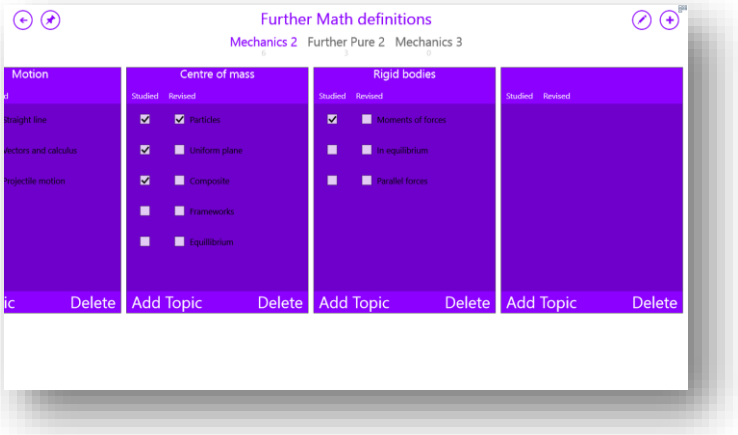

Classes editor
Name
Tutorial
NotesEditor
Tutorial is very boring....
Teacher
Ahn
Room
IT01
Add lesson time
Day
Monday
Starts at
13
30
Ends at
14
30
Delete
Name
New Class
NotesEditor
Teacher
Room
Add lesson time

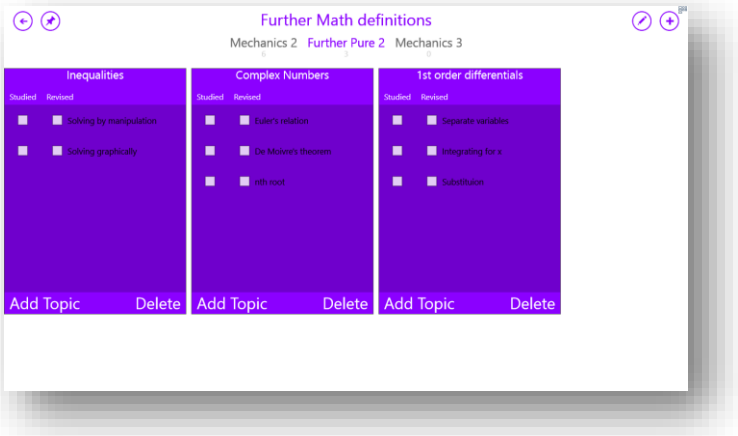
FIGURES 3


Classes editor
Name
Tutorial
NotesEditor
Tutorial is very boring...
Name
New Class
NotesEditor
Are you sure you want to delete New Class
This action cannot be undone
Delete
Cancel
Add lesson time
Day
Monday
Starts at
Ends at
Delete

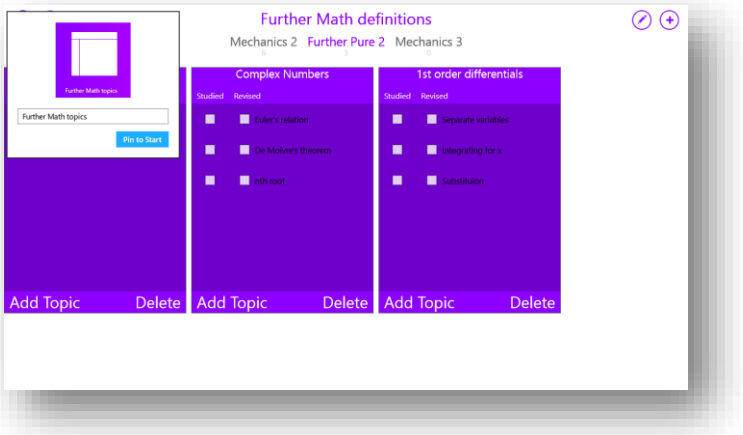

Classes editor
Name
Computing
NotesEditor
This year I will need to make a project for the A2 coursework.
Probably a windows 8 app.
Ideas:
- Solve cancer app (never mind)
- Battletoads clone
Teacher
Yasar
Room
IT107
Name
Tutorial
NotesEditor
Tutorial is very boring...
Teacher
Ahn
Room
IT01
Add lesson time
Day
Monday
Tuesday
Starts at
Ends at
Delete

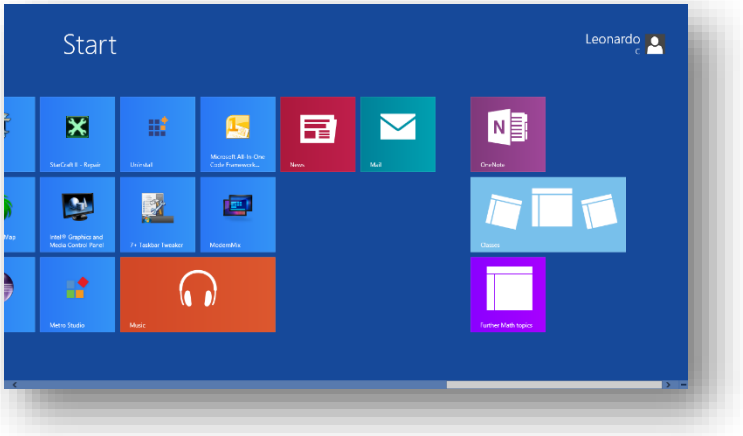
FIGURES 4


Further Math definitions
Mechanics 2 Further Pure 2 Mechanics 3
Motion
Straight line
Vectors and calculus
Projectile motion
Delete
Centre of mass
Studied Revised
Particles
Uniform plane
Composite
Frameworks
Equilibrium
Add Topic Delete
Rigid bodies
Studied Revised
Moments of forces
In equilibrium
Parallel forces
Add Topic Delete
Studied Revised
Add Topic Delete

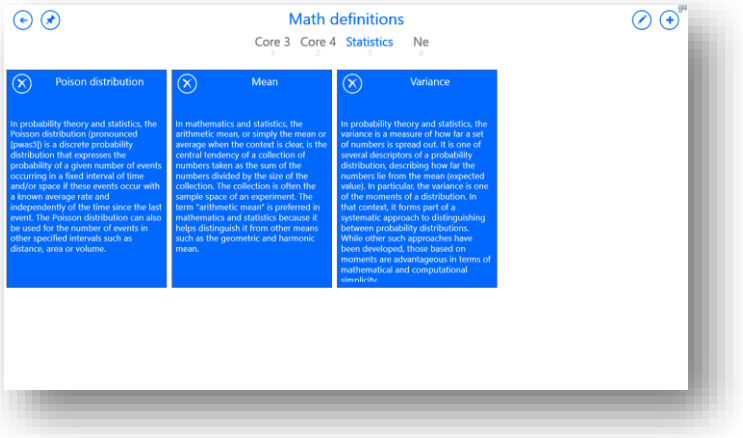

Further Math definitions
Mechanics 2 Further Pure 2 Mechanics 3
Inequalities
Studied Revised
Solving by manipulation
Solving graphically
Add Topic Delete
Complex Numbers
Studied Revised
Euler's relation
De Moivre's theorem
nth root
Add Topic Delete
1st order differentials
Studied Revised
Separate variables
Integrating for x
Substitution
Add Topic Delete

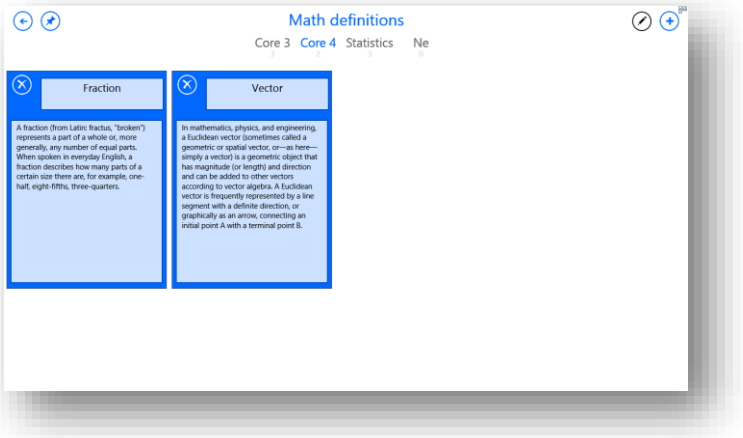
FIGURES 5


Further Math definitions
Mechanics 2
Further Pure 2
Mechanics 3
Complex Numbers
1st order differentials
Studied
Revised
Euler's relation
De Moivre's theorem
nth root
Separate variables
Integrating for x
Substitution
Further Math topics
Pin to Start
Add Topic
Delete
Add Topic
Delete
Add Topic
Delete


Start
Leonardo

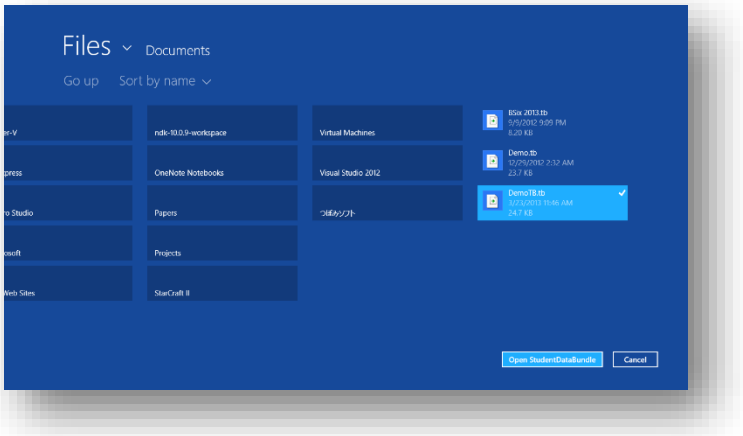
FIGURES 6


Math definitions
Core 3 Core 4 Statistics Ne
Poison distribution
In probability theory and statistics, the Poisson distribution (pronounced [pwas]) is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event. The Poisson distribution can also be used for the number of events in other specified intervals such as distance, area or volume.
Mean
In mathematics and statistics, the arithmetic mean, or simply the mean or average when the context is clear, is the central tendency of a collection of numbers taken as the sum of the numbers divided by the size of the collection. The collection is often the sample space of an experiment. The term "arithmetic mean" is preferred in mathematics and statistics because it helps distinguish it from other means such as the geometric and harmonic mean.
Variance
In probability theory and statistics, the variance is a measure of how far a set of numbers is spread out. It is one of several descriptors of a probability distribution, describing how far the numbers lie from the mean (expected value). In particular, the variance is one of the moments of a distribution. In that context, it forms part of a systematic approach to distinguishing between probability distributions. While other such approaches have been developed, those based on moments are advantageous in terms of mathematical and computational

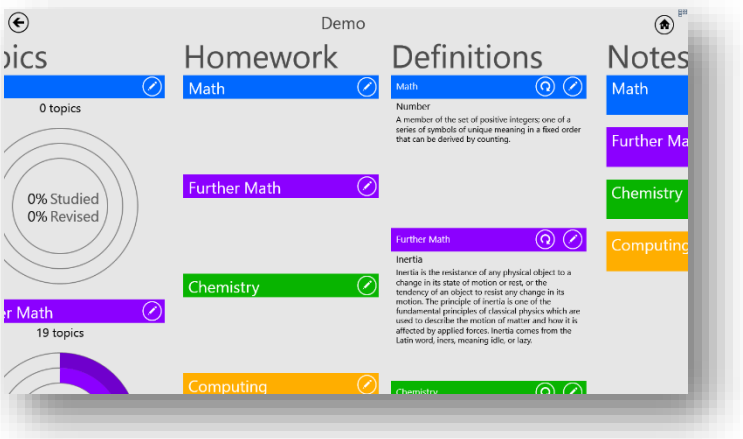

Math definitions
Core 3 Core 4 Statistics Ne
Fraction
A fraction (from Latin: fractus, "broken") represents a part of a whole or, more generally, any number of equal parts. When spoken in everyday English, a fraction describes how many parts of a certain size there are, for example, one-half, eight-fifths, three-quarters.
Vector
In mathematics, physics, and engineering, a Euclidean vector (sometimes called a geometric or spatial vector, or—as here—simply a vector) is a geometric object that has magnitude (or length) and direction and can be added to other vectors according to vector algebra. A Euclidean vector is frequently represented by a line segment with a definite direction, or graphically as an arrow, connecting an initial point A with a terminal point B.

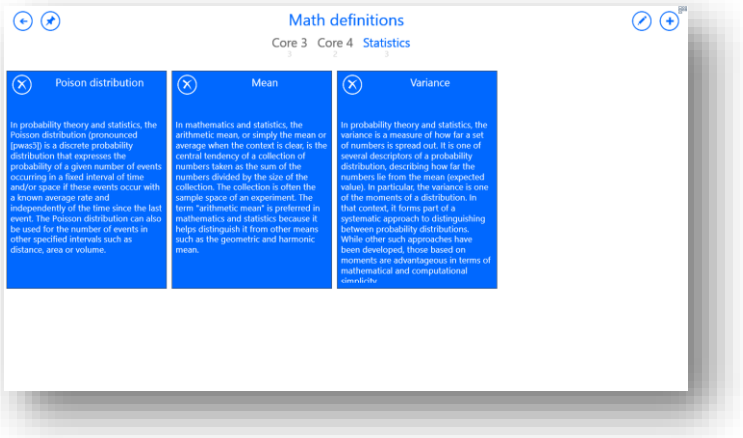
FIGURES 7


Demo
Edit
The purpose of this timetable is to showcase all the features of the application.
Math
Miranda
C101
Further Math
Ahn
C101
EXPORT
OPEN
Add
IMPORT


Files
Documents
Go up
Sort by name
ndk-10.0.9-workspace
Virtual Machines
OneNote Notebooks
Visual Studio 2012
Papers
Projects
StarCraft II
Demo.tb
Open StudentDataBundle
Cancel

FIGURES 8


Demo
ics
0 topics
0% Studied
0% Revised
r Math
19 topics
Homework
Math
Further Math
Chemistry
Computing
Definitions
Math
Number
A member of the set of positive integers: one of a series of symbols of unique meaning in a fixed order that can be derived by counting.
Further Math
Inertia
Inertia is the resistance of any physical object to a change in its state of motion or rest, or the tendency of an object to resist any change in its motion. The principle of inertia is one of the fundamental principles of classical physics which are used to describe the motion of matter and how it is affected by applied forces. Inertia comes from the Latin word, iners, meaning idle, or lazy.
Chemistry
Notes
Math
Further Ma
Chemistry
Computing

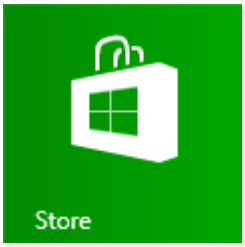

Math definitions
Core 3 Core 4 Statistics
Poison distribution
In probability theory and statistics, the Poisson distribution (pronounced /pwɑːsɔ̃/) is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event. The Poisson distribution can also be used for the number of events in other specified intervals such as distance, area or volume.
Mean
In mathematics and statistics, the arithmetic mean, or simply the mean or average when the context is clear, is the central tendency of a collection of numbers taken as the sum of the numbers divided by the size of the collection. The collection is often the sample space of an experiment. The term "arithmetic mean" is preferred in mathematics and statistics because it helps distinguish it from other means such as the geometric and harmonic mean.
Variance
In probability theory and statistics, the variance is a measure of how far a set of numbers is spread out. It is one of several descriptors of a probability distribution, describing how far the numbers lie from the mean (expected value). In particular, the variance is one of the moments of a distribution. In that context, it forms part of a systematic approach to distinguishing between probability distributions. While other such approaches have been developed, those based on moments are advantageous in terms of mathematical and computational

USER MANUAL

INSTALLATION

This application will run on any PC or tablet running Windows 8 or Windows RT.

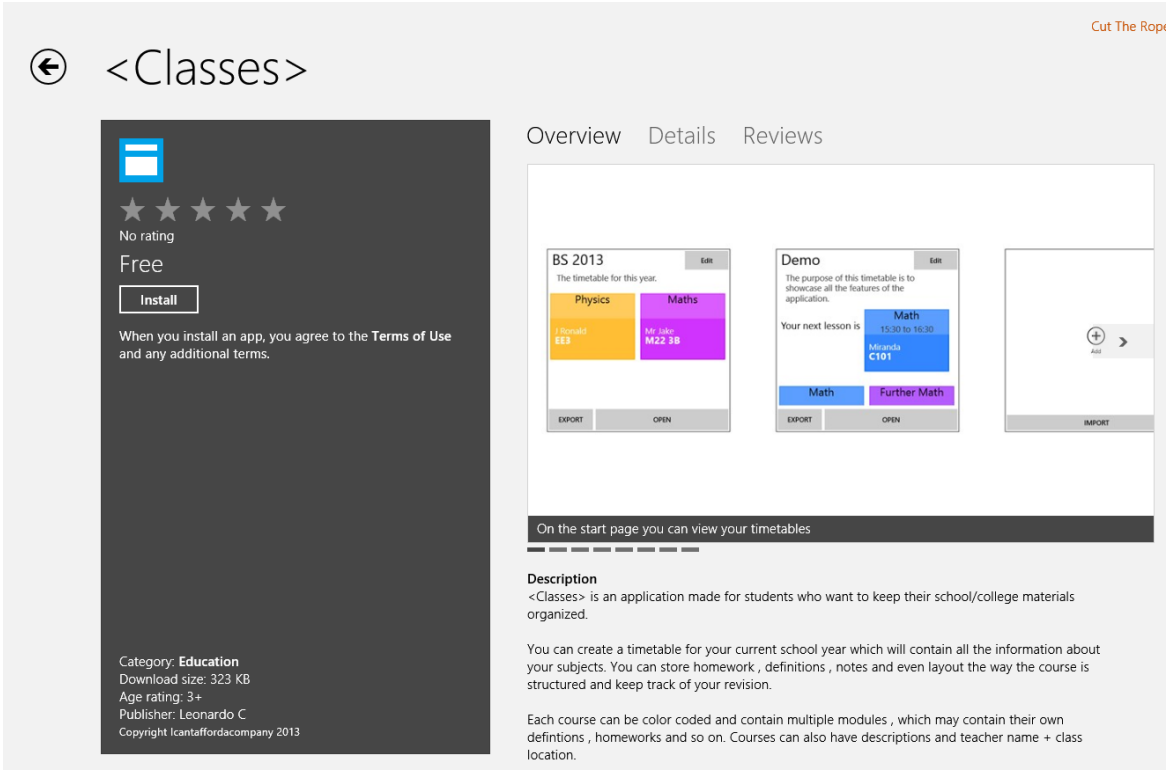
You will need to install the app from the Windows Store. To start , press the Windows key on your keyboard to bring up the start screen (or swipe from left and press windows logo on a tablet). Then press the Windows Store icon:



If you cannot find it on your start screen just type "Store" and it will show up in the search results.

After you have opened the app, start typing Classes and then find the app called <Classes> in the search result.

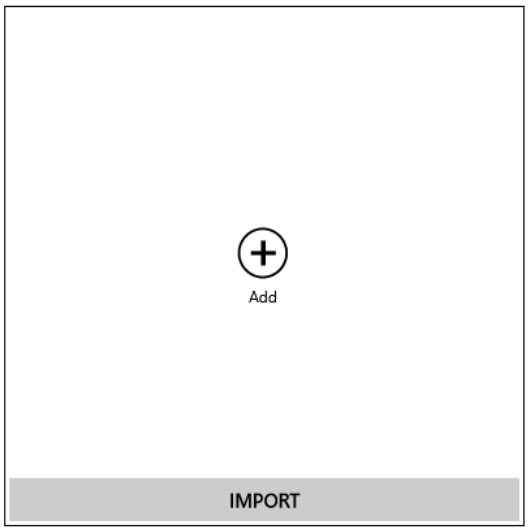
Tap it and you should be brought to a description page - press Install (on the left) to download the app.



CREATING TIMETABLES

After the installation is complete , go to your start screen and press the classes icon:

You must first create a Timetable which will hold all your information. To do this , press the + button in the box:



You will then need to enter a title (under 30 characters) and a description (under 200 characters) for the timetable. When you are done press create in the upper right corner of the box.

USING THE APP

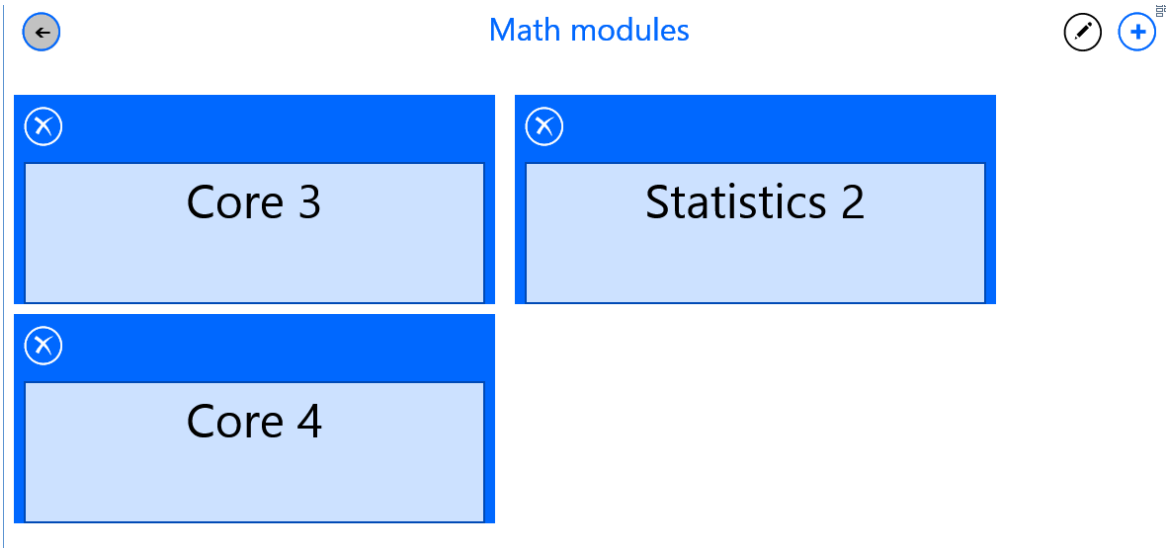
You will now see a new box with the given name. Press Open to start editing.

Before you can add data you will want to fill out your class information. You can do this by pressing the pencil icon next to the heading Classes:

To add a class , tap the + button in the top right corner. A box will show up where you can add your class info. Do this for each of your classes (remember to choose a color for each one). When you are done , press the back button in the top left to go back to the previous screen.

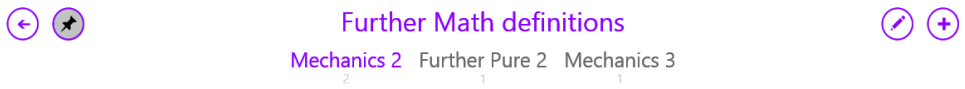
You can now add modules to your classes. For instance your Math class may have C1 , C2 and S1 modules. After you press the back button you will notice you have multiple items under Modules. Just press the pencil (edit) button next to the relevant name.

Press the + button to add a module. To edit the name of the module , press the pencil button next to the + , when you are done editing the text press the pencil button again to stop edit mode.



After filling in your modules , you can start adding notes , topics , homework and definitions.

They all work very similarly , to add a new item - press the + button. To edit the items , press the pencil button. To change the module just tap the module you want in the middle top of the page:



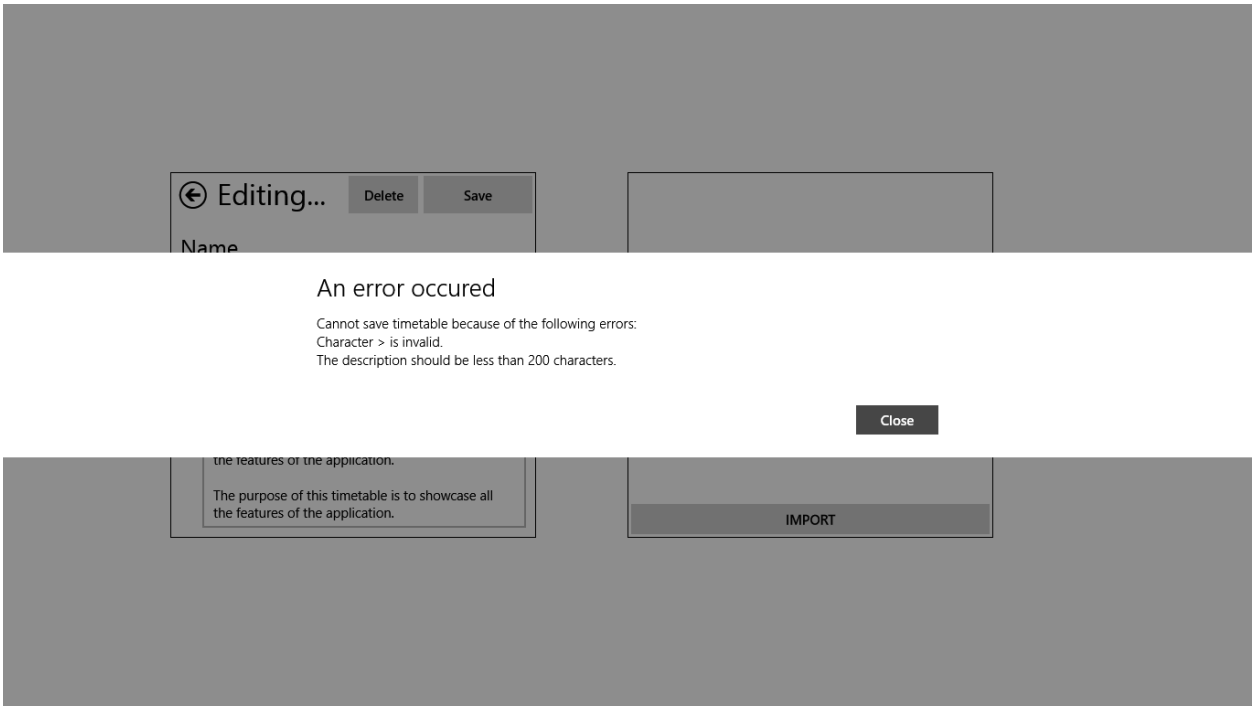
You can also export and import timetables to backup them or share them with your friends. Simply press the export button next to Open and choose a filename. To import , press Import under the Add button.

TROUBLESHOOTING

If you enter something that is too long or does not match the required input you will get a message indicating what you did wrong and in some cases the app will automatically fix it for you.

Keep in mind that the title of the timetable is the name of the file it will be saved in. Therefore you cannot import a timetable with the same name.

If you see a dialog looking like this:



Make sure to read the message or note it down if you need assistance with it.

For any further support, please contact us at:

LEONARDO.CIOCAN@OUTLOOK.COM (DEVELOPER)

CLASSESAPP@BSIX.AC.UK (IT COMPLAINTS & HELP)

APPRAISAL

OBJECTIVES MET:

Provide a way to digitally input a current timetable schedule

The app lets the user choose Saturday or Sunday as a day for one of the course instances (lessons) and the timetable view intelligently shows the weekend only if there are any lessons that day. The user can choose any time between 00:00 to 23:59

Allow the user to schedule lessons in the weekend and have the timetable adjust accordingly

When the user creates an Instance of a course that takes place on Saturday or Sunday, the TimetableView resizes itself to show the weekend as well.

Only show relevant data which is appropriate to the user.

The start page only shows information which may be of use while browsing timetables such as lessons. The main page shows all kinds of information about his items but hides stuff that is empty or unavailable.

Let the user specify modules for each course and aggregate all other information types on them

Each course can have an arbitrary number of modules, the user can easily add modules with the ModulesEditor , the link to which is right next to the course previews on the Main Page.

Allow the user to enhance the visual difference between classes by letting the user assign colors

The user can choose between preset colors that my testers found pleasing to the eye and different enough. The colors are propagated throughout the UI in every page and the majority of controls.

Enable exporting of timetables to XML

The user can press the export button next to the open button in any timetable on the Start Page to open a file saver dialog.

Enable importing of timetables from files

The import button is on the New Timetable Box , the user will be prompted for a file and it will be added to the list of timetables.

Give the user the ability to create homework

This can be done using the HomeworkEditor.

Present the homework with a useful view that aggregates them including due time warnings to indicate the proximity of the date the homework is due on.

The most urgent (closest) 10 homework are shown on the main page of the app for each lesson.

Let the user plan and track his revision based on a table of contents which described the structure of the course

The user can check checkboxes to note whether the topic was studied or revised.

Aggregate notes based on individual modules

A module can have multiple notes (which can have multiple paragraphs)

Let the user add definitions

Definitions can be added via the DefinitionsEditor.

Enable the user to customize the list of courses with attributes like teachers and notes

Inside of the ClassesEditor , the user can use the Class Box of each course to edit it’s properties which include teacher name and notes.

Enable the user to pick lesson times for each lesson.

Instances can be added to each course and they are defined using the start and end time of the lesson

Visualize the user's schedule for the day (i.e next lesson)

On the Start Page, each Timetable Box shows the next lesson , if no lessons are found for the day the control is hidden.

Visualize the user's week schedule using a timetable

The Timetable View shows a visual representation of the user’s timetable and is located at the right of the Main Page.

Show all his timetables and courses.

On the Start Page , the user is shown all the timetables he has and all the courses in each timetable.

ALLOW THE USER TO DELETE AND CREATE NEW TIMETABLES.

The user can do this in the Start Page, deletion is done by pressing the Delete button on the edit part of each timetable (which is accessed by pressing Edit). Adding new timetables is done by pressing the + button in the New Timetable Box.

Let the user customize their notes with formatting

Each paragraph can have its own font size , text location (left or center) , underline and font weight (bold or not).

Show the user a visual indication of where he currently is in the day so he may see upcoming lessons easily.

A red line is drawn at the current day and time on the timetable view.

Enable each timetable to have bundled information about the academy the timetable is related to, either entered by the user or generated by the academy.

This can be done by pressing the Home icon at the top right of the MainPage. An Editor will pop up and let you enter things such as academy name , address and more.

OBJECTIVES NOT MET

Share timetable as picture

API is not currently available with this version of the WinRT API , it will have to wait until an update is made available.

Add definitions to notes.

Removed because the tester suggested it was not useful if it's synchronized with the actual definition (which is not currently feasible since the objects are self-contained and do not have any ID)

Help the users memorize them by using quizzes or similar memory games.

Initial tester was against the prototype version because it was too rigid and checked exact words rather than meaning. Scraped because its addition would be detrimental to the overall project.

USER FEEDBACK

My user tested my app throughout the development process which helped me make some of the major UI and functionality decisions. I have then asked for him to tell me his opinion on whether the app meets the objectives we discussed at the beginning. Here is the reply to my email:

> “
>
> Hi Leonardo,
>
> […]
>
> The application is very useful and I think it will help me a lot, especially next year when I go to university.
>
> I got the application from the Windows Store, which was nice and easy. The first thing I did was making a timetable, this was very intuitive since the + button is self-explanatory. I entered all my courses into the app. I was pleased with the selection of attributes each course could have, colors are very useful and made let me customize the look of the app.
>
> Modules are a really neat way to filter my content which is quite a lot since exams are coming up. I really like the way the application shows information really prominently, including the next lesson on the first screen and the all the snippets of information on the other.
>
> Overall, the app was very intuitive and easy to work with. Sharing via files was useful when the IT administrator gave me a basic template to try, but changing any of the school information was slightly confusing as I did not see the home button on the top right.

There are some things that could be improved, for instance it would be nice to share my stuff with people who do not have the app, i.e. as an image on Facebook.

Also, some other Windows 8 apps can show previews of stuff on the windows 8 start screen – the

“

app could benefit from this.

ANALYSIS OF USER FEEDBACK

It seems the user found the app very user friendly which means the design was effecting.

Regarding his comment on changing college information, the home icon represents the editor used to edit the academy information. This might not be emphasized enough.

IMPROVEMENTS

Here is how I would implement the changes suggested by the user

The home button is not an intuitive way to bring up the academy editor

I would delete the button and then add a pencil edit button right next to the information about the college that would be easier to spot

Sharing as images

This is not natively possible because of the lack of an appropriate rendering API , a fix should be coming soon.

A different, unconventional way would be to draw the XAML view onto a DirectX surface and render that using DirectX calls. That is however outside of my capabilities.

Live tiles to show content on Windows 8 start screen

This could be done using the SecondaryTile class and the scheduled updates functionality in Windows 8. Keeping track of changes in the source would however be slightly more complicated, but not unfeasible.