Howest TSTP app

Documentation

Thijs Windels, Killian Deloof, Jorden Ysewyn

Table of content

Models

Attachment

Name

The Filename of the Attachment

Type

The Content MIME Type of the Attachment in String format

Content

This array of bytes returns the actual content of the Attachment

Building

Address

The full address of the building

UCODE

Contains example RSS.T

UDESC

Contains example rss.t

CCODE

Contains temporary classes

Campus

Contains the full campus where the building belongs to

IMKey

Contains an unique key for a building

Campus

CampusClusterID

Contains the unique ID of a campuscluster

Address

This string contains the full address of a campus

UCODE

Campuscode example campus Buda → BUD

UDESC
Show the fully campusname. Example campus Graaf Karel de Goedelaan

Picture
This will contain a small image (max. 250px x 250px) of a campus

CampusCluster
ID
The unique identity of a campus cluster

Category
SubCategoryId
Contains the unique identity of a subCategorie

Subtitle
Contains the subtitle for a category

IsLocationRequired
Tells whether a location is needed or not in order to pick this category

IsStaffRequired
Tells whether this topics is for teachers/staff only (Not used in current state)

Floor
FloorID
Contains the unique identity of a floor

BuildingID
Contains the unique identity of the building where the floor is situated

WingID
Contains the unique identity of the wing where the floor is situated

Forum
ForumId
The unique identity of a forum

MainCategory

CategoryId
Contains the unique identity of Category

Picture
Contains a small icon for the main category

SubCategoryList
Contains a list of subcategories corresponding to the main category.

Repositories

APIRepository
This repository services as a bridge between the Howest API (made by Jeff Daels Hades) and the application

GetBuildingList
This task returns a list of all buildings provided by the API.

GetCampusList
This task returns a list of all campuses provided by the API.

GetWingList
This task returns a list of all wing provided by the API

GetFloorList
This task returns a list of all floors provided by the API

GetRoomList
This task returns a list of all rooms provided by the API

GetForumList
This task returns a list of forum objects provided by the API

GetHardCodedList
This list returns all main categories. This list is needed since the API doesn’t provide main categories.

DatabaseRepos
This repo governs the transactions between the SQLlite database and the app. This repo expects a constructor with the name of a database as parameter (tstp)

DataBaseRepos
This methode makes a connection with the database.

CreateTable
Creates a table in the database , if it doesn't already exist

SaveItem
Updates an item to the database, or inserts it if it wasn't in the database yet.
If the transaction was successful it returns 1.

ExecuteQuery
Executes a prepared query that doesn't return results (ex. INSERT, DELETE, UPDATE).
This methods expects 2 arguments. The first expects the statement as a string. The second one expects an array of arguments. When no arguments are needed, this must be null

Query
Executes a SQL query and returns the results, this method contains two arguments. The first one needs a fully escaped SQL statement. The second argument are the arguments to substitude, the occurences of "?" in the query.

GetItems
Gets The objects from a table
returns an IEnumerable of the given ObjectType

DeleteItem
Deletes an item from the database. Expects an id as argument.
If it succeeds, it returns 1.

DeleteAll
Deletes all entries From the given Table in the database. It returns 1 once it’s succesfull.

GPSRepository
This repository manages all geolocation data. This repository uses the Geolocator plugin.

GetLocation
Gets current coordinates from GPS, returns an array of double: value[0] = latitude and value[1] = longitude

GetCampusDistances
Fills in the distance property of campus items in the given List

CalculateDistance
Calculates the distance between 2 coordinates. Expects the coordinates of a first point as first argument and a second coordinate as second argument.

GetClosestCampus
This method returns the closest campus at your current location. It expects a list of the campuses, your current location and the minimum distance the user needs to be to a campus.

LoginRepository
This interface manage everything needed to login

Login
Opens a new screen and authenticates the user.

MediaPicker
This repository helps to add media files to the attachment of the ticket.

TakePhoto
This photo opens a camera application installed on the device

PickPhoto
Select a picture via a gallery app installed on the device

MediaFileToByteArr
Converts a receive file to a byte array.

Interfaces

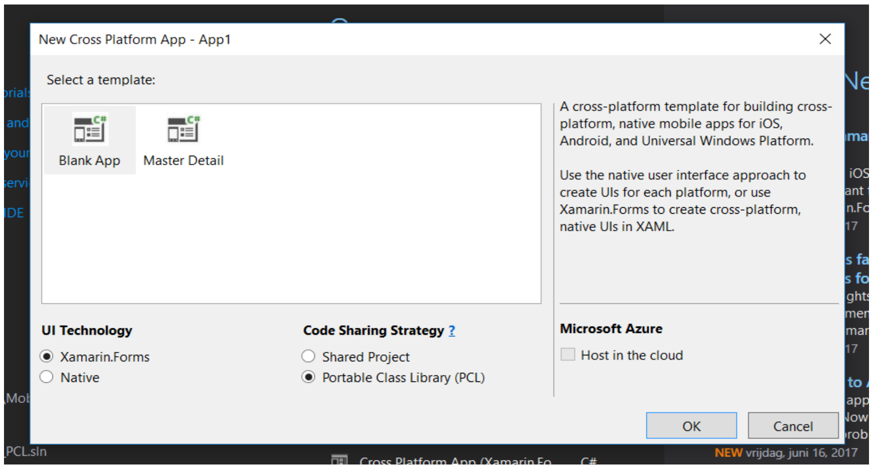
ISQLite
This interface manage the local SQLite database inside the app

The setup

To build a Xamarin Cross platform application, search for Xamarin, and choose for “Cross platform app (Xamarin)”. Ensure you check the next radio buttons before continuing:

- Xamarin.Forms

- Portable Class Library (PCL)



New Cross Platform App - App1

Select a template:

A cross-platform template for building cross-platform, native mobile apps for iOS, Android, and Universal Windows Platform.

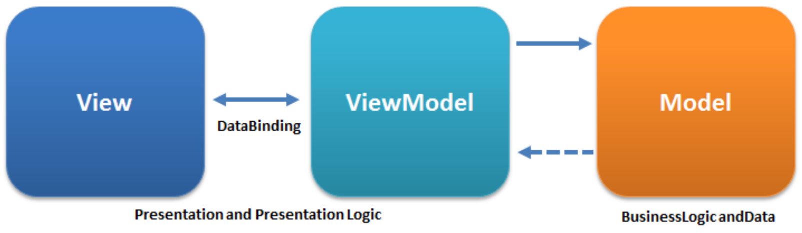
Use the native user interface approach to create UIs for each platform, or use Xamarin.Forms to create cross-platform, native UIs in XAML.

UI Technology
[✓ Xamarin.Forms] Native

Code Sharing Strategy ?
Shared Project [✓ Portable Class Library (PCL)]

Microsoft Azure
Host in the cloud

First thing first, when the solution is ready, before writing any code, ensure you **update** all the Nuget packages. After updating the Nuget packages, go to build and **clean** your solution. When Visual Studio is done cleaning, choose to **rebuild** the solution. This steps are mandatory.



When previous steps are done correctly we will setup or project environment. In this environment, we need to respect the MVVM rules, we will separate models, the code behind and the view from each other.

Right click on the solution and choose for add → new project and choose for a "Class Library (C#)". In this library, all models will be made.

The solution should have 5 project right now:

- A shared project (standard project)
- An Android Project
- An IOS project
- An UWP project
- The models class library

Be sure to add a **reference** to the models class library in every project in this solution.

In the shared project, make 3 folders, one for the Views, one for the ViewModels and one for repositories.

In the View, we make all the pages for the application. In order to create a view, right click on the View folder and select add. Choose for a **ContentPage** (c#). You can design the page using xaml code. To add an event to a control we use binding. The **event** will be described and executed in the ViewModel of the corresponding View. To create a ViewModel right click on the folder ViewModel and choose add. Select a class. This new class needs to implement the interface **INotifyPropertyChanged**.

Now we will prepare all the solutions to load our created page(s). Open your shared project (standard) and navigate to the App.xaml.cs. Change the code After InitializeComponent(); to this:

```
MainPage = new NavigationPage(new LoginPage()).
```

Now the MainPage will load your page (LoginPage in this example) instead of the default page.

List of additional Packages

- Microsoft.Azure.Mobile.Client
- Newtonsoft.Json
- sqlite-net-pcl (Frank A.Krueger)
- Xam.Plugin.Geolocator
- Xam.Plugin.media
- Xamarin.CustomControls.StateButton
- Xamarin.CustomControls.AccordionView