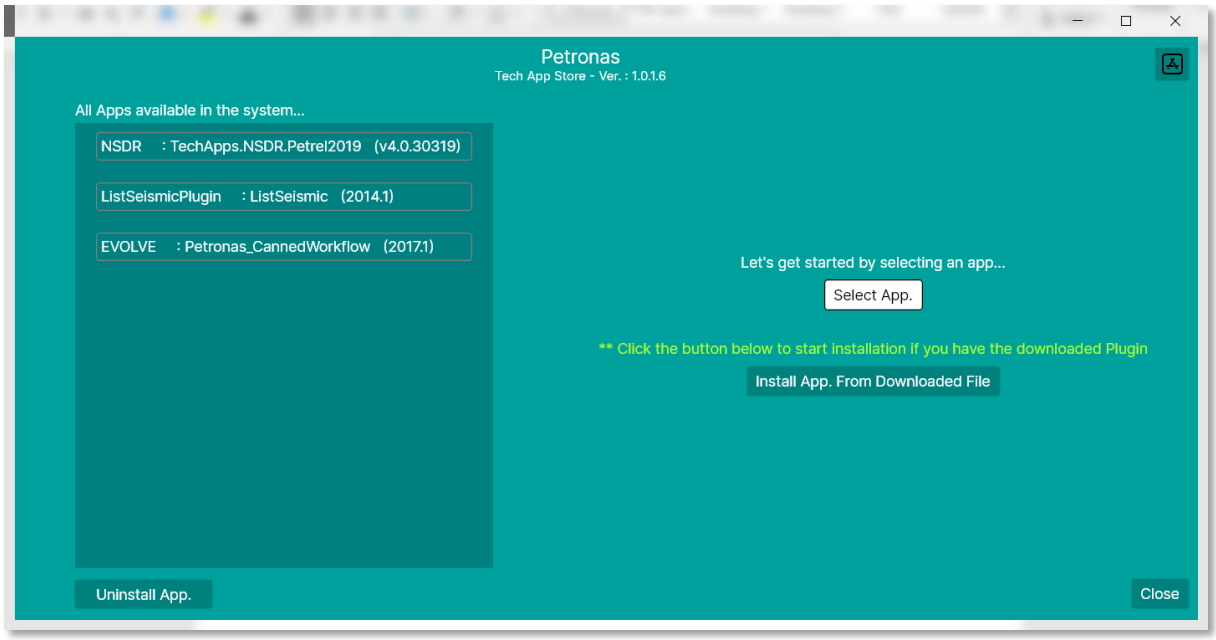
TechAppLauncher

Application Name	TechAppLauncher
Application Framework	.Net 5.0
UI Framework	Avalonia
IDE	Visual Studio 2019
Repository	https://github.com/azzulhisham/Petronas_TechAppLauncher.git
Application Url	N/A – Standalone Desktop Application
Host	Software Center

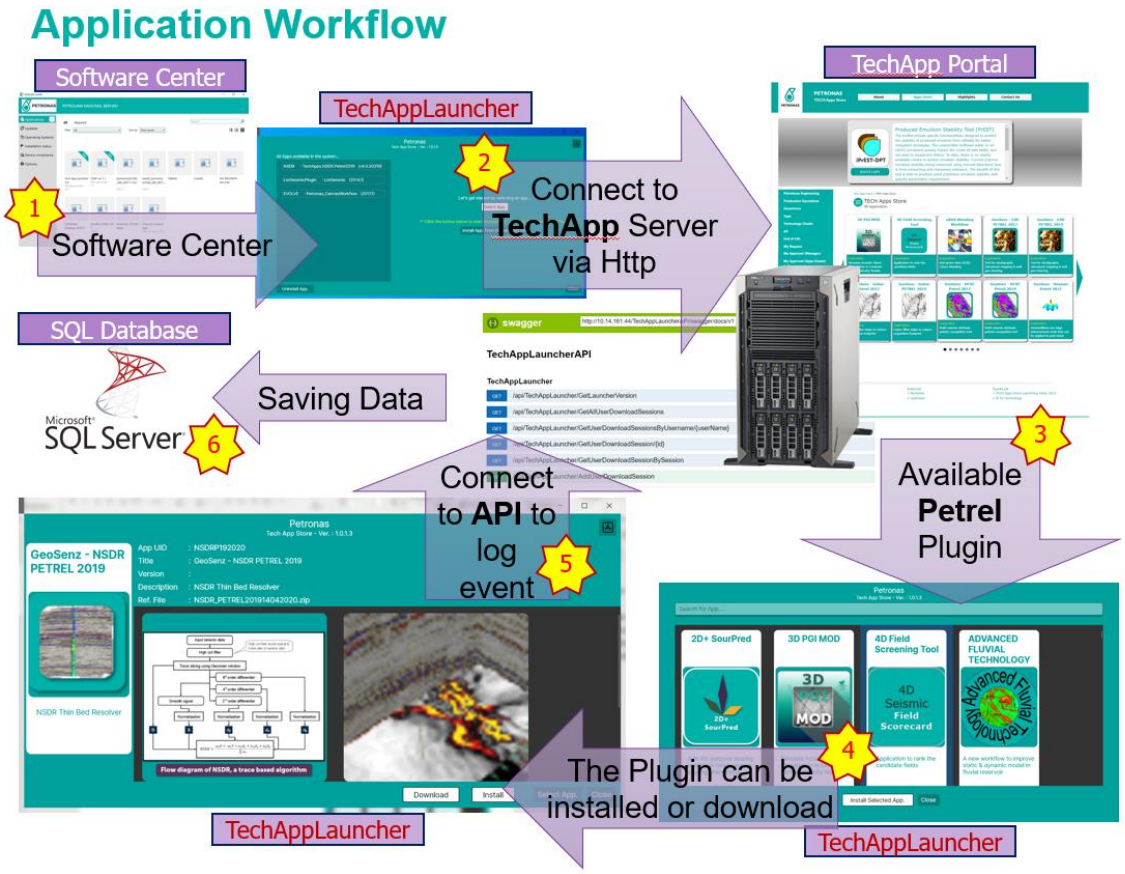
Overview



TechAppLauncher is a standalone desktop application that engaging the **Avalonia UI framework**. The Application is hosted at the **Software Center** in order to distribute to the client.

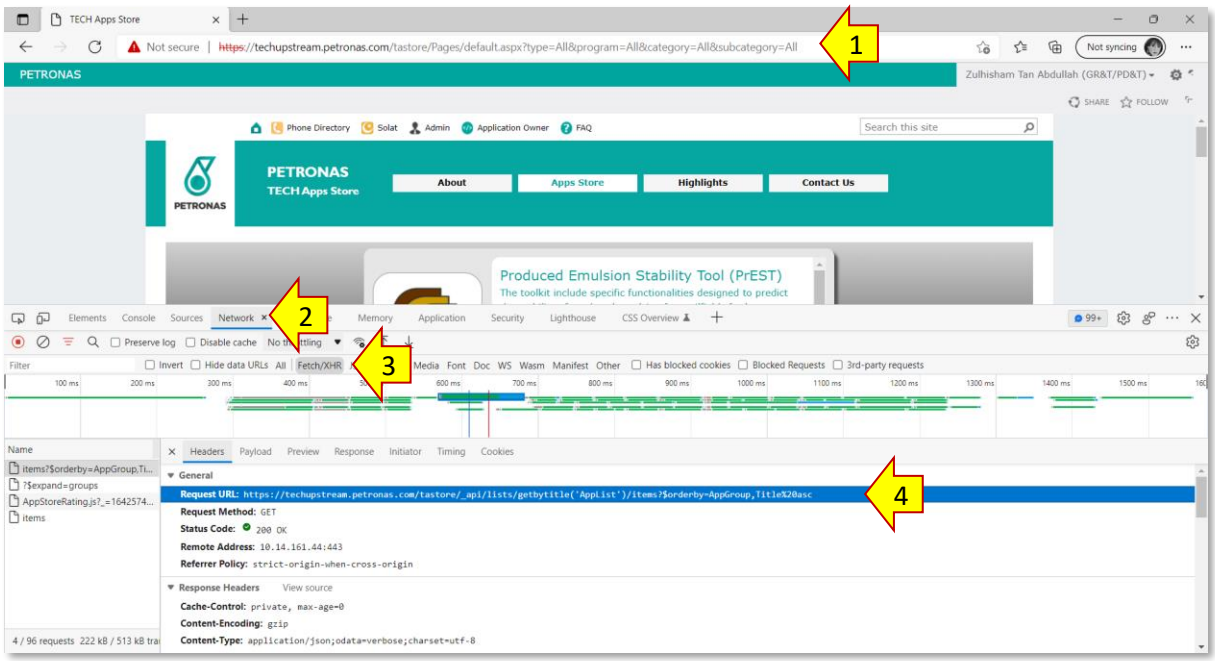
Click the link Basics - Avalonia (avaloniaui.net) in order to learn more about Avalonia. You can follow the tutorial provides by the home page of Avalonia in order to understand how the application works in details.

The entire flow of the application is describes in the next page.

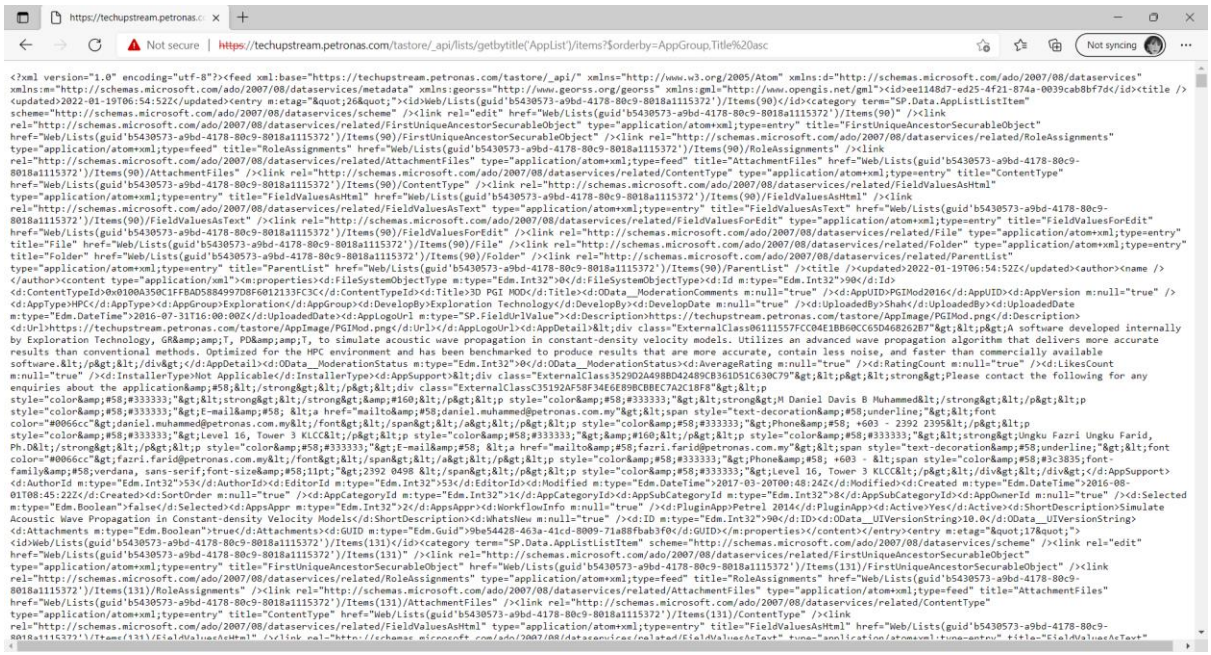


1. The application is hosted in the **Software Center**, hence user need to be installed the application from the **Software Center**.
2. The application is connected to the **TechApp Store Portal** to retrieve all the available app in the **TechApp store**.
3. The application list all the available application which is typically identical to the **TechApp Store portal**.
4. The application provides the **download** feature as well as **install** feature to the client.
5. Whenever user successfully download or install a TechApp's application, it will then connect to its **API**.
6. The application updates necessary **download session information** through its API.

Using Browser to track important endpoint



1. Browser to the **TechApp Store Portal** and then press '**F12**' to enter to the developer tools.
2. In the Developer tool, select '**Network**' tab.
3. Switch to '**Fetch/XHR**' mode and then refresh the page.
4. The endpoint shown in the '**Headers**' tab is used by the **TechAppLauncher** in order to retrieve all the available app in **TecApp Store**. You can copy the URL of the endpoint and then paste it to your browser, the screenshot below shows the response from the server in xml format.



In order to used these data in C# code, first select all the text which response from the server in the browser, and then switch Visual Studio 2019, add a new class file to the desire folder that you wish, make sure the new added class file is being selected. Then, in the menu, select '**Edit**' >> '**Paste**

Special' >> '**Paste XML as Classes**'. Since the data return from the server is in **XML** format, we need to engage the '**Paste XML as Classes**' to help us convert the data format into C# entity. If you are pro in XML then you may skip this step. The generate C# class is actually serves as an temporary object to keep the XML data.

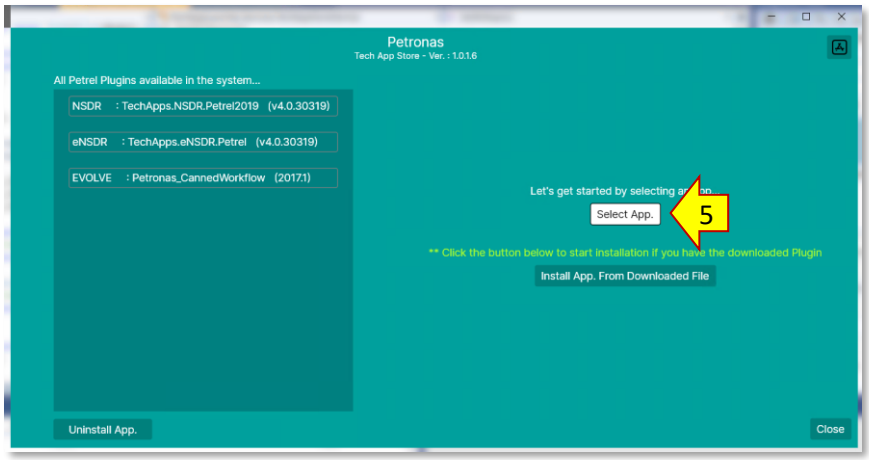
```
TechAppLauncher - TechAppStoreService.cs*
TechAppStoreService.cs*
TechAppLauncher | TechAppLauncher.Services.TechAppStoreService | GetAllAsync()
46      public async Task<IList<Models.App>> GetAllAsync()
47      {
48          string url = @"https://techupstream.petronas.com/tastore/_api/lists/getbytitle('AppList')/items?$orderby=AppGroup,Titl
49          var request = new HttpRequestMessage()
50          {
51              RequestUri = new Uri(url),
52              Method = HttpMethod.Get,
53          };
54
55          request.Headers.Accept.Clear();
56          request.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
57
58          s_httpClient = new HttpClient(_handler);
59          s_httpClient.DefaultRequestHeaders.Accept.Clear();
60          s_httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
61
62          var response = await s_httpClient.SendAsync(request, HttpCompletionOption.ResponseContentRead);
63          //var content = await response.Content.ReadFromJsonAsync(typeof(Models.AppDetail));
64          var content = await response.Content.ReadAsStringAsync();
65
66          Serializer serializer = new Serializer();
67          //var feeds = serializer.DeserializeFeed(content);
68          var feeds = serializer.Deserialize<Models.xml.AppList.feed>(content);
69
70
71          var results = feeds.entry.Select(s => new Models.App()
72          {
73              AppUID = s.content.properties.AppUID != null ? s.content.properties.AppUID.Value : "",
74              AppType = s.content.properties.AppType != null ? s.content.properties.AppType : "",
75              PluginApp = s.content.properties.PluginApp != null ? s.content.properties.PluginApp.Value : "",
76              ID = s.content.properties.ID != null ? s.content.properties.ID.Value : 0,
77              Title = s.content.properties.Title != null ? s.content.properties.Title.Value : "",
78              ShortDescription = s.content.properties.ShortDescription != null ? s.content.properties.ShortDescription.Value : "
79              AppLogoUrl = new Models.AppLogoUrl()
80              {
81                  Url = s.content.properties.AppLogoUrl != null && s.content.properties.AppLogoUrl.Url != null ? s.content.prope
82              }
83          }).ToList();
84
85          return results;
86      }
```

If we look t the code, line 48 to line 62 is actually send out the http request to the server. The data came from the server is then read as a string (which is XML) and keep in variable '**content**', this is in line 64.

In line 68, the content in variable '**content**' is then deserialize into C# object '**feeds**'. Line 71 to line 83, it picks up those necessary data as '**result**' object and send back to the caller as a list in line 85.

The function is call whenever user click the '**Select App.**' button[5].

The screenshot below shows the code where the function is being called (line 136).



```
TechAppLauncher - AppStoreViewModel.cs
AppStoreViewModel.cs
TechAppLauncher | TechAppLauncher.ViewModels.AppStoreViev | ListAllApp()
125         }
126
            1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
127         private async void ListAllApp()
128         {
129             _cancellationTokenSource?.Cancel();
130             _cancellationTokenSource = new CancellationTokenSource();
131
132             try
133             {
134                 //Call API
135                 ITechAppStoreNetworkRequestService techAppStoreService = new TechAppStoreService();
136                 _apps = await techAppStoreService.GetAllAsync();
137                 await LoadAppListView(_apps);
138             }
139             catch (Exception ex)
140             {
141
142             }
143         }
144
100 %   0   5   Ln: 137   Ch: 1   SPC   LF
```

The '**ListAllApp()**' function is actually being call from the class's constructor.

```
TechAppLauncher - AppStoreViewModel.cs
AppStoreViewModel.cs
TechAppLauncher | TechAppLauncher.ViewModels.AppStoreViev | AppStoreViewModel()
            1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
76          public AppStoreViewModel()
77          {
78              var assemblyVersion = Assembly.GetExecutingAssembly().GetName().Version;
79              AppTitleBar = $"Tech App Store - Ver. : {assemblyVersion.Major}.{assemblyVersion.MajorRev
80
81              this.WhenAnyValue(x => x.SearchText)
82                  .Throttle(TimeSpan.FromMilliseconds(400))
83                  .ObserveOn(RxApp.MainThreadScheduler)
84                  .Subscribe(DoSearch!);
85
86
87              GetAppSelectCommand = ReactiveCommand.Create(() =>
88              {
89                  return SelectedApp;
90              });
91
92              GetAppSelectCommandClose = ReactiveCommand.Create(() =>
93              {
94                  SelectedApp = null;
95                  return SelectedApp;
96              });
97
98              ListAllApp();
99          }
100 %   0   5   Ln: 99   Ch: 1   SPC   LF
```

The '**AppStoreViewModel.cs**' is a view model for '**AppStoreView**' interface.

```
194         public MainWindowViewModel()
195         {
196             var assemblyVersion = Assembly.GetExecutingAssembly().GetName().Version;
197             AppTitleBar = $"Tech App Store - Ver. : {assemblyVersion.Major}.{assemblyVersion.MajorRevision}.{assemblyVersion.Bu
198
199             _techAppStoreService = new TechAppStoreService();
200             ShowAppDialog = new Interaction<AppStoreViewModel, AppViewModel?>();
201             ShowMsgDialog = new Interaction<MessageDialogViewModel, MessageDialogViewModel>();
202             ShowRemoveAppDialog = new Interaction<RemoveAppViewModel, RemoveAppViewModel>();
203
204             SelectAppCommand = ReactiveCommand.CreateFromTask(async () =>
205             {
206                 var versionControl = await _techAppStoreService.GetLauncherVersion();
207
208                 if (versionControl != null)
209                 {
210                     if (assemblyVersion.Major < versionControl.Major || assemblyVersion.MajorRevision < versionControl.MajorRev
211                         assemblyVersion.Build < versionControl.Minor || assemblyVersion.Revision < versionControl.MinorRevisio
212                     {
213                         string messageBoxText = "There is a newer version available.\r\nKindly update your app before start.";
214                         var messageBoxDialog = new MessageDialogViewModel(messageBoxText, Enums.MessageBoxStyle.IconStyle.Warni
215                         await ShowMsgDialog.Handle(messageBoxDialog);
216
217                         return;
218                     }
219                 }
220                 else
221                 {
222                     string messageBoxText = "Your system is not in the correct network.\r\nThis app will not work correctly.";
223                     var messageBoxDialog = new MessageDialogViewModel(messageBoxText, Enums.MessageBoxStyle.IconStyle.Warning);
224                     await ShowMsgDialog.Handle(messageBoxDialog);
225                 }
226
227                 var store = new AppStoreViewModel();
228                 var result = await ShowAppDialog.Handle(store);
229
230                 if (_refFileDetails == null)
```

Now, here is the magic happened in **Avalonia**. From the screenshot above, line 204, this is where the command is being fired when the '**Select App.**' Button is pressed. It shows the dialogue of **AppStoreView**. Whenever the user select an app to install, the **AppStoreView** will returns a **result** to the main window which is in line 228 as a **AppViewModel** defined in line 200. The AppViewModel contain all the information about the selected app.

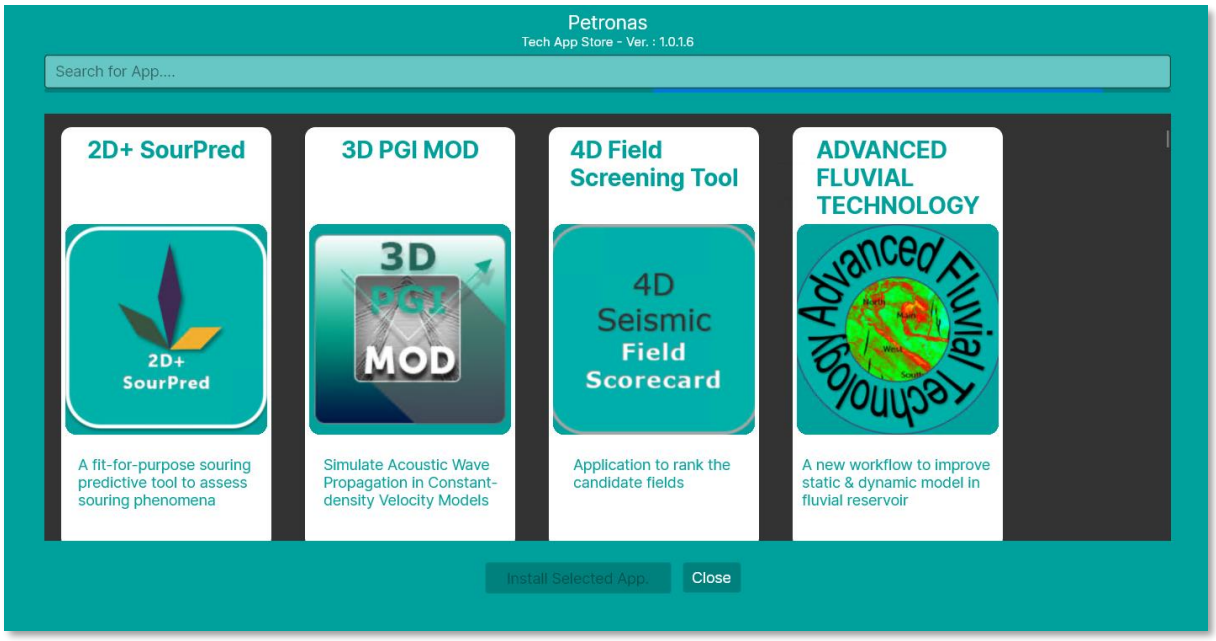
TechAppLauncher - AppViewModel.cs

AppViewModel.cs

TechAppLauncher | TechAppLauncher.ViewModels.AppViewModel | _app

```
17  1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
    public AppViewModel(Models.App app)
18  {
19      _app = app;
20  }
21
    1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
22  public string Description => _app.ShortDescription;
23
    1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
24  public string Title => _app.Title;
25
    0 references | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
26  public string AppGroup => _app.AppGroup;
27
    2 references | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
28  public int AppId => _app.ID;
29
    4 references | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
30  public string AppUID => _app.AppUID;
    1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
31  public string AppType => _app.AppType;
    1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
32  public string PluginApp => _app.PluginApp;
33
    2 references | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
34  public double? AppVersion => _app.AppVersion;
35
36  private Bitmap? _appImg;
37
    1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
38  public Bitmap? AppImg
39  {
40      get => _appImg;
41      private set => this.RaiseAndSetIfChanged(ref _appImg, value);
42  }
43
    1 reference | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
44  public async Task LoadAppImage()
```

100 % | 0 | 1 | Ln: 13 | Ch: 30 | SPC | LF



While preparing the interaction user-interface as shown above, the TechAppLauncher make another http call to the Server (line 232 in the following screenshot).

```
TechAppLauncher - MainWindowViewModel.cs
MainWindowViewModel.cs
TechAppLauncher | TechAppLauncher.ViewModels.MainWindowViewModel | MainWindowViewModel()
227            var store = new AppStoreViewModel();
228            var result = await ShowAppDialog.Handle(store);
229
230            if (_refFileDetails == null)
231            {
232                _refFileDetails = await _techAppStoreService.GetAllRefFilesAsync();
233            }
234
235            bool isLaunchAble = false;
236            this.IsLaunchAble = isLaunchAble;
237            this.IsDownloadAble = false;
238
239            Apps.Clear();
240
241            if (result != null && _refFileDetails != null)
242            {
243                Apps.Add(result);
244
245                var refFileDetailSelect = _refFileDetails.Where(n => n.AppUID == result.AppId.ToString()).FirstOrDefault()
246                var refFileUrl = refFileDetailSelect != null ? refFileDetailSelect.FileRefUrl : null;
247
100 %   0   23   Ln: 232   Ch: 85   SPC   CRLF
```

This is to get all the zip file that attached to the specific application based on **AppUID**. Notice that it is picking up all the necessary data (line 159 to line 162) such as AppUID, File URL, Title of the app and etc…

```
TechAppLauncher - TechAppStoreService.cs
TechAppStoreService.cs
TechAppLauncher | TechAppLauncher.Services.TechAppStoreService | GetAllRefFilesAsync()
2 references | Zulhisham Tan Abdullah, 7 days ago | 1 author, 1 change
132        public async Task<IList<RefFileDetail>> GetAllRefFilesAsync()
133        {
134            string url = @"https://techupstream.petronas.com/tastore/_api/lists/getbytitle('AppVault')/items";
135            var request = new HttpRequestMessage()
136            {
137                RequestUri = new Uri(url),
138                Method = HttpMethod.Get,
139            };
140
141            request.Headers.Accept.Clear();
142            request.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
143
144            s_httpClient = new HttpClient(_handler);
145            s_httpClient.DefaultRequestHeaders.Accept.Clear();
146            s_httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
147
148            try
149            {
150                var response = await s_httpClient.SendAsync(request, HttpCompletionOption.ResponseContentRead);
151                var content = await response.Content.ReadAsStringAsync();
152
153                Serializer serializer = new Serializer();
154                var feeds = serializer.Deserialize<Models.xml.AppRefFile.feed>(content);
155
156
157                var results = feeds.entry.Select(s => new RefFileDetail()
158                {
159                    AppUID = s.content.properties.AppUIDId != null ? s.content.properties.AppUIDId.Value : "",
160                    Title = s.content.properties.Title != null ? s.content.properties.Title.Value : "",
161                    FileID = s.content.properties.ID?.Value,
162                    FileRefUrl = feeds.@base + s.id.Trim() + "/File"
163                }).ToList();
164
165                return results;
166            }
167            catch (Exception ex)
168            {
100 %   0   30   Ln: 132   Ch: 68   SPC   CRLF
```

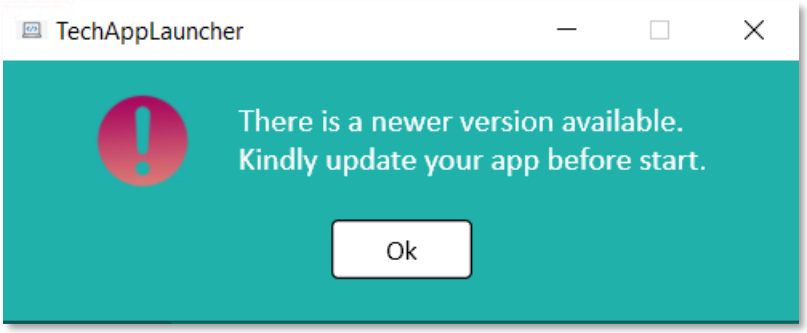
Hence, based of the file URL, the TechAppLauncher knows where to download the attachment files.

TechAppLauncherAPI

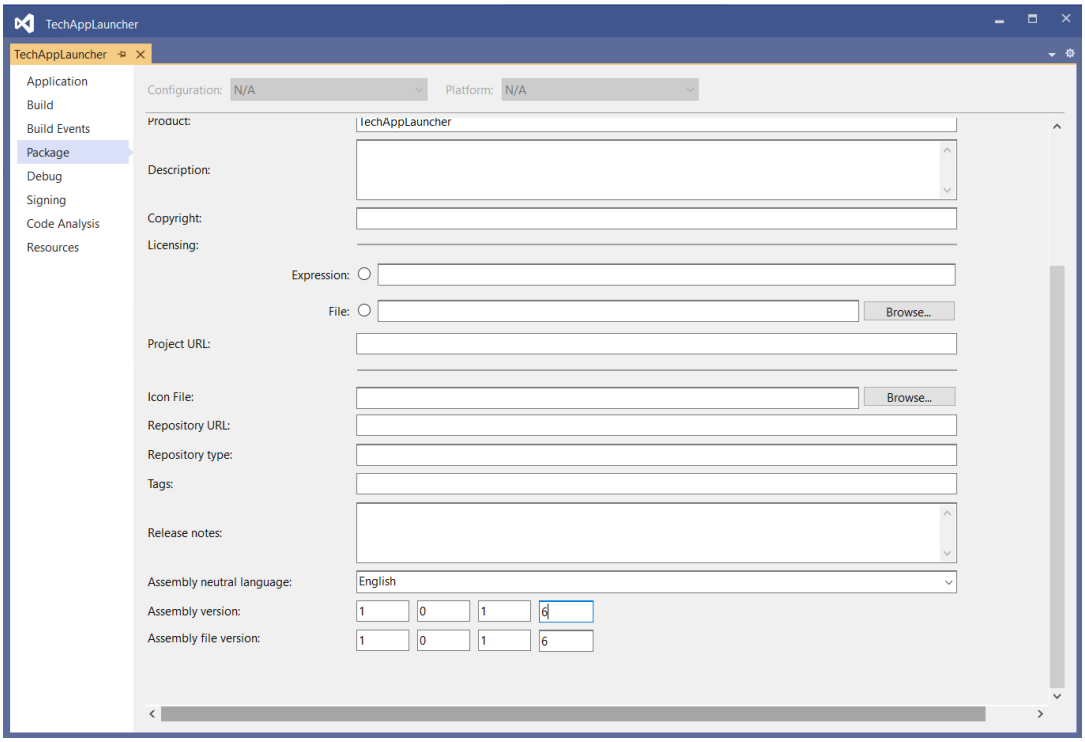
There are few important endpoints that the **TechAppLauncher** connected to:

```
1. /api/TechAppLauncher/GetLauncherVersion
2. /api/TechAppLauncher/GetlauncherAppConfig
3. /api/TechAppLauncher/GetUserDownloadSessionsByUsername
4. /api/TechAppLauncher/AddUserDownloadSession
5. /api/TechAppLauncher/GetAppDistributionReferenceDetailByAppUID
```

1. **GetLauncherVersion** – The launcher uses this information to determine whether the client is using the latest version of the application, otherwise it will prompt the user with the message as shown below.



The launcher version number is set at it's project property as shown below. Hence, it is recommended to increase the number every time there is a new update deploy to the server.



You can use the following endpoints to update the version number.

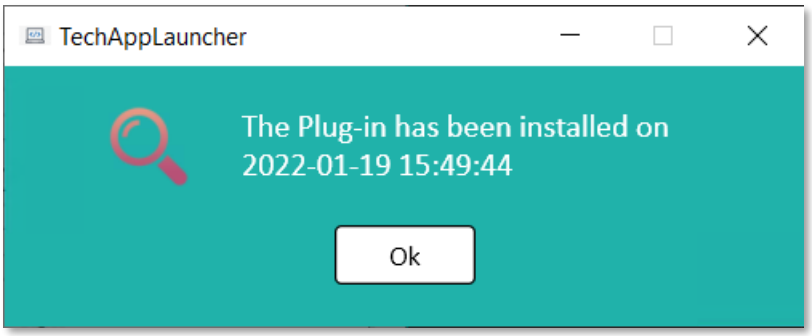
```
 i.   /api/TechAppLauncher/UpdateAppVerMajorNumber
ii.   /api/TechAppLauncher/UpdateAppVerMajorRevisionNumber
iii.  /api/TechAppLauncher/UpdateAppVerMinorNumber
iv.   /api/TechAppLauncher/UpdateAppVerMinorRevisionNumber
```

2. **GetlauncherAppConfig** – The launcher uses this endpoint to retrieve the **TechApp Store** server credentials.

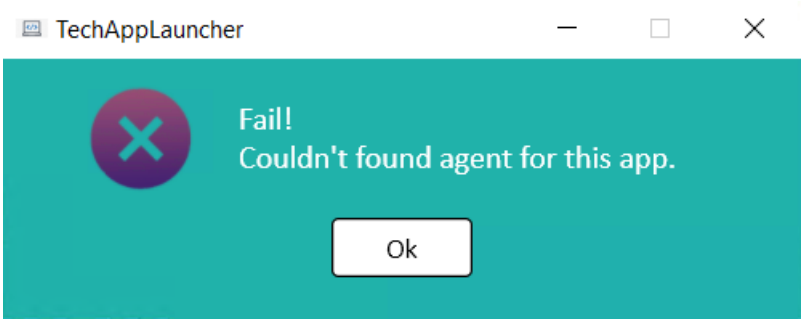
   Since the server credentials might need to update in a period of time, you can update the credential information with the following endpoints.

```
 i.   /api/TechAppLauncher/UpdateAppStoreServerDomainName
ii.   /api/TechAppLauncher/UpdateAppStoreServerUserName
iii.  /api/TechAppLauncher/UpdateAppStoreServerPassword
```

3. **GetUserDownloadSessionsByUsername** – The launcher uses this endpoint to determine whether a specific plugin had already installed at the client workstation. Its will prompt the message as shown below if that specific plugin had already installed.



4. **AddUserDownloadSession** – The launcher uses this endpoint to insert new record of User Download Session into the database.
5. **GetAppDistributionReferenceDetailByAppUID** – The launcher uses this endpoint to find for the installation agent for a specific application such as **Software Center**. The launcher will prompt an error message if you try to install a standalone application but there is no information has been setup in the database.



You can use the following endpoint to add and to delete a specific agent.

```
 i.   /api/TechAppLauncher/AddAppDistributionReferenceDetail
ii.   /api/TechAppLauncher/DeleteAppDistributionReferenceDetail
```

You can also update the details of a specific agent with the following endpoints.

```
 i.   /api/TechAppLauncher/UpdateAppDistributionReferenceDetailByApp
      UID
ii.   /api/TechAppLauncher/UpdateAppDistributionReferenceDetailById
```