Innovationn Project – ISS

To set an example for modern facilities and innovative facility solutions in their new headquarters in Germany ISS Facility Services is building a solution to identify registered visitors, greet them personally and direct them to their destination in the office building. The solution is a kiosk application for Windows 10 that uses Microsoft Cognitive Services to identify faces of returning visitors and employees. The solution lets you manage the database of people that can be identified and lets you add new faces the service can be trained with. Eventually the application will run in Kiosk mode to identify visitors and greet visitors as well as let people add themselves to the database.

Key technologies:

- Universal Windows Platform
- Microsoft Cognitive Services Face API
- Microsoft Visual Studio 2017

Core Team:

- Kevin Rottsieper - Software Engineer, ISS Facility Services Holding GmbH
- Daniel Wittkuhn - Teamlead ICT Application Development, ISS Facility Services Holding GmbH
- Karsten Mostersteg - Lead ICT, ISS Facility Services Holding GmbH
- Malte Lantin - Technical Evangelist, Microsoft Deutschland GmbH

Customer profile

With almost 500.000 employees worldwide ISS has grown to one of the leading facility management service companies worldwide. With its headquarter in Denmark and subsidiaries in 75 countries they serve customers in the fields facility services, cleaning services, technical services, catering services and more. They entered the German marked in 1960 and currently build their new German headquarter in Düsseldorf. They aspire to drive innovation in their services and build their new headquarter as an example of what is possible. They want to make visiting the new headquarter more personal by using technology to identify registered visitors, greet them personally and direct them to their destination in the office building.

Problem statement

To deliver a personal experience and accelerate the registration process for visitors at the reception desk ISS plans to set up kiosk PC at their reception that will be identifying visitors who have given their consent and registered for this service before. The visitors are supposed to be greeted personally by the reception desk staff and then redirected to their host or meeting room.

As face recognition is a complex process they were looking for a solution that would allow them to built such a system in a short period of time as well as at low cost. The application is supposed to run on a kiosk PC as well as on the PC the reception staff is already using. Ideally the solution would work with commodity hardware and cameras.

Solution and steps

Together with Microsoft a small team at ISS started building a solution based on the Windows Universal Platform (UWP). Using the Universal Windows Platform has the advantage that the

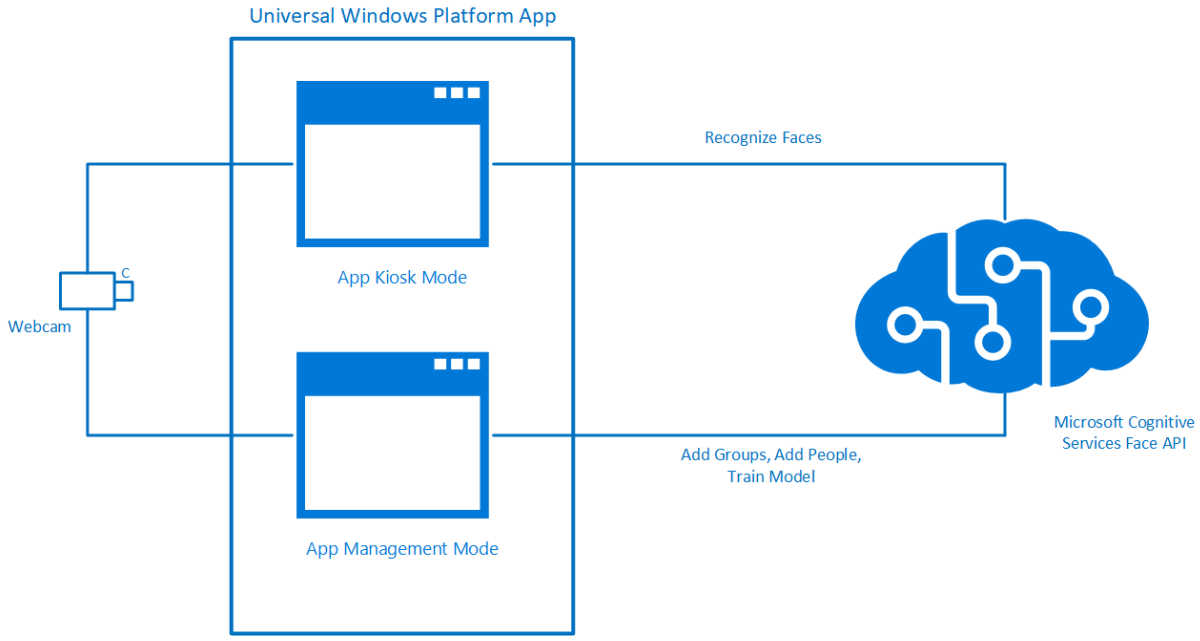
application can make use of all Windows compatible hardware like webcams and will run on a variety of devices like Intel Compute Sticks, Laptop and Desktop PCs, Kiosk PCs, the Surface Hub and Windows IoT Core Devices. This gives ISS great flexibility on the hardware to run the app on.

Another advantage of using UWP is, that there is a great sample project (Intelligent Kiosk) that could be used as a reference for working with Microsoft Cognitive Services.

As an easy to use out-of-the-box solution for face recognition was required the team decided to use the Microsoft Cognitive Services Face API that can not only be used to recognize faces and return face properties, but also to identify known faces. The face identification feature is used by first creating a new group of people through the API, then adding a person to the database and then adding face images for each person. After this the face recognition model has to be trained to enable face recognition and identification.

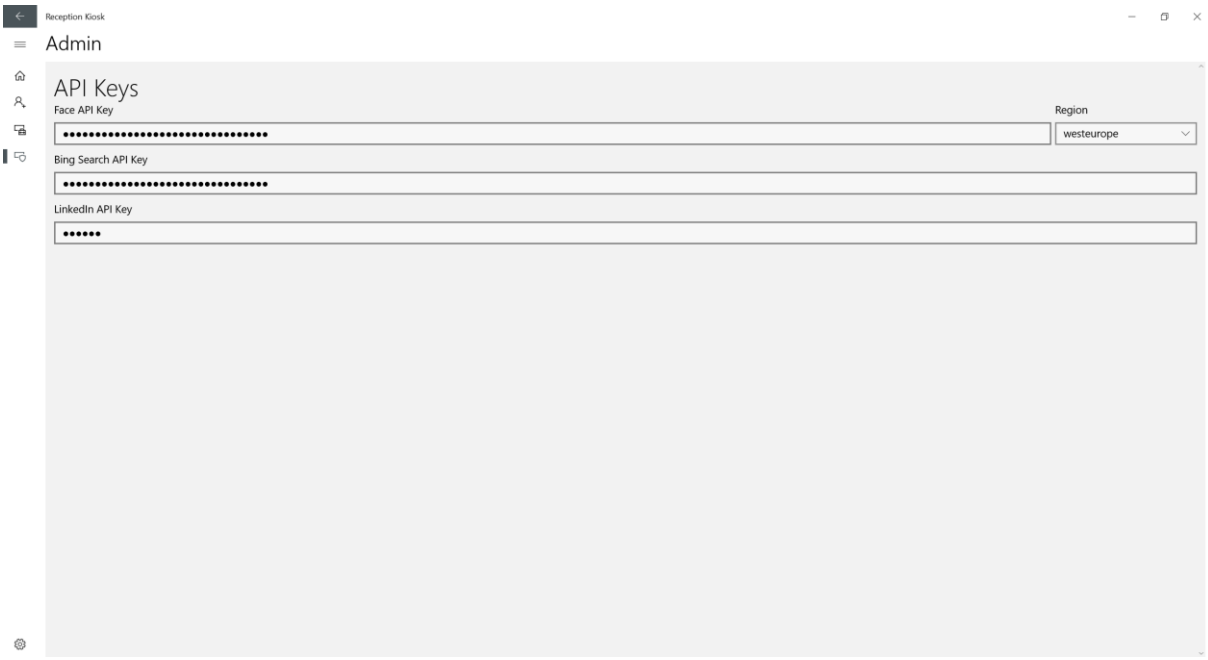
The team decided to just develop one app that would contain several views for management tasks, adding faces for recognition and for identifying people. These led to the following architecture for the application.

Solution architecture:

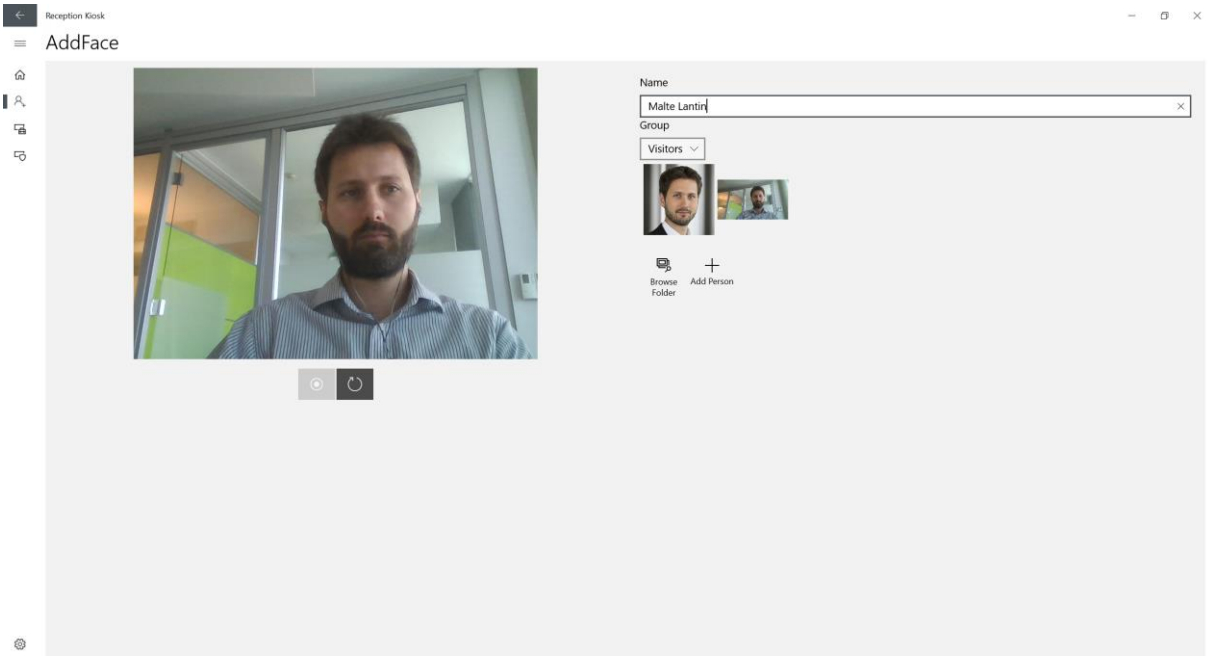


When using the application the user first has to enter a valid Cognitive Services Face API Key and then can add faces to be recognized later. As the plan is to enable web search for images and adding profile information from LinkedIn later the app already implements entering API keys for those services as well, although they are currently not being used.

Screenshot API keys:

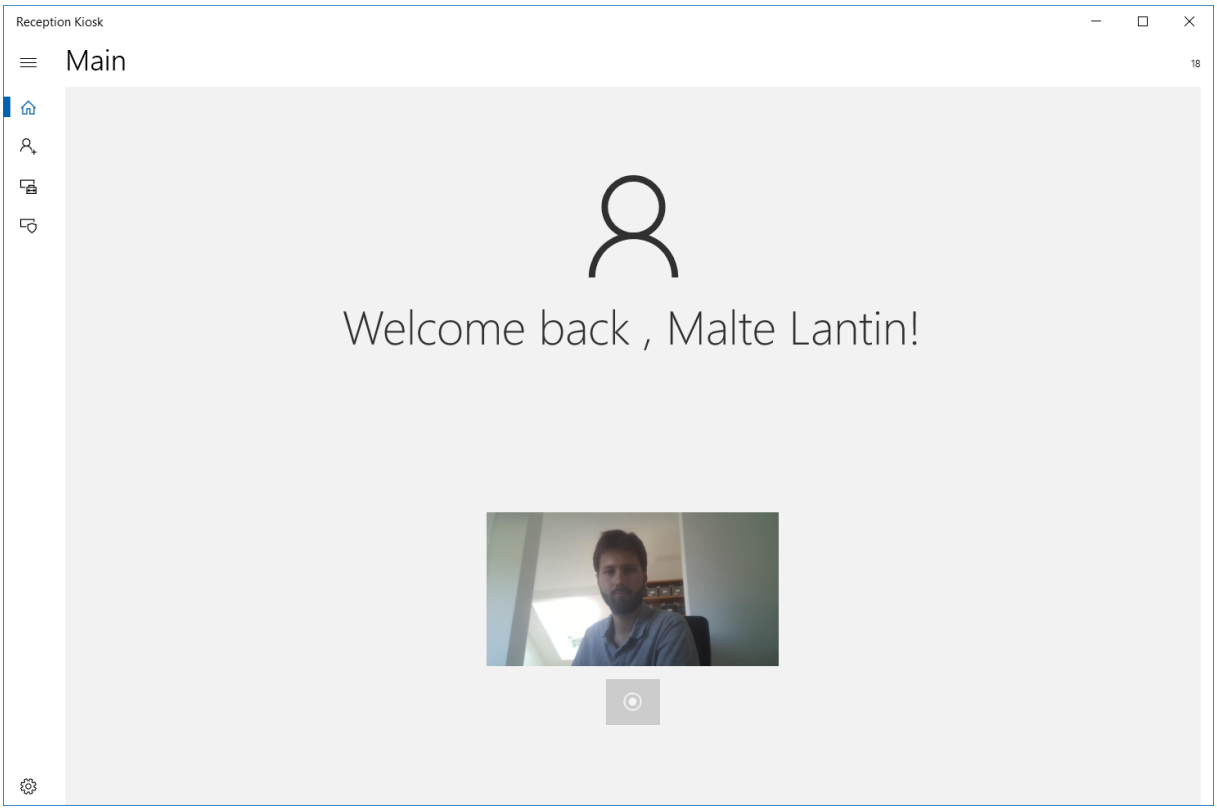


Screenshot adding faces:



Once a people group and a person with several faces it set up, the model behind the service has to be trained for each group. After doing the training the main page of the app can be used for people recognition.

Screenshot visitor recognition:



You can have a look at the project and already use it to manage you own Cognitive Services Face API database by cloning the project on GitHub.

Technical delivery

As mentioned above the project is a Universal Windows Platform application. It was set up using Visual Studio and the new Windows Template Studio. Windows Template Studio has several boilerplate page templates that helped us to quickly set up a project using the Model-View-ViewModel (MVVM) design pattern and navigation. The project was developed in C# with XAML as the UI language.

The application settings like API keys are currently saved locally for the current user and photos that are captured with the app are saved in the user's image library before being used for setting up a face model for recognition. For displaying a webcam preview for the user and capturing the photo we build a custom camera control. For building this control, saving the images and binding them to the UI the UWP documentation helped a lot. You can find more information on previewing the camera stream in the library here. Saving and binding images through the SoftwareBitmap class is documented in the UWP docs here.

The project makes use of some of the helper classes in the Intelligent Kiosk sample, which in general was very helpful, although sometimes being too complex. I recommend everyone who is building a project with Microsoft Cognitive Services to take a look at this sample project.

For working with the Cognitive Service Face API we used the corresponding Windows SDK which is compatible with UWP apps. The SDK is available through NuGet as Microsoft.ProjectOxford.Face.

As mentioned in the solution description above, one or more people groups have to be added to do face recognition. This is done though the following code using the Face API SDK:

```
private async Task ExecuteAddGroupCommandAsync()
{
```

```
        try
        {
            if (string.IsNullOrEmpty(GroupToAdd))
                throw new ArgumentNullException(nameof(GroupToAdd), "Please enter a group
name.");

            //Remember which group was selected by its unique ID
            var tempSelectedGroupId = SelectedPersonGroup.PersonGroupId;

            await FaceService.CreatePersonGroupAsync(Guid.NewGuid().ToString(),
GroupToAdd);
            await (new MessageDialog($"'{GroupToAdd}' successfully added.")).ShowAsync();

            //Cleanup UI
            GroupToAdd = string.Empty;
            await LoadGroupsAsync();

            //Set the selected group back to the group we had selected before
            foreach (var group in PersonGroups)
            {
                if (group.PersonGroupId.Equals(tempSelectedGroupId))
                {
                    SelectedPersonGroup = group;
                    break;
                }
            }

        }
        catch (Exception ex)
        {
            var dialog = new MessageDialog(ex.Message, "Group could not be added.");
            await dialog.ShowAsync();
        }
    }
```

ManageFacesViewModel.cs

After groups are created they can be read from the service and displayed in the UI:

```
         private async Task LoadGroupsAsync()
    {
        PersonGroups.Clear();
        var fscPersonGroups = await FaceService.ListPersonGroupsAsync();
        fscPersonGroups.OrderBy(pg => pg.Name).ForEach(pg => PersonGroups.Add(pg));
    }
```

ManageFacesViewModel.cs

Adding a new person for recognition to the service model has two steps. The first step is to add the new person to an existing person group. After this step is complete up to 248 faces can be added for every person (see the API documentation). The face images can be passed to the Face API as a public URL or a stream. In our case we use a a stream as the format to send the images we took with the web cam or images loaded from the local file system.

```
private async Task ExecuteAddPersonCommand()
        {
            if (NewFaceName != string.Empty && Pictures.Count > 0 && SelectedPersonGroup !=
null)
            {
                IsLoading = true;
                try
                {
                    List<AddPersistedFaceResult> faces = new
List<AddPersistedFaceResult>();

                    var result = await
FaceService.CreatePersonAsync(SelectedPersonGroup.PersonGroupId, NewFaceName);
```

```
                    foreach (var picture in Pictures)
                    {
                        var currentPicture = picture.Bitmap;

                        IRandomAccessStream randomAccessStream = new
InMemoryRandomAccessStream();

                        BitmapEncoder encoder = await
BitmapEncoder.CreateAsync(BitmapEncoder.JpegEncoderId, randomAccessStream);

                        encoder.SetSoftwareBitmap(currentPicture);

                        await encoder.FlushAsync();

                        var stream = randomAccessStream.AsStreamForRead();

                        faces.Add(await
FaceService.AddPersonFaceAsync(SelectedPersonGroup.PersonGroupId, result.PersonId,
stream));
                    }

                    await new MessageDialog($"Successfully added {faces.Count} faces for
person {NewFaceName} ({result.PersonId}).").ShowAsync();

                    //Reset the form
                    Pictures.Clear();
                    NewFaceName = "";
                }
                catch (FaceAPIException e)
                {
                    await new MessageDialog(e.ErrorMessage).ShowAsync();
                    //await new
MessageDialog(loader.GetString("AddFace_CompleteInformation")).ShowAsync();
                }
                finally
                {
                    IsLoading = false;
                }
            }
            else
            {
                await new
MessageDialog(loader.GetString("AddFace_CompleteInformation")).ShowAsync();
            }
        }
```

AddFaceViewModel.cs

The last step when adding a new person is training the model for the person group the new person is a member of.

```
    private async Task ExecuteTrainCommandAsync()
    {
        try
        {
            IsLoading = true;
            await FaceService.TrainPersonGroupAsync(SelectedPersonGroup.PersonGroupId);
            IsLoading = false;
            await MessageDialogHelper.MessageDialogAsync($"Group {SelectedPersonGroup.Name}
has successfully been trained.");
        }
        catch (Exception e)
        {
            await MessageDialogHelper.MessageDialogAsync("Group could not be trained",
e.Message);
        }
        finally
        {
```

```
            IsLoading = false;
        }
    }
```

ManageFacesViewModel.cs

After a person has been added and the model has been training, face recognition through the Cognitive Services Face API is ready to use. Recognizing a face is a multi-step process again and needs to be done per available group. In the Reception Kiosk application we used the video stream from the web cam to recognize faces and identify people we already know. To save calls to the Cognitive Services API we combine using the API with offline face recognition available in UWP and Windows 10. You can find the code for using offline face recognition in the code for our camera control that handles this part (ReceptionCamera.xaml.cs). We only call the Face API when faces are detected offline on the camera stream. Right now we just do face recognition every few seconds which seems to be enough for this use case and reduces API usage. Once faces are recognized a capture from the video stream is taken and send to the face API for detection. Recognized faces are returned with a face ID for further usage.

```
var faces = await FaceService.DetectAsync(stream, true, false);
```

ReceptionCamera.xaml.cs

For looking for a person in all available groups we have to loop through the groups and use the group ID and face IDs from the steps before. The service returns all identified face candidates, that can then be used for returning the right person.

```
foreach (var group in _personGroups)
         {
                  foreach (var face in _cameraControl.LastFaces)
                  {
                           IdentifyResult[] results;
                           try
                           {
                                    results = await
FaceService.IdentifyAsync(group.PersonGroupId, _cameraControl.LastFaces);
                                    foreach (var result in results)
                                    {
                                             if (result.Candidates.Length > 0)
                                             {
                                                      var resultCandidate =
result.Candidates[0];
                                                      //Let's start with 50% confidence
for now
                                                      if (resultCandidate.Confidence >
0.5)
                                                      {
                                                               try
                                                               {
                                                                        var
identifiedPerson = await FaceService.GetPersonAsync(group.PersonGroupId,
resultCandidate.PersonId);

          people.Add(identifiedPerson.Name);
                                                               }
                                                               catch
                                                               {
                                                   //Handle possible API exceptions
                                                               }
                                                      }
                                             }
                                    }
                           }
                  }
         }
```

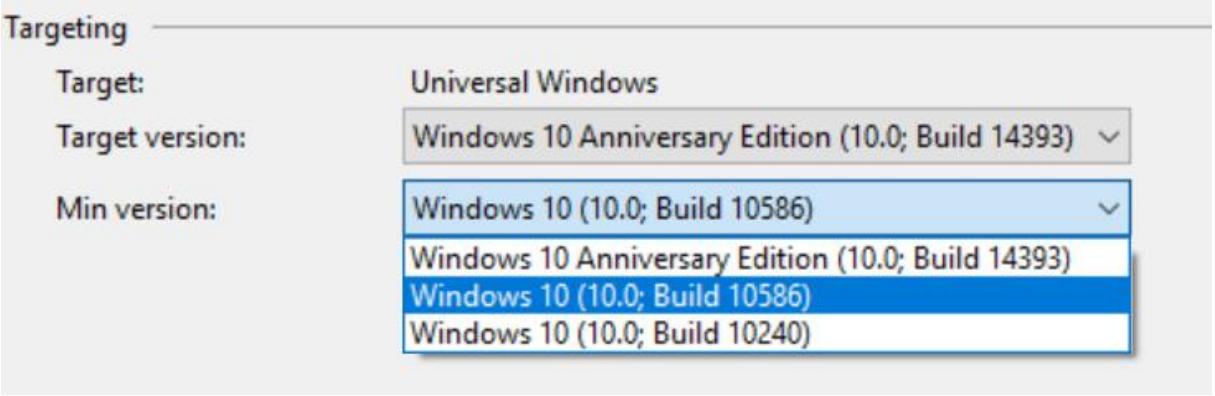
MainViewModel.cs

Once the faces have been matched to the right person, the UI can be updated to greet the visitor. After each successful face recognition loop we wait a few seconds to reduce API calls.

Challenges

When developing the project we sometimes struggled with differences between UWP XAML and WPF XAML as the developers had very few experience with UWP until then, but these problems could all be solved. When looking for information on working with specific classes in UWP I recommend to always include "UWP" in your search term as classes with the same name probably also exist in WPF and might differ.

Another learning is to chose the Minimum target version for your UWP project wisely. As we wanted the solution to run on as many Windows 10 systems as possible we set Windows Build 10586 as the minimum Version in the application settings.



This version allows the project to run on all recent major versions of Windows but does not allow to make use of some of the newest features available in the Windows Anniversary or Creators Update without conditional code. You can find more information on this topic in the documentation.

Adding groups, adding people to the groups, adding face images and then training a group is pretty straightforward when using the Windows SDK for the Cognitive Services Face API. You can find the code we used in the ViewModel for the Manage Faces page and Add Face page.

The documentation and source for the SDK is available on GitHub as well. The full API reference for the Face API can be found online. At one point we had some trouble with adding faces to a person which was caused by wrong encoding for the images we send to the face API and a size that was not accepted by the service. All requirements for image data can be found in the API documentation. Most important for V1.0 are the following restrictions:

JPEG, PNG, GIF(the first frame), and BMP are supported. The image file size should be larger than or equal to 1KB but no larger than 4MB.

This is valid for both adding a face for a person and identifying a face.

One more thing to keep in mind is that the face recognition on the video stream has to be asynchronously and might involve running a detection loop on a different thread. This might cause some challenges in handling the data between this threads.

Conclusion

The project is currently under development and will be extended with further features. The greatest learning of working on this project was that when using the Windows SDK using the Cognitive Services Face API is easy to use as it supports both URLs and Streams as an input and will return a .Net object to work with. Working with the webcam and finding the right way to

capture, save and bind images took us some time. Our approach can be found in the project code. Please provide any input and feedback through the GitHub project if you have any.