LAMPIRAN

Lampiran - 1 Listing Program

1. Modul Api (Koneksi Data Karya Seni)
Api/ApiEndpoint.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;

namespace FokusTour.Api
{
    /// <summary>
    ///  Resolves API URL: prefer HTTPS, fall back to HTTP.
    ///     Default     host: fokustour.my.id
    /// </summary>
    public    static    class ApiEndpoint
    {
        public const string DefaultHost                 = "fokustour.my.id";
        public const string DefaultApiPath              = "/api/artworks.php";

        public static string HttpsApiUrl => "https://" + DefaultHost + DefaultApiPath;
        public static string HttpApiUrl => "http://" + DefaultHost + DefaultApiPath;

        public static string ToHttps(string url)
        {
            if (string.IsNullOrWhiteSpace(url))
                return url;

            if (url.StartsWith("http://", System.StringComparison.OrdinalIgnoreCase))
                return "https://"                   + url.Substring("http://".Length);

            return url;
        }

        public static string ToHttp(string url)
        {
            if (string.IsNullOrWhiteSpace(url))
                return url;

            if (url.StartsWith("https://", System.StringComparison.OrdinalIgnoreCase))
                return "http://"                    + url.Substring("https://".Length);

            return url;
        }

        public static bool IsRequestFailed(UnityWebRequest request)
        {
#if UNITY_2020_2_OR_NEWER
            return request.result               != UnityWebRequest.Result.Success;
#else
            return request.isNetworkError       || request.isHttpError;
#endif
        }

        public static string BuildCandidate(string preferredUrl,             bool preferHttps)
        {
            string    url    = string.IsNullOrWhiteSpace(preferredUrl)   ?   HttpsApiUrl   : preferredUrl.Trim();

            if
```

```
(!url.StartsWith("http://",
System.StringComparison.Ordina
lIgnoreCase) &&

!url.StartsWith("https://",
System.StringComparison.Ordina
lIgnoreCase))
            {
                url          =
(preferHttps  ?  "https://"  :
"http://")                   +
url.TrimStart('/');
            }

            return preferHttps
? ToHttps(url) : ToHttp(url);
        }

        /// <summary>
        ///  Probe  HTTPS  then
HTTP.  Invokes  onResolved  with
the working URL, or onFailed.
        /// </summary>
        public          static
IEnumerator
ResolveWorkingApiUrl(
            string
preferredUrl,

System.Action<string>
onStatus,

System.Action<string>
onResolved,

System.Action<string>
onFailed)
        {
            string  httpsUrl  =
BuildCandidate(preferredUrl,
preferHttps: true);
            string  httpUrl  =
BuildCandidate(preferredUrl,
preferHttps: false);


onStatus?.Invoke("Menghubungka
n (HTTPS)...");
            string  httpsError
= null;
            bool   httpsOk   =
false;
            yield       return
ProbeGet(httpsUrl,  (ok,  err)
=>
            {
                httpsOk = ok;
                httpsError   =
err;
            });

            if (httpsOk)
            {

onResolved?.Invoke(httpsUrl);
                yield break;
            }


onStatus?.Invoke("HTTPS gagal,
mencoba HTTP...");
            string httpError =
null;
            bool   httpOk   =
false;
            yield       return
ProbeGet(httpUrl, (ok, err) =>
            {
                httpOk = ok;
                httpError    =
err;
            });

            if (httpOk)
            {

onResolved?.Invoke(httpUrl);
                yield break;
            }

            string   detail   =
!string.IsNullOrEmpty(httpsErr
or) ? httpsError : httpError;
            if
(string.IsNullOrEmpty(detail))
                detail       =
"Tidak   dapat   terhubung   ke
server.";


onFailed?.Invoke(detail);
        }

        /// <summary>
        ///  Download  texture
preferring  HTTPS,  then  HTTP
equivalent.
        /// </summary>
        public          static
IEnumerator
DownloadTexturePreferHttps(str
ing                  imageUrl,
```

```
System.Action<Texture2D>
onSuccess)
        {
            if
(string.IsNullOrWhiteSpace(ima
geUrl))
                yield break;

            string httpsUrl =
ToHttps(imageUrl);
            string httpUrl =
ToHttp(imageUrl);

            Texture2D texture
= null;
            yield return
TryDownloadTexture(httpsUrl, t
=> texture = t);
            if (texture !=
null)
            {

onSuccess?.Invoke(texture);
                yield break;
            }

            if
(!string.Equals(httpsUrl,
httpUrl,
System.StringComparison.Ordina
lIgnoreCase))
            {
                yield return
TryDownloadTexture(httpUrl, t
=> texture = t);
                if (texture !=
null)
                {

onSuccess?.Invoke(texture);
                    yield
break;
                }
            }


Debug.LogWarning($"ApiEndpoint
:    gagal    unduh    gambar
({imageUrl}).");
        }

        private         static
IEnumerator    ProbeGet(string
url,       System.Action<bool,
string> done)
        {
            using
(UnityWebRequest   request   =
UnityWebRequest.Get(url))
            {

request.timeout = 12;
                yield return
request.SendWebRequest();

                if
(IsRequestFailed(request))
                {

done?.Invoke(false,
string.IsNullOrEmpty(request.e
rror)  ?  "Koneksi  gagal."  :
request.error);
                    yield
break;
                }

                // Accept any
successful HTTP response (JSON
validated later by caller).

done?.Invoke(true, null);
            }
        }

        private         static
IEnumerator
TryDownloadTexture(string url,
System.Action<Texture2D> done)
        {
            using
(UnityWebRequest   request   =
UnityWebRequestTexture.GetText
ure(url))
            {

request.timeout = 20;
                yield return
request.SendWebRequest();

                if
(IsRequestFailed(request))
                {

done?.Invoke(null);
                    yield
break;
                }


done?.Invoke(DownloadHandlerTe
xture.GetContent(request));
```

```
            }
        }
    }
}
```

Api/ArtworkApiDtos.cs

```
using System;

namespace FokusTour.Api
{
    [Serializable]
    public class ArtworkListResponse
    {
        public bool success;
        public int count;
        public ArtworkDto[] data;
    }

    [Serializable]
    public class ArtworkDto
    {
        public int id;
        public string title;
        public string creator_name;
        public string description;
        public string image_url;
        public int is_active;
    }
}
```

Api/ArtworkApiLoader.cs

```
using System.Collections;
using System.Collections.Generic;
using FokusTour.Artwork;
using FokusTour.Player;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;
using TMPro;

namespace FokusTour.Api
{
    /// <summary>
    /// Binds artwork data to ArtworkInteractable in the gallery scene.
    ///                 Prefers ArtworkSessionCache (prefetched from Main Menu); falls back to live API fetch.
    /// </summary>
    public class ArtworkApiLoader : MonoBehaviour
    {
        public static ArtworkApiLoader Instance { get; private set; }

        [Header("API")]
        [Tooltip("Preferred API URL. App tries HTTPS first, then HTTP.")]
        [SerializeField] private string apiUrl = ApiEndpoint.HttpsApiUrl;
        [SerializeField] private bool loadOnStart = true;

        [Header("Loading Gate")]
        [SerializeField] private bool lockPlayerUntilLoaded = true;
        [SerializeField] private bool showLoadingOverlay = true;

        private readonly Dictionary<int, ArtworkItem> _itemsById = new Dictionary<int, ArtworkItem>();

        private GameObject _loadingRoot;
        private TextMeshProUGUI _loadingStatusText;
        private FirstPersonController _playerController;
        private bool _playerWasEnabled;

        public bool IsLoaded { get; private set; }
        public string LastError { get; private set; }

        public string ApiUrl => apiUrl;

        private void Awake()
```

```
        {
            if (Instance != null && Instance != this)
            {
                Destroy(gameObject);
                return;
            }

            Instance = this;
        }

        private void Start()
        {
            if (loadOnStart)
                StartCoroutine(LoadAndBindRoutine());
        }

        public void Reload()
        {
            StopAllCoroutines();
            _itemsById.Clear();
            IsLoaded = false;
            LastError = null;
            StartCoroutine(LoadAndBindRoutine());
        }

        public bool TryGetItem(int artworkId, out ArtworkItem item)
        {
            return _itemsById.TryGetValue(artworkId, out item);
        }

        private IEnumerator LoadAndBindRoutine()
        {
            IsLoaded = false;
            LastError = null;

            BeginLoadingGate();
            SetLoadingStatus("Memuat data karya...");

            ArtworkSessionCache cache = ArtworkSessionCache.Instance;
            if (cache != null && cache.IsReady)
            {
                CopyFromCache(cache);
                SetLoadingStatus("Menyiapkan karya...");
                yield return BindInteractables();
                FinishLoadingGate(success: true);
                yield break;
            }

            yield return FetchFromApi();
            if (!string.IsNullOrEmpty(LastError))
            {
                SetLoadingStatus(LastError);
                FinishLoadingGate(success: false);
                yield break;
            }

            yield return BindInteractables();
            FinishLoadingGate(success: true);
        }

        private void CopyFromCache(ArtworkSessionCache cache)
        {
            _itemsById.Clear();
            foreach (KeyValuePair<int, ArtworkItem> pair in cache.Items)
                _itemsById[pair.Key] = pair.Value;
        }

        private IEnumerator FetchFromApi()
        {
```

```
            string workingUrl
= null;
            yield       return
ApiEndpoint.ResolveWorkingApiU
rl(
                apiUrl,

SetLoadingStatus,
                url         =>
workingUrl = url,
                err         =>
LastError = err);

            if
(string.IsNullOrEmpty(workingU
rl))
            {
                if
(string.IsNullOrEmpty(LastErro
r))
                    LastError
= "Gagal terhubung ke server.";

Debug.LogError($"ArtworkApiLoa
der: {LastError}");
                yield break;
            }


SetLoadingStatus("Mengunduh
data karya...");

            using
(UnityWebRequest   request   =
UnityWebRequest.Get(workingUrl
))
            {

request.timeout = 15;
                yield   return
request.SendWebRequest();

                if
(ApiEndpoint.IsRequestFailed(r
equest))
                {
                    LastError
= request.error;

Debug.LogError($"ArtworkApiLoa
der:    gagal    memuat    API
({workingUrl}).
{request.error}");
                    yield
break;
                }

ArtworkListResponse response =
JsonUtility.FromJson<ArtworkLi
stResponse>(request.downloadHa
ndler.text);
                if   (response
== null || !response.success ||
response.data == null)
                {
                    LastError
= "Respons API tidak valid.";

Debug.LogError("ArtworkApiLoad
er: respons API tidak valid.");
                    yield
break;
                }


_itemsById.Clear();
                foreach
(ArtworkDto       dto       in
response.data)
                {
                    if (dto ==
null || dto.is_active == 0)

continue;


ArtworkItem       item       =
ArtworkItem.FromDto(dto);
                    if   (item
!= null)

_itemsById[item.Id] = item;
                }
            }

            if
(_itemsById.Count == 0)
            {
                LastError    =
"Tidak ada data karya aktif.";
                yield break;
            }

            List<ArtworkItem>
items           =           new
List<ArtworkItem>(_itemsById.V
alues);
            for (int i = 0; i
< items.Count; i++)
            {
                ArtworkItem
```

```
item = items[i];

SetLoadingStatus($"Mengunduh
gambar          ({i          +
1}/{items.Count})...");

                if
(!string.IsNullOrWhiteSpace(it
em.ImageUrl))
                {
                    yield
return
ApiEndpoint.DownloadTexturePre
ferHttps(item.ImageUrl,
texture =>
                    {

item.PreviewTexture = texture;
                    });
                }
            }
        }

        private    IEnumerator
BindInteractables()
        {

ArtworkInteractable[]
interactables                =
FindObjectsByType<ArtworkInter
actable>(FindObjectsSortMode.N
one);
            foreach
(ArtworkInteractable
interactable in interactables)
            {
                int    id    =
interactable.ArtworkId;
                if (id <= 0 ||
!_itemsById.TryGetValue(id,
out ArtworkItem item))
                    continue;


interactable.ApplyRuntimeData(
item);
            }

            IsLoaded = true;

Debug.Log($"ArtworkApiLoader:
{_itemsById.Count}       karya
dimuat.");
            yield return null;
        }

        private           void
BeginLoadingGate()
        {
            if
(lockPlayerUntilLoaded)
            {

_playerController            =
FindFirstObjectByType<FirstPer
sonController>();
                if
(_playerController != null)
                {

_playerWasEnabled            =
_playerController.enabled;

_playerController.enabled    =
false;
                }
            }

            if
(showLoadingOverlay)

EnsureLoadingOverlay();

            if   (_loadingRoot
!= null)

_loadingRoot.SetActive(true);
        }

        private           void
FinishLoadingGate(bool
success)
        {
            IsLoaded = success
|| IsLoaded;

            if   (_loadingRoot
!= null)
            {
                if (success)

_loadingRoot.SetActive(false);
                else

SetLoadingStatus(string.IsNull
OrEmpty(LastError)
                        ?
"Gagal memuat data."
                        :
LastError + "\nKembali ke menu
untuk mencoba lagi.");
            }
```

```
            if (success &&
lockPlayerUntilLoaded &&
_playerController != null)

_playerController.enabled =
_playerWasEnabled;
        }

        private void
SetLoadingStatus(string
message)
        {
            if
(_loadingStatusText != null)

_loadingStatusText.text =
message;
        }

        private void
EnsureLoadingOverlay()
        {
            if (_loadingRoot
!= null)
                return;

            GameObject
canvasObject = new
GameObject("TourLoadingCanvas"
, typeof(RectTransform));

canvasObject.transform.SetPare
nt(transform, false);

            Canvas canvas =
canvasObject.AddComponent<Canv
as>();
            canvas.renderMode
=
RenderMode.ScreenSpaceOverlay;

canvas.sortingOrder = 200;

            CanvasScaler
scaler =
canvasObject.AddComponent<Canv
asScaler>();
            scaler.uiScaleMode
=
CanvasScaler.ScaleMode.ScaleWi
thScreenSize;

scaler.referenceResolution =
new Vector2(1920f, 1080f);
scaler.matchWidthOrHeight =
0.5f;


canvasObject.AddComponent<Grap
hicRaycaster>();

            GameObject
dimmerObject = new
GameObject("Dimmer",
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(Image));

dimmerObject.transform.SetPare
nt(canvasObject.transform,
false);
            RectTransform
dimmerRect =
dimmerObject.GetComponent<Rect
Transform>();

dimmerRect.anchorMin =
Vector2.zero;

dimmerRect.anchorMax =
Vector2.one;

dimmerRect.offsetMin =
Vector2.zero;

dimmerRect.offsetMax =
Vector2.zero;
            Image dimmer =
dimmerObject.GetComponent<Imag
e>();
            dimmer.color = new
Color(0.05f, 0.06f, 0.08f,
0.92f);

dimmer.raycastTarget = true;

            GameObject
titleObject = new
GameObject("Title",
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(TextMeshProUGUI));

titleObject.transform.SetParen
t(dimmerObject.transform,
false);
            TextMeshProUGUI
title =
titleObject.GetComponent<TextM
eshProUGUI>();
```

```
            title.text       =
"Memuat Tour";
            title.fontSize   =
42f;
            title.fontStyle  =
FontStyles.Bold;
            title.color      =
Color.white;
            title.alignment  =
TextAlignmentOptions.Center;

title.raycastTarget = false;
            RectTransform
titleRect                    =
title.rectTransform;

titleRect.anchorMin    =    new
Vector2(0.5f, 0.5f);

titleRect.anchorMax    =    new
Vector2(0.5f, 0.5f);
            titleRect.pivot  =
new Vector2(0.5f, 0.5f);

titleRect.anchoredPosition   =
new Vector2(0f, 40f);

titleRect.sizeDelta    =    new
Vector2(900f, 60f);

            GameObject
statusObject       =       new
GameObject("Status",
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(TextMeshProUGUI));

statusObject.transform.SetPare
nt(dimmerObject.transform,
false);
            _loadingStatusText
=
statusObject.GetComponent<Text
MeshProUGUI>();

_loadingStatusText.text      =
"Memuat data karya...";

_loadingStatusText.fontSize  =
26f;

_loadingStatusText.color = new
Color(0.75f,   0.78f,   0.82f,
1f);

_loadingStatusText.alignment =
TextAlignmentOptions.Center;

_loadingStatusText.raycastTarg
et = false;
            RectTransform
statusRect                   =
_loadingStatusText.rectTransfo
rm;

statusRect.anchorMin   =   new
Vector2(0.5f, 0.5f);

statusRect.anchorMax   =   new
Vector2(0.5f, 0.5f);
            statusRect.pivot =
new Vector2(0.5f, 0.5f);

statusRect.anchoredPosition  =
new Vector2(0f, -20f);

statusRect.sizeDelta   =   new
Vector2(1000f, 100f);

            _loadingRoot     =
canvasObject;
        }
    }
}
```

Api/ArtworkSessionCache.cs

```
using System;
using System.Collections;
using
System.Collections.Generic;
using FokusTour.Artwork;
using UnityEngine;
using UnityEngine.Networking;

namespace FokusTour.Api
{
    /// <summary>
    ///   Prefetches   artwork
metadata  +  textures  before
entering the gallery scene.
    /// Survives scene loads
via DontDestroyOnLoad.
    /// </summary>
    public               class
ArtworkSessionCache          :
MonoBehaviour
    {
        public          static
ArtworkSessionCache Instance {
get; private set; }

        private       readonly
```

```
Dictionary<int, ArtworkItem> _itemsById = new Dictionary<int, ArtworkItem>();

        public bool IsReady { get; private set; }
        public string LastError { get; private set; }
        public string ResolvedApiUrl { get; private set; }
        public int TotalCount { get; private set; }
        public int LoadedTextureCount { get; private set; }

        public IReadOnlyDictionary<int, ArtworkItem> Items => _itemsById;

        private void Awake()
        {
            if (Instance != null && Instance != this)
            {
                Destroy(gameObject);
                return;
            }

            Instance = this;
            DontDestroyOnLoad(gameObject);
        }

        private void OnDestroy()
        {
            if (Instance == this)
                Instance = null;
        }

        public static ArtworkSessionCache EnsureExists()
        {
            if (Instance != null)
                return Instance;

            GameObject go = new GameObject("ArtworkSessionCache");
            return go.AddComponent<ArtworkSessionCache>();
        }

        public void Clear()
        {
            StopAllCoroutines();

            foreach (ArtworkItem item in _itemsById.Values)
            {
                if (item?.PreviewTexture != null)
                    Destroy(item.PreviewTexture);
            }

            _itemsById.Clear();
            IsReady = false;
            LastError = null;
            ResolvedApiUrl = null;
            TotalCount = 0;
            LoadedTextureCount = 0;
        }

        public bool TryGetItem(int artworkId, out ArtworkItem item)
        {
            return _itemsById.TryGetValue(artworkId, out item);
        }

        public IEnumerator Prefetch(string preferredApiUrl, Action<string> onStatus = null)
        {
            Clear();
            IsReady = false;
            LastError = null;

            string workingUrl
```

```
= null;
            yield       return
ApiEndpoint.ResolveWorkingApiU
rl(
preferredApiUrl,
                onStatus,
                url         =>
workingUrl = url,
                err         =>
LastError = err);

            if
(string.IsNullOrEmpty(workingU
rl))
            {
                if
(string.IsNullOrEmpty(LastErro
r))
                    LastError
= "Gagal terhubung ke server.";
                yield break;
            }

            ResolvedApiUrl   =
workingUrl;

onStatus?.Invoke("Mengunduh
data karya...");

            using
(UnityWebRequest   request   =
UnityWebRequest.Get(workingUrl
))
            {

request.timeout = 15;
                yield   return
request.SendWebRequest();

                if
(ApiEndpoint.IsRequestFailed(r
equest))
                {
                    LastError
=
string.IsNullOrEmpty(request.e
rror)
                        ?
"Gagal terhubung ke server."
                        :
request.error;
                    yield
break;
                }

ArtworkListResponse response =
JsonUtility.FromJson<ArtworkLi
stResponse>(request.downloadHa
ndler.text);
                if   (response
== null || !response.success ||
response.data == null)
                {
                    LastError
= "Respons API tidak valid.";
                    yield
break;
                }

                foreach
(ArtworkDto       dto       in
response.data)
                {
                    if (dto ==
null || dto.is_active == 0)

continue;


ArtworkItem       item       =
ArtworkItem.FromDto(dto);
                    if   (item
!= null)

_itemsById[item.Id] = item;
                }
            }

            TotalCount       =
_itemsById.Count;
            if  (TotalCount  ==
0)
            {
                LastError    =
"Tidak ada data karya aktif.";
                yield break;
            }

            LoadedTextureCount
= 0;
            List<ArtworkItem>
items           =           new
List<ArtworkItem>(_itemsById.V
alues);
            for (int i = 0; i
< items.Count; i++)
            {
                ArtworkItem
item = items[i];
```

```
onStatus?.Invoke($"Mengunduh gambar ({i + 1}/{items.Count})...");

                if (!string.IsNullOrWhiteSpace(item.ImageUrl))
                {
                    yield return ApiEndpoint.DownloadTexturePreferHttps(item.ImageUrl, texture =>
                    {
                        item.PreviewTexture = texture;
                    });
                }

                LoadedTextureCount++;
            }

            IsReady = true;
            onStatus?.Invoke("Data siap.");
        }
    }
}
```

2. Modul Artwork (Objek Karya Seni)
Artwork/ArtworkInteractable.cs

```
using System.Text.RegularExpressions;
using UnityEngine;

namespace FokusTour.Artwork
{
    /// <summary>
    /// Place on each artwork object. Requires a Collider for proximity detection.
    /// Set artworkId to match row id in MySQL (e.g. Artwork_01_01 → 1).
    /// Frame texture is loaded at runtime from API only.
    /// </summary>
    [RequireComponent(typeof(Collider))]
    public class ArtworkInteractable : MonoBehaviour
    {
        private static readonly int BaseMapId = Shader.PropertyToID("_BaseMap");

        [SerializeField] private int artworkId;

        private ArtworkItem _runtimeItem;

        public int ArtworkId => artworkId;

        public bool HasData => _runtimeItem != null && _runtimeItem.HasMetadata;

        private void Awake()
        {
            if (artworkId <= 0)
                artworkId = ParseIdFromObjectName(gameObject.name);

            ClearFrameTexture();
        }

        public ArtworkItem GetActiveItem()
        {
            return HasData ? _runtimeItem : null;
        }

        public void ApplyRuntimeData(ArtworkItem item)
        {
            _runtimeItem = item;
            ApplyFrameTexture(item?.PreviewTexture);
        }

        private void ApplyFrameTexture(Texture2D texture)
        {
            MeshRenderer renderer =
```

```
GetComponent<MeshRenderer>();
            if   (renderer   ==
null)
                return;

            Material  material
= renderer.material;
            if   (material   ==
null)
                return;

            if   (texture   ==
null)
            {

material.SetTexture(BaseMapId,
null);
                return;
            }


material.SetTexture(BaseMapId,
texture);
        }

        private           void
ClearFrameTexture()
        {

ApplyFrameTexture(null);
        }

        private   static   int
ParseIdFromObjectName(string
objectName)
        {
            Match   match   =
Regex.Match(objectName,
@"Artwork_\d+_(\d+)$");
            if  (match.Success
&&
int.TryParse(match.Groups[1].V
alue, out int id))
                return id;

            return 0;
        }

        private void Reset()
        {
            Collider   col   =
GetComponent<Collider>();
            if (col != null)
                col.isTrigger
= false;
        }
#if UNITY_EDITOR
        private           void
OnDrawGizmosSelected()
        {
            Gizmos.color = new
Color(0.2f, 0.8f, 1f, 0.35f);

Gizmos.DrawWireSphere(transfor
m.position, 0.25f);
        }
#endif
    }
}
```

Artwork/ArtworkItem.cs

```
using FokusTour.Api;
using UnityEngine;

namespace FokusTour.Artwork
{
    /// <summary>
    ///  Runtime  artwork  data
loaded from API.
    /// </summary>
    public class ArtworkItem
    {
        public  int  Id  {  get;
set; }
        public string Title {
get; set; }
        public          string
CreatorName { get; set; }
        public          string
Description { get; set; }
        public string ImageUrl
{ get; set; }
        public       Texture2D
PreviewTexture { get; set; }

        public            bool
HasMetadata =>

!string.IsNullOrWhiteSpace(Tit
le) ||

!string.IsNullOrWhiteSpace(Cre
atorName) ||

!string.IsNullOrWhiteSpace(Des
cription);

        public          static
ArtworkItem FromDto(ArtworkDto
dto)
        {
```

```
            if (dto == null)
                return null;

            return         new
ArtworkItem
            {
                Id = dto.id,
                Title        =
dto.title,
                CreatorName  =
dto.creator_name,
                Description  =
dto.description,
                ImageUrl     =
dto.image_url,
            };
        }
    }
}
```

3. Modul Input (Kontrol Perangkat Mobile)

Input/TouchLookArea.cs

```
using UnityEngine;
using
UnityEngine.EventSystems;

namespace FokusTour.Input
{
    /// <summary>
    /// Transparent UI area for
first-person look. Place on the
right side of the screen.
    /// </summary>
    public class TouchLookArea
:               MonoBehaviour,
IPointerDownHandler,
IDragHandler,
IPointerUpHandler
    {
        private        Vector2
_delta;
        private           bool
_isDragging;

        /// <summary>
        /// Look delta since
last consume. Call from the
controller once per frame.
        /// </summary>
        public         Vector2
ConsumeDelta()
        {
            Vector2   value   =
_delta;
            _delta           =
Vector2.zero;
            return value;
        }

        public bool IsDragging
=> _isDragging;

        public            void
OnPointerDown(PointerEventData
eventData)
        {
            _isDragging      =
true;
            _delta           =
Vector2.zero;
        }

        public            void
OnDrag(PointerEventData
eventData)
        {
            _delta          +=
eventData.delta;
        }

        public            void
OnPointerUp(PointerEventData
eventData)
        {
            _isDragging      =
false;
            _delta           =
Vector2.zero;
        }
    }
}
```

Input/VirtualJoystick.cs

```
using UnityEngine;
using
UnityEngine.EventSystems;
using UnityEngine.UI;

namespace FokusTour.Input
{
    /// <summary>
    /// Virtual joystick for
mobile movement.
    ///    Generates    simple
circular sprites at runtime so
no external art is required.
    /// </summary>
    public               class
VirtualJoystick              :
MonoBehaviour,
IPointerDownHandler,
```

```
IDragHandler,
IPointerUpHandler
    {
        [Header("References")]
        [SerializeField]
private          RectTransform
background;
        [SerializeField]
private RectTransform handle;
        [SerializeField]
private  float  handleRange  =
75f;

        [Header("Look")]
        [SerializeField]
private bool generateVisuals =
true;
        [SerializeField]
private  Color  backgroundColor
= new Color(1f, 1f, 1f, 0.28f);
        [SerializeField]
private Color handleColor = new
Color(1f, 1f, 1f, 0.75f);
        [SerializeField]
private                  Color
handleActiveColor    =    new
Color(1f, 1f, 1f, 0.9f);
        [SerializeField]
private int spriteSize = 128;
        [SerializeField]
[Range(0.05f,  0.45f)] private
float  backgroundRingThickness
= 0.18f;

        private        Vector2
_input;
        private         Canvas
_canvas;
        private         Camera
_uiCamera;
        private          Image
_backgroundImage;
        private          Image
_handleImage;
        private         Sprite
_backgroundSprite;
        private         Sprite
_handleSprite;

        public  Vector2  Value
=> _input;

        private void Awake()
        {
            _canvas          =
GetComponentInParent<Canvas>()
;
            if   (_canvas   !=
null && _canvas.renderMode !=
RenderMode.ScreenSpaceOverlay)
                _uiCamera    =
_canvas.worldCamera;

            CacheImages();

            if
(generateVisuals)

ApplyGeneratedVisuals();

            if    (handle    !=
null)

handle.anchoredPosition      =
Vector2.zero;

            if (handleRange <=
0f && background != null)
                handleRange  =
Mathf.Min(background.sizeDelta
.x,  background.sizeDelta.y)  *
0.45f;
        }

        private           void
OnDestroy()
        {

DestroyGeneratedSprite(ref
_backgroundSprite);

DestroyGeneratedSprite(ref
_handleSprite);
        }

        public            void
OnPointerDown(PointerEventData
eventData)
        {
            if   (_handleImage
!= null)

_handleImage.color           =
handleActiveColor;

            OnDrag(eventData);
        }

        public            void
OnDrag(PointerEventData
eventData)
        {
```

```
            if (background ==
null)
                return;

RectTransformUtility.ScreenPoi
ntToLocalPointInRectangle(
                background,
eventData.position,
                _uiCamera,
                out    Vector2
localPoint);

            Vector2  radius  =
background.sizeDelta * 0.5f;
            if   (radius.x   <
0.01f || radius.y < 0.01f)
                return;

            _input    =    new
Vector2(localPoint.x         /
radius.x,    localPoint.y    /
radius.y);
            _input           =
Vector2.ClampMagnitude(_input,
1f);

            if    (handle    !=
null)
handle.anchoredPosition      =
_input * handleRange;
        }

        public            void
OnPointerUp(PointerEventData
eventData)
        {
            _input           =
Vector2.zero;

            if    (handle    !=
null)
handle.anchoredPosition      =
Vector2.zero;

            if   (_handleImage
!= null)
_handleImage.color           =
handleColor;
        }

        private           void
CacheImages()
        {
            if  (background  !=
null)
_backgroundImage             =
background.GetComponent<Image>
();

            if    (handle    !=
null)
                _handleImage =
handle.GetComponent<Image>();
        }

        private           void
ApplyGeneratedVisuals()
        {
            if
(_backgroundImage != null)
            {
DestroyGeneratedSprite(ref
_backgroundSprite);
_backgroundSprite            =
CreateRingSprite(spriteSize,
backgroundRingThickness);
_backgroundImage.sprite      =
_backgroundSprite;
_backgroundImage.color       =
backgroundColor;
_backgroundImage.type        =
Image.Type.Simple;
_backgroundImage.preserveAspec
t = true;
_backgroundImage.raycastTarget
= true;
            }

            if   (_handleImage
!= null)
            {
DestroyGeneratedSprite(ref
_handleSprite);
                _handleSprite
=
CreateFilledCircleSprite(sprit
eSize);
```

```
_handleImage.sprite          =
_handleSprite;

_handleImage.color           =
handleColor;

_handleImage.type            =
Image.Type.Simple;

_handleImage.preserveAspect  =
true;

_handleImage.raycastTarget   =
false;
            }
        }

        private static Sprite
CreateFilledCircleSprite(int
size)
        {
            Texture2D texture
= new Texture2D(size, size,
TextureFormat.RGBA32, false)
            {
                name         =
"JoystickHandleTexture",
                filterMode   =
FilterMode.Bilinear,
                wrapMode     =
TextureWrapMode.Clamp,
                hideFlags    =
HideFlags.HideAndDontSave
            };

            Color32[] pixels =
new Color32[size * size];
            float   center   =
(size - 1) * 0.5f;
            float   radius   =
center - 1f;
            float  softEdge  =
Mathf.Max(1.5f, size * 0.03f);

            for (int y = 0; y
< size; y++)
            {
                for (int x = 0;
x < size; x++)
                {
                    float dx =
x - center;
                    float dy =
y - center;
                    float dist
= Mathf.Sqrt(dx * dx + dy *
dy);
                    float
alpha = Mathf.Clamp01((radius
- dist) / softEdge);
                    pixels[y *
size + x] = new Color32(255,
255,   255,   (byte)(alpha   *
255f));
                }
            }


texture.SetPixels32(pixels);

texture.Apply(false, true);

            return
Sprite.Create(
                texture,
                new   Rect(0f,
0f, size, size),
                new
Vector2(0.5f, 0.5f),
                100f);
        }

        private static Sprite
CreateRingSprite(int     size,
float thicknessNormalized)
        {
            Texture2D texture
= new Texture2D(size, size,
TextureFormat.RGBA32, false)
            {
                name         =
"JoystickBackgroundTexture",
                filterMode   =
FilterMode.Bilinear,
                wrapMode     =
TextureWrapMode.Clamp,
                hideFlags    =
HideFlags.HideAndDontSave
            };

            Color32[] pixels =
new Color32[size * size];
            float   center   =
(size - 1) * 0.5f;
            float  outerRadius
= center - 1f;
            float  innerRadius
=   outerRadius   *   (1f   -
Mathf.Clamp(thicknessNormalize
d, 0.05f, 0.45f));
            float  softEdge  =
Mathf.Max(1.5f, size * 0.03f);
```

```
            for (int y = 0; y
< size; y++)
            {
                for (int x = 0;
x < size; x++)
                {
                    float dx =
x - center;
                    float dy =
y - center;
                    float dist
= Mathf.Sqrt(dx * dx + dy *
dy);

                    float
outerAlpha                    =
Mathf.Clamp01((outerRadius   -
dist) / softEdge);
                    float
innerAlpha                    =
Mathf.Clamp01((dist          -
innerRadius) / softEdge);
                    float
alpha    =    outerAlpha    *
innerAlpha;

                    //    Soft
fill inside the ring so the
touch area feels solid.
                    float fill
= Mathf.Clamp01((innerRadius -
dist) / softEdge) * 0.25f;
                    alpha    =
Mathf.Max(alpha, fill);

                    pixels[y *
size + x] = new Color32(255,
255,   255,   (byte)(alpha   *
255f));
                }
            }


texture.SetPixels32(pixels);

texture.Apply(false, true);

            return
Sprite.Create(
                texture,
                new   Rect(0f,
0f, size, size),
                new
Vector2(0.5f, 0.5f),
                100f);
        }

        private   static   void
DestroyGeneratedSprite(ref
Sprite sprite)
        {
            if    (sprite    ==
null)
                return;

            Texture2D  texture
= sprite.texture;
            Destroy(sprite);
            if   (texture   !=
null)

Destroy(texture);

            sprite = null;
        }
    }
}
```

4. Modul Player (Kontrol Pemain)
Player/FirstPersonController.cs

```
using FokusTour.Input;
using UnityEngine;
using UnityEngine.InputSystem;

namespace FokusTour.Player
{
    /// <summary>
    /// First-person movement
for mobile (virtual joystick +
look area).
    ///         Keyboard/mouse
fallback   is   available   for
testing in the Editor.
    /// </summary>

[RequireComponent(typeof(Chara
cterController))]
    public               class
FirstPersonController        :
MonoBehaviour
    {
        [Header("References")]
        [SerializeField]
private    CharacterController
characterController;
        [SerializeField]
private              Transform
cameraTransform;
        [SerializeField]
```

```
private        VirtualJoystick
moveJoystick;
        [SerializeField]
private          TouchLookArea
lookArea;

        [Header("Movement")]
        [SerializeField]
private float moveSpeed = 4f;
        [SerializeField]
private float gravity = -15f;

        [Header("Look")]
        [SerializeField]
private  float  lookSensitivity
= 0.12f;
        [SerializeField]
private float mouseSensitivity
= 0.08f;
        [SerializeField]
private float minPitch = -80f;
        [SerializeField]
private float maxPitch = 80f;

        [Header("Editor
Fallback")]
        [SerializeField]
private                    bool
enableKeyboardMouseFallback  =
true;

        private float _pitch;
        private float _yaw;
        private          float
_verticalVelocity;

        private void Awake()
        {
            if
(characterController == null)

characterController          =
GetComponent<CharacterControll
er>();

            _yaw             =
transform.eulerAngles.y;
            if
(cameraTransform != null)
                _pitch       =
cameraTransform.localEulerAngl
es.x;
        }

        private void Update()
        {
            Vector2  moveInput
= ReadMoveInput();
            Vector2  lookInput
= ReadLookInput();


ApplyLook(lookInput);

ApplyMovement(moveInput);
        }

        private        Vector2
ReadMoveInput()
        {
            if   (moveJoystick
!=           null           &&
moveJoystick.Value.sqrMagnitud
e > 0.0001f)
                return
moveJoystick.Value;

            if
(!enableKeyboardMouseFallback)
                return
Vector2.zero;

            Keyboard  keyboard
= Keyboard.current;
            if   (keyboard   ==
null)
                return
Vector2.zero;

            Vector2   input   =
Vector2.zero;
            if
(keyboard.wKey.isPressed    ||
keyboard.upArrowKey.isPressed)
input.y += 1f;
            if
(keyboard.sKey.isPressed    ||
keyboard.downArrowKey.isPresse
d) input.y -= 1f;
            if
(keyboard.aKey.isPressed    ||
keyboard.leftArrowKey.isPresse
d) input.x -= 1f;
            if
(keyboard.dKey.isPressed    ||
keyboard.rightArrowKey.isPress
ed) input.x += 1f;
            return
Vector2.ClampMagnitude(input,
1f);
        }
```

```
        private        Vector2
ReadLookInput()
        {
            if   (lookArea   !=
null && lookArea.IsDragging)
                return
lookArea.ConsumeDelta()      *
lookSensitivity;

            if
(!enableKeyboardMouseFallback)
                return
Vector2.zero;

            Mouse    mouse    =
Mouse.current;
            if  (mouse  ==  null
||
!mouse.rightButton.isPressed)
                return
Vector2.zero;

            return
mouse.delta.ReadValue()      *
mouseSensitivity;
        }

        private           void
ApplyLook(Vector2 lookInput)
        {
            if
(lookInput.sqrMagnitude      <
0.0001f)
                return;

            _yaw            +=
lookInput.x;
            _pitch          -=
lookInput.y;
            _pitch           =
Mathf.Clamp(_pitch,  minPitch,
maxPitch);

            transform.rotation
=  Quaternion.Euler(0f,  _yaw,
0f);
            if
(cameraTransform != null)

cameraTransform.localRotation
= Quaternion.Euler(_pitch, 0f,
0f);
        }

        private           void
ApplyMovement(Vector2
moveInput)
        {
            Vector3   move   =
transform.right  *  moveInput.x
+     transform.forward     *
moveInput.y;
            move *= moveSpeed;

            if
(characterController.isGrounde
d && _verticalVelocity < 0f)

_verticalVelocity = -2f;

            _verticalVelocity
+= gravity * Time.deltaTime;
            move.y           =
_verticalVelocity;


characterController.Move(move
* Time.deltaTime);
        }
    }
}
```

Player/PlayerInteractor.cs

```
using FokusTour.Artwork;
using FokusTour.UI;
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.UI;

namespace FokusTour.Player
{
    /// <summary>
    /// Detects nearby artwork
and handles interact input (UI
button + keyboard E).
    /// </summary>
    public               class
PlayerInteractor             :
MonoBehaviour
    {
        [Header("Detection")]
        [SerializeField]
private              Transform
detectionOrigin;
        [SerializeField]
private float interactRange =
2.5f;
        [SerializeField]
private              LayerMask
detectionMask = ~0;

        [Header("UI")]
```

```
        [SerializeField]
private             GameObject
interactPrompt;
        [SerializeField]
private Button interactButton;
        [SerializeField]
private          ArtworkInfoUI
artworkInfoUI;

        [Header("Player")]
        [SerializeField]
private  FirstPersonController
firstPersonController;

        [Header("Editor
Fallback")]
        [SerializeField]
private                    bool
enableKeyboardFallback = true;

        private
ArtworkInteractable
_currentArtwork;
        private       readonly
Collider[]  _overlapResults  =
new Collider[16];

        private void Awake()
        {
            if
(detectionOrigin == null)

detectionOrigin = transform;

            if
(firstPersonController      ==
null)

firstPersonController        =
GetComponent<FirstPersonContro
ller>();


SetPromptVisible(false);

            if (interactButton
!= null)

interactButton.onClick.AddList
ener(TryInteract);
        }

        private           void
OnDestroy()
        {
            if (interactButton
!= null)

interactButton.onClick.RemoveL
istener(TryInteract);
        }

        private void Update()
        {
            if  (artworkInfoUI
!=           null           &&
artworkInfoUI.IsOpen)
            {

SetCurrentArtwork(null);
                return;
            }


UpdateClosestArtwork();

            if
(!enableKeyboardFallback)
                return;

            Keyboard  keyboard
= Keyboard.current;
            if   (keyboard   !=
null                          &&
keyboard.eKey.wasPressedThisFr
ame)
                TryInteract();
        }

        public            void
TryInteract()
        {
            if
(_currentArtwork  ==  null  ||
!_currentArtwork.HasData)
                return;

            if  (artworkInfoUI
== null)
                return;

            ArtworkItem item =
_currentArtwork.GetActiveItem(
);
            if (item == null)
                return;


artworkInfoUI.Open(item,
OnInfoClosed);

SetPromptVisible(false);
```

```
SetPlayerControlEnabled(false)
;
        }

        private           void
OnInfoClosed()
        {

SetPlayerControlEnabled(true);

UpdateClosestArtwork();
        }

        private           void
UpdateClosestArtwork()
        {
            Vector3  origin  =
detectionOrigin  !=  null  ?
detectionOrigin.position     :
transform.position;
            int    count    =
Physics.OverlapSphereNonAlloc(
                origin,
                interactRange,

_overlapResults,
                detectionMask,

QueryTriggerInteraction.Collid
e);


ArtworkInteractable  closest  =
null;
            float
closestDistance              =
float.MaxValue;

            for (int i = 0; i
< count; i++)
            {
                Collider hit =
_overlapResults[i];
                if   (hit   ==
null)
                    continue;


ArtworkInteractable artwork =
hit.GetComponentInParent<Artwo
rkInteractable>();
                if (artwork ==
null || !artwork.HasData)
                    continue;
                float distance
=     Vector3.Distance(origin,
artwork.transform.position);
                if (distance <
closestDistance)
                {

closestDistance = distance;
                    closest  =
artwork;
                }
            }


SetCurrentArtwork(closest);
        }

        private           void
SetCurrentArtwork(ArtworkInter
actable artwork)
        {
            if
(_currentArtwork == artwork)
                return;

            _currentArtwork  =
artwork;

SetPromptVisible(_currentArtwo
rk != null);
        }

        private           void
SetPromptVisible(bool visible)
        {
            if (interactPrompt
!= null)

interactPrompt.SetActive(visib
le);
        }

        private           void
SetPlayerControlEnabled(bool
enabled)
        {
            if
(firstPersonController      !=
null)

firstPersonController.enabled
= enabled;
        }

#if UNITY_EDITOR
        private           void
```

```
OnDrawGizmosSelected()
        {
            Vector3 origin = detectionOrigin != null ? detectionOrigin.position : transform.position;
            Gizmos.color = new Color(0.3f, 1f, 0.4f, 0.25f);

Gizmos.DrawWireSphere(origin, interactRange);
        }
#endif
    }
}
```

5. Modul UI (Antarmuka Pengguna)

UI/ArtworkInfoPanelLayout.cs

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace FokusTour.UI
{
    /// <summary>
    /// Styles the artwork info panel as a readable landscape card.
    /// Must sit on ArtworkUI together with ArtworkInfoUI.
    /// </summary>

[RequireComponent(typeof(ArtworkInfoUI))]
    public class ArtworkInfoPanelLayout : MonoBehaviour
    {
        [SerializeField] private ArtworkInfoUI artworkInfoUI;

        [Header("Card")]
        [SerializeField] private Vector2 cardSize = new Vector2(1200f, 640f);
        [SerializeField] private Color dimmerColor = new Color(0f, 0f, 0f, 0.55f);
        [SerializeField] private Color panelColor = new Color(0.09f, 0.10f, 0.12f, 0.96f);
        [SerializeField] private Color frameColor = new Color(1f, 1f, 1f, 0.14f);
        [SerializeField] private Color titleColor = Color.white;
        [SerializeField] private Color creatorColor = new Color(0.72f, 0.75f, 0.80f, 1f);
        [SerializeField] private Color descriptionColor = new Color(0.90f, 0.91f, 0.93f, 1f);
        [SerializeField] private Color closeButtonColor = new Color(0.95f, 0.95f, 0.96f, 1f);
        [SerializeField] private Color closeButtonTextColor = new Color(0.13f, 0.14f, 0.16f, 1f);

        private GameObject _dimmer;

        private void Awake()
        {
            if (artworkInfoUI == null)
                artworkInfoUI = GetComponent<ArtworkInfoUI>();

            Apply();

SetDimmerVisible(false);
        }

        public void Apply()
        {
            if (artworkInfoUI == null)
                artworkInfoUI = GetComponent<ArtworkInfoUI>();

            if (artworkInfoUI == null)
                return;


StretchOverlay(transform as RectTransform);
            EnsureDimmer();

SetDimmerVisible(true);
```

```
StylePanel(artworkInfoUI.Panel
Root);

StylePreview(artworkInfoUI.Pre
viewImage);

StyleTitle(artworkInfoUI.Title
Text);

StyleCreator(artworkInfoUI.Cre
atorText);

StyleDivider(artworkInfoUI.Pan
elRoot);

StyleDescription(artworkInfoUI
.DescriptionText);

StyleCloseButton(artworkInfoUI
.CloseButton);
        }

        public            void
SetDimmerVisible(bool visible)
        {
            if   (_dimmer   !=
null)

_dimmer.SetActive(visible);
        }

        private           void
EnsureDimmer()
        {
            Transform existing
=
transform.Find("ArtworkDimmer"
);
            if   (existing   ==
null)
            {
                _dimmer = new
GameObject("ArtworkDimmer",
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(Image));

_dimmer.transform.SetParent(tr
ansform, false);
            }
            else
            {
                _dimmer      =
existing.gameObject;
            }

_dimmer.transform.SetAsFirstSi
bling();

StretchOverlay(_dimmer.transfo
rm as RectTransform);

            Image    image    =
_dimmer.GetComponent<Image>();
            image.color      =
dimmerColor;

image.raycastTarget = true;
        }

        private   static   void
StretchOverlay(RectTransform
overlay)
        {
            if   (overlay   ==
null)
                return;

            overlay.anchorMin
= Vector2.zero;
            overlay.anchorMax
= Vector2.one;
            overlay.pivot    =
new Vector2(0.5f, 0.5f);

overlay.anchoredPosition     =
Vector2.zero;
            overlay.sizeDelta
= Vector2.zero;
            overlay.localScale
= Vector3.one;
            overlay.offsetMin
= Vector2.zero;
            overlay.offsetMax
= Vector2.zero;
        }

        private           void
StylePanel(GameObject
panelRoot)
        {
            if  (panelRoot  ==
null)
                return;

            RectTransform rect
=
panelRoot.GetComponent<RectTra
nsform>();
            if (rect != null)
```

```
            {
                rect.anchorMin
= new Vector2(0.5f, 0.5f);
                rect.anchorMax
= new Vector2(0.5f, 0.5f);
                rect.pivot   =
new Vector2(0.5f, 0.5f);

rect.anchoredPosition        =
Vector2.zero;
                rect.sizeDelta
= cardSize;

rect.localScale = Vector3.one;
            }

            Image   image   =
panelRoot.GetComponent<Image>(
);
            if (image != null)
            {
                image.color  =
panelColor;

image.raycastTarget = true;
            }
        }

        private           void
StylePreview(RawImage
previewImage)
        {
            if   (previewImage
== null)
                return;

            Vector2 box = new
Vector2(520f, 500f);
            Vector2   size   =
box;

            if
(previewImage.texture != null)
            {
                float aspect =
previewImage.texture.width   /
(float)previewImage.texture.he
ight;
                float
boxAspect = box.x / box.y;
                if  (aspect  >=
boxAspect)
                    size = new
Vector2(box.x,     box.x     /
aspect);
                else
                    size = new
Vector2(box.y     *     aspect,
box.y);
            }

            float  x  =  48f  +
(box.x - size.x) * 0.5f;
            float  y  =  -48f  -
(box.y - size.y) * 0.5f;

            Image    frame    =
EnsureChildImage(previewImage.
transform.parent,
"PreviewFrame");

SetTopLeft(frame.rectTransform
, x - 8f, y + 8f, size.x + 16f,
size.y + 16f);
            frame.color      =
frameColor;

frame.raycastTarget = false;

frame.transform.SetSiblingInde
x(0);

previewImage.transform.SetSibl
ingIndex(1);


SetTopLeft(previewImage.rectTr
ansform, x, y, size.x, size.y);
            previewImage.color
= Color.white;

previewImage.raycastTarget   =
false;
        }

        private           void
StyleTitle(TextMeshProUGUI
titleText)
        {
            if  (titleText  ==
null)
                return;


SetTopLeft(titleText.rectTrans
form, 604f, -48f, 548f, 80f);

ApplyTextStyle(titleText,
titleColor,               42f,
FontStyles.Bold,
TextAlignmentOptions.TopLeft);
```

```
titleText.enableAutoSizing = true;

titleText.fontSizeMin = 32f;

titleText.fontSizeMax = 42f;
        }

        private void StyleCreator(TextMeshProUGUI creatorText)
        {
            if (creatorText == null)
                return;

SetTopLeft(creatorText.rectTransform, 604f, -132f, 548f, 36f);

ApplyTextStyle(creatorText, creatorColor, 26f, FontStyles.Normal, TextAlignmentOptions.TopLeft);
        }

        private void StyleDivider(GameObject panelRoot)
        {
            if (panelRoot == null)
                return;

            Image divider = EnsureChildImage(panelRoot.transform, "ContentDivider");

SetTopLeft(divider.rectTransform, 604f, -176f, 548f, 2f);
            divider.color = new Color(1f, 1f, 1f, 0.16f);

divider.raycastTarget = false;
        }

        private void StyleDescription(TextMeshProUGUI descriptionText)
        {
            if (descriptionText == null)
                return;

SetTopLeft(descriptionText.rectTransform, 604f, -196f, 548f, 328f);

ApplyTextStyle(descriptionText, descriptionColor, 26f, FontStyles.Normal, TextAlignmentOptions.TopLeft);

descriptionText.lineSpacing = 12f;
        }

        private void StyleCloseButton(Button closeButton)
        {
            if (closeButton == null)
                return;

            RectTransform rect = closeButton.GetComponent<RectTransform>();
            rect.anchorMin = new Vector2(1f, 0f);
            rect.anchorMax = new Vector2(1f, 0f);
            rect.pivot = new Vector2(1f, 0f);

rect.anchoredPosition = new Vector2(-40f, 32f);
            rect.sizeDelta = new Vector2(168f, 52f);
            rect.localScale = Vector3.one;

            Image image = closeButton.GetComponent<Image>();
            if (image != null)
            {
                image.color = closeButtonColor;

image.raycastTarget = true;
            }

            TextMeshProUGUI label = closeButton.GetComponentInChildren<TextMeshProUGUI>(true);
            if (label != null)
            {
```

```
StretchOverlay(label.rectTransform);
                label.color = closeButtonTextColor;
                label.fontSize = 26f;

label.fontStyle             = FontStyles.Bold;

label.alignment             = TextAlignmentOptions.Center;

label.textWrappingMode      = TextWrappingModes.NoWrap;

label.raycastTarget = false;
            }
        }

        private static Image EnsureChildImage(Transform parent, string name)
        {
            Transform existing = parent.Find(name);
            GameObject go;
            if (existing == null)
            {
                go = new GameObject(name, typeof(RectTransform), typeof(CanvasRenderer), typeof(Image));

go.transform.SetParent(parent, false);
            }
            else
            {
                go = existing.gameObject;
            }

            return go.GetComponent<Image>();
        }

        private static void SetTopLeft(RectTransform rect, float x, float y, float width, float height)
        {
            rect.anchorMin = new Vector2(0f, 1f);
            rect.anchorMax = new Vector2(0f, 1f);
            rect.pivot = new Vector2(0f, 1f);

rect.anchoredPosition = new Vector2(x, y);
            rect.sizeDelta = new Vector2(width, height);
            rect.localScale = Vector3.one;
        }

        private static void ApplyTextStyle(
            TextMeshProUGUI text,
            Color color,
            float fontSize,
            FontStyles style,

TextAlignmentOptions alignment)
        {
            text.color = color;
            text.fontSize = fontSize;

text.enableAutoSizing = false;
            text.fontStyle = style;
            text.alignment = alignment;

text.textWrappingMode = TextWrappingModes.Normal;
            text.overflowMode = TextOverflowModes.Truncate;
            text.raycastTarget = false;
            text.margin = Vector4.zero;
        }
    }
}
```

UI/ArtworkInfoUI.cs

```
using System;
using FokusTour.Artwork;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace FokusTour.UI
```

```
{
    /// <summary>
    ///  Simple  mobile  panel
that shows artwork information
(TextMeshPro).
    /// </summary>
    public class ArtworkInfoUI
: MonoBehaviour
    {
        [Header("Panel")]
        [SerializeField]
private GameObject panelRoot;

        [Header("Content")]
        [SerializeField]
private        TextMeshProUGUI
titleText;
        [SerializeField]
private        TextMeshProUGUI
creatorText;
        [SerializeField]
private        TextMeshProUGUI
descriptionText;
        [SerializeField]
private RawImage previewImage;
        [SerializeField]
private Button closeButton;

        private         Action
_onClosed;
        private
ArtworkInfoPanelLayout
_layout;

        public  bool  IsOpen  {
get; private set; }

        public      GameObject
PanelRoot => panelRoot;
        public TextMeshProUGUI
TitleText => titleText;
        public TextMeshProUGUI
CreatorText => creatorText;
        public TextMeshProUGUI
DescriptionText             =>
descriptionText;
        public        RawImage
PreviewImage => previewImage;
        public          Button
CloseButton => closeButton;

        private void Awake()
        {
            if (closeButton !=
null)
closeButton.onClick.AddListene
r(Close);

            _layout          =
GetComponent<ArtworkInfoPanelL
ayout>();
            HideImmediate();
        }

        private           void
OnDestroy()
        {
            if (closeButton !=
null)

closeButton.onClick.RemoveList
ener(Close);
        }

        private void Update()
        {
            if (!IsOpen)
                return;

            var   keyboard   =
UnityEngine.InputSystem.Keyboa
rd.current;
            if   (keyboard   !=
null                          &&
keyboard.escapeKey.wasPressedT
hisFrame)
                Close();
        }

        public            void
Open(ArtworkItem  item,  Action
onClosed = null)
        {
            if (item == null ||
!item.HasMetadata)
                return;

            _onClosed        =
onClosed;

            if  (titleText  !=
null)
                titleText.text
= item.Title;

            if (creatorText !=
null)

creatorText.text             =
$"Fotografer:
{item.CreatorName}";
```

```
            if
(descriptionText != null)

descriptionText.text         =
item.Description;

            if   (previewImage
!= null)
            {
                bool  hasImage
= item.PreviewTexture != null;

previewImage.gameObject.SetAct
ive(hasImage);
                if (hasImage)

previewImage.texture         =
item.PreviewTexture;
            }

            if  (panelRoot  !=
null)

panelRoot.SetActive(true);

            if   (_layout   ==
null)
                _layout      =
GetComponent<ArtworkInfoPanelL
ayout>();

            _layout?.Apply();
            IsOpen = true;
        }

        public void Close()
        {
            if (!IsOpen)
                return;

            HideImmediate();

            Action  callback  =
_onClosed;
            _onClosed = null;

callback?.Invoke();
        }

        private           void
HideImmediate()
        {
            IsOpen = false;

            if  (panelRoot  !=
null)

panelRoot.SetActive(false);


_layout?.SetDimmerVisible(fals
e);
        }
    }
}
```

UI/GalleryBackToMenuButton.cs

```
using FokusTour.Api;
using UnityEngine;
using
UnityEngine.SceneManagement;
using UnityEngine.UI;

namespace FokusTour.UI
{
    /// <summary>
    ///  Runtime  exit  icon
button  in  the  gallery  that
returns to MainMenuScene.
    /// Place on Canvas or any
active object in MainScene.
    /// </summary>
    public               class
GalleryBackToMenuButton      :
MonoBehaviour
    {
        [SerializeField]
private                 string
mainMenuSceneName            =
"MainMenuScene";
        [SerializeField]
private          ArtworkInfoUI
artworkInfoUI;
        [SerializeField]
private  Vector2  buttonSize  =
new Vector2(80f, 80f);
        [SerializeField]
private Vector2 topRightOffset
= new Vector2(-72f, -32f);
        [SerializeField]
private Color buttonColor = new
Color(0.09f,   0.10f,   0.12f,
0.92f);
        [SerializeField]
private   Color   iconColor   =
Color.white;
        [SerializeField]
private  float  iconPadding  =
14f;

        private     GameObject
```

```
_buttonRoot;
        private         Sprite
_roundedSprite;
        private         Sprite
_exitIconSprite;

        private void Awake()
        {
            if  (artworkInfoUI
== null)
                artworkInfoUI
=
FindFirstObjectByType<ArtworkI
nfoUI>();

            _roundedSprite   =
CreateRoundedSprite(128, 32f);
            _exitIconSprite  =
CreateExitIconSprite(128);
            BuildButton();
        }

        private void Update()
        {
            if (_buttonRoot ==
null)
                return;

            bool hideForInfo =
artworkInfoUI   !=   null   &&
artworkInfoUI.IsOpen;
            if
(_buttonRoot.activeSelf     ==
hideForInfo)

_buttonRoot.SetActive(!hideFor
Info);
        }

        private           void
OnDestroy()
        {

DestroyGeneratedSprite(ref
_roundedSprite);

DestroyGeneratedSprite(ref
_exitIconSprite);
        }

        private           void
BuildButton()
        {
            Canvas   canvas   =
GetComponentInParent<Canvas>()
;
            if    (canvas    ==
null)
                canvas       =
FindFirstObjectByType<Canvas>(
);

            if    (canvas    ==
null)
            {

Debug.LogWarning("GalleryBackT
oMenuButton:    Canvas    tidak
ditemukan.");
                return;
            }

            _buttonRoot = new
GameObject("BackToMenuButton",
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(Image),
typeof(Button));

_buttonRoot.transform.SetParen
t(canvas.transform, false);

_buttonRoot.transform.SetAsLas
tSibling();

            RectTransform rect
=
_buttonRoot.GetComponent<RectT
ransform>();
            rect.anchorMin   =
new Vector2(1f, 1f);
            rect.anchorMax   =
new Vector2(1f, 1f);
            rect.pivot  =  new
Vector2(1f, 1f);

rect.anchoredPosition        =
topRightOffset;
            rect.sizeDelta   =
buttonSize;

            Image background =
_buttonRoot.GetComponent<Image
>();
            if (_roundedSprite
!= null)
            {

background.sprite            =
_roundedSprite;

background.type              =
```

```
Image.Type.Sliced;

background.pixelsPerUnitMultiplier = 1.2f;
            }
            background.color = buttonColor;

            Button button = _buttonRoot.GetComponent<Button>();

button.targetGraphic = background;

button.onClick.AddListener(GoToMainMenu);

            GameObject iconObject = new GameObject("ExitIcon", typeof(RectTransform), typeof(CanvasRenderer), typeof(Image));

iconObject.transform.SetParent(_buttonRoot.transform, false);

            RectTransform iconRect = iconObject.GetComponent<RectTransform>();
            iconRect.anchorMin = Vector2.zero;
            iconRect.anchorMax = Vector2.one;
            iconRect.offsetMin = new Vector2(iconPadding, iconPadding);
            iconRect.offsetMax = new Vector2(-iconPadding, -iconPadding);

            Image icon = iconObject.GetComponent<Image>();
            icon.sprite = _exitIconSprite;
            icon.type = Image.Type.Simple;

icon.preserveAspect = true;
            icon.color = iconColor;
            icon.raycastTarget = false;
        }

        private void GoToMainMenu()
        {
            if (ArtworkSessionCache.Instance != null)
            {

ArtworkSessionCache.Instance.Clear();

Destroy(ArtworkSessionCache.Instance.gameObject);
            }


SceneManager.LoadScene(mainMenuSceneName);
        }

        private static Sprite CreateExitIconSprite(int size)
        {
            Texture2D texture = new Texture2D(size, size, TextureFormat.RGBA32, false)
            {
                name = "ExitIconTexture",
                filterMode = FilterMode.Bilinear,
                wrapMode = TextureWrapMode.Clamp,
                hideFlags = HideFlags.HideAndDontSave
            };

            Color32[] pixels = new Color32[size * size];
            float stroke = size * 0.08f;

            // Door/frame on the left.
            DrawRect(pixels, size, size * 0.18f, size * 0.20f, size * 0.18f, size * 0.60f, stroke);
            // Open side (right of door) stays empty so arrow can leave.

            // Arrow shaft.
```

```
            DrawLine(pixels,
size, size * 0.42f, size *
0.50f, size * 0.78f, size *
0.50f, stroke);
            // Arrow head.
            DrawLine(pixels,
size, size * 0.62f, size *
0.34f, size * 0.80f, size *
0.50f, stroke);
            DrawLine(pixels,
size, size * 0.62f, size *
0.66f, size * 0.80f, size *
0.50f, stroke);


texture.SetPixels32(pixels);

texture.Apply(false, true);

            return
Sprite.Create(
                texture,
                new Rect(0f,
0f, size, size),
                new
Vector2(0.5f, 0.5f),
                100f);
        }

        private static void
DrawRect(Color32[] pixels, int
size, float x, float y, float
width, float height, float
stroke)
        {
            DrawLine(pixels,
size, x, y, x + width, y,
stroke);
            DrawLine(pixels,
size, x, y + height, x + width,
y + height, stroke);
            DrawLine(pixels,
size, x, y, x, y + height,
stroke);
            DrawLine(pixels,
size, x + width, y, x + width,
y + height, stroke);
        }

        private static void
DrawLine(Color32[] pixels, int
size, float x0, float y0, float
x1, float y1, float thickness)
        {
            float dx = x1 - x0;
            float dy = y1 - y0;
            float length =
Mathf.Max(1f, Mathf.Sqrt(dx *
dx + dy * dy));
            int steps =
Mathf.CeilToInt(length * 2f);

            for (int i = 0; i
<= steps; i++)
            {
                float t = i /
(float)steps;
                float x =
Mathf.Lerp(x0, x1, t);
                float y =
Mathf.Lerp(y0, y1, t);

StampCircle(pixels, size, x,
y, thickness * 0.5f);
            }
        }

        private static void
StampCircle(Color32[] pixels,
int size, float cx, float cy,
float radius)
        {
            int minX =
Mathf.Max(0,
Mathf.FloorToInt(cx - radius -
1f));
            int maxX =
Mathf.Min(size - 1,
Mathf.CeilToInt(cx + radius +
1f));
            int minY =
Mathf.Max(0,
Mathf.FloorToInt(cy - radius -
1f));
            int maxY =
Mathf.Min(size - 1,
Mathf.CeilToInt(cy + radius +
1f));

            for (int y = minY;
y <= maxY; y++)
            {
                for (int x =
minX; x <= maxX; x++)
                {
                    float dx =
x - cx;
                    float dy =
y - cy;
                    float dist
= Mathf.Sqrt(dx * dx + dy *
dy);
```

```
                    float
alpha = Mathf.Clamp01((radius
- dist) / 1.2f);
                    if (alpha
<= 0f)

continue;

                    int index
= y * size + x;
                    byte value
= (byte)(alpha * 255f);
                    if (value
> pixels[index].a)

pixels[index] = new
Color32(255, 255, 255, value);
                }
            }
        }

        private static Sprite
CreateRoundedSprite(int size,
float cornerRadiusPixels)
        {
            Texture2D texture
= new Texture2D(size, size,
TextureFormat.RGBA32, false)
            {
                name =
"GalleryMenuButtonTexture",
                filterMode =
FilterMode.Bilinear,
                wrapMode =
TextureWrapMode.Clamp,
                hideFlags =
HideFlags.HideAndDontSave
            };

            Color32[] pixels =
new Color32[size * size];
            float radius =
Mathf.Clamp(cornerRadiusPixels
, 4f, size * 0.45f);
            float softEdge =
1.25f;

            for (int y = 0; y
< size; y++)
            {
                for (int x = 0;
x < size; x++)
                {
                    float
alpha = RoundedRectAlpha(x, y,
size, size, radius, softEdge);
                    pixels[y *
size + x] = new Color32(255,
255, 255, (byte)(alpha *
255f));
                }
            }

texture.SetPixels32(pixels);

texture.Apply(false, true);

            int border =
Mathf.CeilToInt(radius +
softEdge + 1f);
            return
Sprite.Create(
                texture,
                new Rect(0f,
0f, size, size),
                new
Vector2(0.5f, 0.5f),
                100f,
                0,

SpriteMeshType.FullRect,
                new
Vector4(border, border,
border, border));
        }

        private static float
RoundedRectAlpha(int x, int y,
int width, int height, float
radius, float softEdge)
        {
            float left =
radius;
            float right =
width - 1 - radius;
            float bottom =
radius;
            float top = height
- 1 - radius;

            float px = x;
            float py = y;
            float dx = 0f;
            float dy = 0f;

            if (px < left)
                dx = left - px;
            else if (px >
right)
                dx = px -
right;
```

```
            if (py < bottom)
                dy = bottom -
py;
            else if (py > top)
                dy = py - top;

            if (dx <= 0f && dy
<= 0f)
                return 1f;

            float    dist    =
Mathf.Sqrt(dx * dx + dy * dy);
            return
Mathf.Clamp01((radius  -  dist)
/ softEdge);
        }

        private  static  void
DestroyGeneratedSprite(ref
Sprite sprite)
        {
            if    (sprite    ==
null)
                return;

            Texture2D  texture
= sprite.texture;
            Destroy(sprite);
            if   (texture   !=
null)

Destroy(texture);
            sprite = null;
        }
    }
}
```

UI/InteractPromptLayout.cs

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

namespace FokusTour.UI
{
    /// <summary>
    ///   Styles   the   "Lihat
Karya"   prompt   button   for
landscape mobile.
    ///    Place    this    on
InteractPrompt.
    /// </summary>
    public               class
InteractPromptLayout         :
MonoBehaviour
    {
        [SerializeField]
private Button interactButton;
        [SerializeField]
private  Vector2  buttonSize  =
new Vector2(280f, 72f);
        [SerializeField]
private  float  bottomOffset  =
110f;
        [SerializeField]
private Color buttonColor = new
Color(0.09f,   0.10f,   0.12f,
0.94f);
        [SerializeField]
private  Color  labelColor  =
Color.white;
        [SerializeField]
private float fontSize = 28f;

        private void Awake()
        {
            Apply();
        }

        private           void
OnEnable()
        {
            Apply();
        }

        public void Apply()
        {
            if (interactButton
== null)
                interactButton
=
GetComponentInChildren<Button>
(true);


StretchOverlay(transform    as
RectTransform);

StyleButton(interactButton);
        }

        private  static  void
StretchOverlay(RectTransform
overlay)
        {
            if   (overlay   ==
null)
                return;

            overlay.anchorMin
= Vector2.zero;
            overlay.anchorMax
```

```
= Vector2.one;
            overlay.pivot    =
new Vector2(0.5f, 0.5f);

overlay.anchoredPosition     =
Vector2.zero;
            overlay.sizeDelta
= Vector2.zero;
            overlay.localScale
= Vector3.one;
            overlay.offsetMin
= Vector2.zero;
            overlay.offsetMax
= Vector2.zero;
        }

        private           void
StyleButton(Button button)
        {
            if    (button    ==
null)
                return;

            RectTransform rect
=
button.GetComponent<RectTransf
orm>();
            rect.anchorMin   =
new Vector2(0.5f, 0f);
            rect.anchorMax   =
new Vector2(0.5f, 0f);
            rect.pivot  =  new
Vector2(0.5f, 0f);

rect.anchoredPosition   =   new
Vector2(0f, bottomOffset);
            rect.sizeDelta   =
buttonSize;
            rect.localScale  =
Vector3.one;

            Image    image    =
button.GetComponent<Image>();
            if (image != null)
            {
                image.color  =
buttonColor;

image.raycastTarget = true;
            }

            TextMeshProUGUI
label                        =
button.GetComponentInChildren<
TextMeshProUGUI>(true);
            if (label == null)
                return;

StretchOverlay(label.rectTrans
form);
            label.text       =
"Lihat Karya";
            label.color      =
labelColor;
            label.fontSize   =
fontSize;
            label.fontStyle  =
FontStyles.Bold;
            label.alignment  =
TextAlignmentOptions.Center;

label.textWrappingMode       =
TextWrappingModes.NoWrap;

label.enableAutoSizing       =
false;

label.raycastTarget = false;
        }
    }
}
```

UI/MainMenuUI.cs

```
using System.Collections;
using FokusTour.Api;
using TMPro;
using UnityEngine;
using
UnityEngine.EventSystems;
using
UnityEngine.InputSystem.UI;
using
UnityEngine.SceneManagement;
using UnityEngine.UI;

namespace FokusTour.UI
{
    /// <summary>
    /// Builds the entire Main
Menu UI at runtime (landscape
mobile).
    ///  Attach  to  an  empty
GameObject in MainMenuScene.
    /// </summary>
    public class MainMenuUI :
MonoBehaviour
    {
        [Header("Scene")]
        [SerializeField]
private                  string
galleryScene Name            =
```

```
"MainScene";

        [Header("API")]
        [Tooltip("Preferred API URL. App tries HTTPS first, then HTTP.")]
        [SerializeField] private string apiUrl = ApiEndpoint.HttpsApiUrl;

        [Header("Content")]
        [SerializeField] [TextArea(4, 12)]
        private string sejarahText =
            "PROFIL UKM FOTOGRAFI KAMPUS UMPAR\n\n" +
            "1. Nama Organisasi\n" +
            "Unit Kegiatan Mahasiswa Fotografi Kampus Universitas Muhammadiyah Parepare, " +
            "disingkat UKM FOKUS UMPAR.\n\n" +
            "2. Maksud, Tujuan, Visi dan Misi\n" +
            "Mengembangkan potensi mahasiswa pada bidang Fotografi, Videografi, dan Pers, " +
            "baik secara praktis maupun teknis, agar bermanfaat bagi intra maupun ekstra kampus, " +
            "serta mengacu pada visi dan misi Universitas demi terwujudnya mahasiswa yang kreatif " +
            "dan kompetitif dalam ipteks.\n\n" +
            "Visi dan misi UKM Fotografi Kampus meliputi:\n" +
            "a. Menghasilkan model pengembangan kreativitas mahasiswa di bidang Fotografi, " +
            "Videografi, dan Pers yang terencana, terprogram, dan berkelanjutan.\n" +
            "b. Meningkatkan relevansi kegiatan mahasiswa untuk memperkuat komitmen, semangat, " +
            "kemandirian, dan ketangguhan.\n" +
            "c. Meningkatkan kerja sama dan kemitraan dengan dunia usaha agar lulusan memiliki " +
            "daya saing di dunia kerja.\n\n" +
            "3. Sejarah Singkat\n" +
            "Pencetusan UKM Fotografi Kampus UMPAR diawali diskusi lepas pada bulan Ramadhan, " +
            "15 Agustus 2010, oleh mahasiswa dari berbagai fakultas, antara lain Ridwan Syam, " +
            "Ade Riswan, Kaharuddin Hanun, Fathuddin Djollong, Ali Wira Rahman, dan sejumlah " +
            "mahasiswa lainnya.\n\n" +
            "Dari diskusi tersebut muncul keinginan mendirikan UKM sebagai wadah kompetitif " +
            "di bidang Fotografi, Videografi, dan Pers. Pada rapat paripurna 24 Agustus 2010 " +
            "dibentuk struktur pengurus dan dibahas AD/ART. Selanjutnya disusun proposal " +
            "pendirian, dikumpulkan dukungan mahasiswa antar fakultas, lalu diajukan melalui " +
            "BEM UMPAR kepada pimpinan universitas untuk melegalkan organisasi.\n\n" +
            "4. Atribut Logo UKM FOKUS UMPAR\n" +
            "a. Sinar Matahari Biru\n" +
            "Melambangkan sinar sebagai simbol Perserikatan Muhammadiyah.\n\n" +
            "b. Lingkaran Biru\n" +
            "Melambangkan Ikatan Mahasiswa Universitas Muhammadiyah Parepare.\n\n" +
```

```
            "c.      Lingkaran
Merah\n" +
            "Lingkaran
berwarna merah dengan dasar
hitam       di       dalamnya,
melambangkan " +
            "wujud gerakan dan
tujuan UKM FOKUS UMPAR.\n\n" +
            "Sumber:
ukmfokusumpar.blogspot.com";

        [SerializeField]
[TextArea(4, 12)]
        private         string
tentangPembuatText =
            "Aplikasi
FokusTour dikembangkan sebagai
tugas akhir / skripsi " +
            "Program     Studi
Teknik Informatika.\n\n" +
            "Judul:\n" +
            "Rancang    Bangun
Virtual   Tour   Hasil   Karya
Fotografi UKM Fokus " +
            "Universitas
Muhammadiyah Parepare\n\n" +
            "Pembuat:\n" +
            "Chaerul  Anwar\n"
+
            "NIM.
219280182\n\n" +
            "Aplikasi      ini
menampilkan     hasil     karya
fotografi   UKM   Fokus   dalam
bentuk " +
            "virtual      tour
berbasis 3D, sehingga pengguna
dapat menjelajahi ruang " +
            "pameran   virtual
dan  melihat  informasi  tiap
karya.";

        [Header("Look")]
        [SerializeField]
private Color backgroundColor
=  new  Color(0.06f,  0.07f,
0.09f, 1f);
        [SerializeField]
private Color cardColor = new
Color(0.10f,   0.11f,   0.14f,
0.96f);
        [SerializeField]
private Color buttonColor = new
Color(0.16f,   0.18f,   0.22f,
1f);
        [SerializeField]
private                  Color
primaryButtonColor    =    new
Color(0.92f,   0.93f,   0.95f,
1f);
        [SerializeField]
private  Color  titleColor  =
Color.white;
        [SerializeField]
private Color subtitleColor =
new Color(0.72f, 0.75f, 0.80f,
1f);
        [SerializeField]
private Color buttonLabelColor
= Color.white;
        [SerializeField]
private                  Color
primaryButtonLabelColor = new
Color(0.12f,   0.13f,   0.15f,
1f);
        [SerializeField]
private float cardCornerRadius
= 28f;
        [SerializeField]
private                   float
buttonCornerRadius = 18f;

        private     GameObject
_infoPanel;
        private
TextMeshProUGUI _infoTitle;
        private
TextMeshProUGUI _infoBody;
        private       RawImage
_infoLogo;
        private  RectTransform
_infoContent;
        private         Sprite
_roundedCardSprite;
        private         Sprite
_roundedButtonSprite;
        private     GameObject
_loadingPanel;
        private
TextMeshProUGUI
_loadingStatusText;
        private         Button
_startTourButton;
        private           bool
_isStartingTour;
        private  const  float
LogoSize = 180f;
        private  const  float
LogoSpacing = 24f;

        private void Awake()
        {
```

```
            EnsureCamera();
EnsureEventSystem();
            _roundedCardSprite
=     CreateRoundedSprite(128,
40f);

_roundedButtonSprite         =
CreateRoundedSprite(128, 32f);
            BuildUI();
        }

        private           void
OnDestroy()
        {

DestroyGeneratedSprite(ref
_roundedCardSprite);

DestroyGeneratedSprite(ref
_roundedButtonSprite);
        }

        private           void
EnsureCamera()
        {
            if (Camera.main !=
null)
                return;

            GameObject
cameraObject       =       new
GameObject("Main Camera");
            Camera   camera   =
cameraObject.AddComponent<Came
ra>();
            camera.clearFlags
= CameraClearFlags.SolidColor;

camera.backgroundColor       =
backgroundColor;
            cameraObject.tag =
"MainCamera";

cameraObject.AddComponent<Audi
oListener>();
        }

        private           void
EnsureEventSystem()
        {
            if
(FindFirstObjectByType<EventSy
stem>() != null)
                return;

            GameObject
eventSystem       =       new
GameObject("EventSystem");

eventSystem.AddComponent<Event
System>();

eventSystem.AddComponent<Input
SystemUIInputModule>();
        }

        private void BuildUI()
        {
            GameObject
canvasObject       =       new
GameObject("MainMenuCanvas",
typeof(RectTransform));
            Canvas   canvas   =
canvasObject.AddComponent<Canv
as>();
            canvas.renderMode
=
RenderMode.ScreenSpaceOverlay;

canvas.sortingOrder = 10;

            CanvasScaler
scaler                       =
canvasObject.AddComponent<Canv
asScaler>();
            scaler.uiScaleMode
=
CanvasScaler.ScaleMode.ScaleWi
thScreenSize;

scaler.referenceResolution   =
new Vector2(1920f, 1080f);

scaler.matchWidthOrHeight    =
0.5f;


canvasObject.AddComponent<Grap
hicRaycaster>();


CreateBackground(canvasObject.
transform);

CreateMenuCard(canvasObject.tr
ansform);

CreateInfoPanel(canvasObject.t
ransform);

CreateLoadingPanel(canvasObjec
```

```
t.transform);
        }

        private           void
CreateBackground(Transform
parent)
        {
            Image background =
CreateImage("Background",
parent);

StretchFull(background.rectTra
nsform);
            background.color =
backgroundColor;

background.raycastTarget     =
false;
        }

        private           void
CreateMenuCard(Transform
parent)
        {
            Image    card    =
CreateImage("MenuCard",
parent);
            RectTransform
cardRect = card.rectTransform;
            cardRect.anchorMin
= new Vector2(0.5f, 0.5f);
            cardRect.anchorMax
= new Vector2(0.5f, 0.5f);
            cardRect.pivot   =
new Vector2(0.5f, 0.5f);
            cardRect.sizeDelta
= new Vector2(760f, 620f);

cardRect.anchoredPosition    =
Vector2.zero;

ApplyRoundedSprite(card,
_roundedCardSprite, cardColor,
1.15f);

            TextMeshProUGUI
title   =   CreateText("Title",
card.transform,   "FokusTour",
54f,          FontStyles.Bold,
titleColor);

SetTopCenter(title.rectTransfo
rm, 0f, -48f, 680f, 70f);

            TextMeshProUGUI
subtitle = CreateText(
                "Subtitle",

card.transform,
                "Virtual  Tour
Karya      Fotografi      UKM
Fokus\nUniversitas
Muhammadiyah Parepare",
                22f,

FontStyles.Normal,

subtitleColor);

SetTopCenter(subtitle.rectTran
sform, 0f, -118f, 680f, 56f);

            float buttonY = -
210f;
            float buttonGap =
88f;
            Vector2 buttonSize
= new Vector2(420f, 72f);

            _startTourButton =
CreateMenuButton(card.transfor
m, "Mulai Tour", buttonSize,
buttonY,   primaryButtonColor,
primaryButtonLabelColor,
StartTour);

CreateMenuButton(card.transfor
m,   "Sejarah   UKM   Fokus",
buttonSize,     buttonY     -
buttonGap,        buttonColor,
buttonLabelColor,
ShowSejarah);

CreateMenuButton(card.transfor
m,     "Tentang     Pembuat",
buttonSize,     buttonY     -
buttonGap * 2f, buttonColor,
buttonLabelColor,
ShowTentangPembuat);

CreateMenuButton(card.transfor
m,    "Keluar",    buttonSize,
buttonY - buttonGap * 3f,
buttonColor, buttonLabelColor,
QuitApp);
        }

        private           void
CreateLoadingPanel(Transform
parent)
        {
            Image   dimmer   =
```

```
CreateImage("LoadingDimmer",
parent);

StretchFull(dimmer.rectTransfo
rm);
            dimmer.color = new
Color(0f, 0f, 0f, 0.72f);

dimmer.raycastTarget = true;

            Image    card    =
CreateImage("LoadingCard",
dimmer.transform);
            RectTransform
cardRect = card.rectTransform;
            cardRect.anchorMin
= new Vector2(0.5f, 0.5f);
            cardRect.anchorMax
= new Vector2(0.5f, 0.5f);
            cardRect.pivot   =
new Vector2(0.5f, 0.5f);
            cardRect.sizeDelta
= new Vector2(640f, 260f);

ApplyRoundedSprite(card,
_roundedCardSprite, cardColor,
1.15f);

            TextMeshProUGUI
title                          =
CreateText("LoadingTitle",
card.transform, "Memuat Tour",
36f,          FontStyles.Bold,
titleColor);

SetTopCenter(title.rectTransfo
rm, 0f, -36f, 560f, 48f);

            _loadingStatusText
= CreateText(

"LoadingStatus",

card.transform,
                "Memuat   data
karya...",
                24f,

FontStyles.Normal,

subtitleColor);

SetTopCenter(_loadingStatusTex
t.rectTransform,   0f,   -110f,
560f, 80f);

_loadingStatusText.textWrappin
gMode                          =
TextWrappingModes.Normal;


CreateMenuButton(card.transfor
m, "Batal", new Vector2(180f,
52f),     0f,     buttonColor,
buttonLabelColor,
CancelStartTour, true);

            _loadingPanel    =
dimmer.gameObject;

_loadingPanel.SetActive(false)
;
        }

        private           void
CreateInfoPanel(Transform
parent)
        {
            Image   dimmer   =
CreateImage("InfoDimmer",
parent);

StretchFull(dimmer.rectTransfo
rm);
            dimmer.color = new
Color(0f, 0f, 0f, 0.55f);

            Image   panel    =
CreateImage("InfoPanel",
dimmer.transform);
            RectTransform
panelRect                      =
panel.rectTransform;

panelRect.anchorMin    =    new
Vector2(0.04f, 0.05f);

panelRect.anchorMax    =    new
Vector2(0.96f, 0.95f);

panelRect.offsetMin          =
Vector2.zero;

panelRect.offsetMax          =
Vector2.zero;
            panelRect.pivot  =
new Vector2(0.5f, 0.5f);

ApplyRoundedSprite(panel,
_roundedCardSprite, cardColor,
1.15f);
```

```
            _infoTitle      =
CreateText("InfoTitle",
panel.transform, "Judul", 42f,
FontStyles.Bold, titleColor);

SetTopCenter(_infoTitle.rectTr
ansform, 0f, -44f, 1400f, 56f);

_infoTitle.alignment        =
TextAlignmentOptions.Center;

            GameObject
viewportObject      =      new
GameObject("InfoViewport",
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(Image),
typeof(RectMask2D));

viewportObject.transform.SetPa
rent(panel.transform, false);
            RectTransform
viewportRect                =
viewportObject.GetComponent<Re
ctTransform>();

viewportRect.anchorMin  =  new
Vector2(0f, 0f);

viewportRect.anchorMax  =  new
Vector2(1f, 1f);

viewportRect.offsetMin  =  new
Vector2(56f, 110f);

viewportRect.offsetMax  =  new
Vector2(-56f, -118f);

viewportObject.GetComponent<Im
age>().color  =  new  Color(1f,
1f, 1f, 0.001f);

            GameObject
contentObject      =      new
GameObject("InfoContent",
typeof(RectTransform));

contentObject.transform.SetPar
ent(viewportObject.transform,
false);
            _infoContent     =
contentObject.GetComponent<Rec
tTransform>();

_infoContent.anchorMin  =  new
Vector2(0f, 1f);

_infoContent.anchorMax  =  new
Vector2(1f, 1f);
            _infoContent.pivot
= new Vector2(0.5f, 1f);

_infoContent.anchoredPosition
= Vector2.zero;

_infoContent.sizeDelta  =  new
Vector2(0f, 1200f);

            GameObject
logoObject        =        new
GameObject("InfoLogo",
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(RawImage));

logoObject.transform.SetParent
(contentObject.transform,
false);
            _infoLogo        =
logoObject.GetComponent<RawIma
ge>();
            RectTransform
logoRect                    =
_infoLogo.rectTransform;
            logoRect.anchorMin
= new Vector2(0.5f, 1f);
            logoRect.anchorMax
= new Vector2(0.5f, 1f);
            logoRect.pivot   =
new Vector2(0.5f, 1f);

logoRect.anchoredPosition    =
new Vector2(0f, -8f);
            logoRect.sizeDelta
=    new    Vector2(LogoSize,
LogoSize);
            _infoLogo.color  =
Color.white;

_infoLogo.raycastTarget      =
false;
            _infoLogo.texture
=
Resources.Load<Texture2D>("Log
o_UKM_FOKUS");

_infoLogo.gameObject.SetActive
(false);

            _infoBody        =
CreateText("InfoBody",
contentObject.transform,
```

```
"Isi", 28f, FontStyles.Normal,
subtitleColor);
            RectTransform
bodyRect                        =
_infoBody.rectTransform;
            bodyRect.anchorMin
= new Vector2(0f, 1f);
            bodyRect.anchorMax
= new Vector2(1f, 1f);
            bodyRect.pivot    =
new Vector2(0.5f, 1f);

bodyRect.anchoredPosition     =
new  Vector2(0f,  -(LogoSize  +
LogoSpacing));
            bodyRect.sizeDelta
= new Vector2(0f, 800f);

_infoBody.alignment           =
TextAlignmentOptions.Top;

_infoBody.horizontalAlignment
=
HorizontalAlignmentOptions.Cen
ter;

_infoBody.verticalAlignment   =
VerticalAlignmentOptions.Top;

_infoBody.textWrappingMode    =
TextWrappingModes.Normal;

_infoBody.overflowMode        =
TextOverflowModes.Overflow;

_infoBody.lineSpacing = 12f;

            ScrollRect  scroll
=
panel.gameObject.AddComponent<
ScrollRect>();
            scroll.viewport   =
viewportRect;
            scroll.content    =
_infoContent;
            scroll.horizontal
= false;
            scroll.vertical   =
true;

scroll.movementType           =
ScrollRect.MovementType.Clampe
d;

scroll.scrollSensitivity      =
45f;

CreateMenuButton(panel.transfo
rm, "Tutup", new Vector2(200f,
60f),  0f,  primaryButtonColor,
primaryButtonLabelColor,
HideInfoPanel, true);

            _infoPanel        =
dimmer.gameObject;

_infoPanel.SetActive(false);
        }

        private           void
RefreshInfoContentHeight(bool
showLogo)
        {
            if  (_infoBody  ==
null || _infoContent == null)
                return;

            float  logoBlock  =
showLogo    ?    LogoSize    +
LogoSpacing : 0f;
            RectTransform
bodyRect                        =
_infoBody.rectTransform;

bodyRect.anchoredPosition     =
new Vector2(0f, -logoBlock);

            float    width    =
_infoContent.rect.width;
            if (width < 10f)
                width = 1400f;

            Vector2  preferred
=
_infoBody.GetPreferredValues(_
infoBody.text, width, 0f);
            float bodyHeight =
Mathf.Max(preferred.y  +  40f,
400f);
            bodyRect.sizeDelta
= new Vector2(0f, bodyHeight);

_infoContent.sizeDelta  =  new
Vector2(0f,     logoBlock     +
bodyHeight);

_infoContent.anchoredPosition
= Vector2.zero;
        }

        private         Button
```

```
CreateMenuButton(
            Transform parent,
            string label,
            Vector2 size,
            float anchoredY,
            Color background,
            Color labelColor,

UnityEngine.Events.UnityAction
onClick,
            bool bottomRight =
false)
        {
            GameObject
buttonObject       =       new
GameObject(label  +  "Button",
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(Image),
typeof(Button));

buttonObject.transform.SetPare
nt(parent, false);

            RectTransform rect
=
buttonObject.GetComponent<Rect
Transform>();
            if (bottomRight)
            {
                rect.anchorMin
= new Vector2(1f, 0f);
                rect.anchorMax
= new Vector2(1f, 0f);
                rect.pivot   =
new Vector2(1f, 0f);

rect.anchoredPosition   =   new
Vector2(-40f, 32f);
            }
            else
            {
                rect.anchorMin
= new Vector2(0.5f, 1f);
                rect.anchorMax
= new Vector2(0.5f, 1f);
                rect.pivot   =
new Vector2(0.5f, 1f);

rect.anchoredPosition   =   new
Vector2(0f, anchoredY);
            }

            rect.sizeDelta   =
size;

            Image    image    =
buttonObject.GetComponent<Imag
e>();

ApplyRoundedSprite(image,
_roundedButtonSprite,
background, 1.25f);

            Button   button   =
buttonObject.GetComponent<Butt
on>();

button.targetGraphic = image;

button.onClick.AddListener(onC
lick);

            TextMeshProUGUI
text   =   CreateText(label   +
"Label",
buttonObject.transform, label,
26f,          FontStyles.Bold,
labelColor);

StretchFull(text.rectTransform
);
            text.alignment   =
TextAlignmentOptions.Center;
            text.raycastTarget
= false;
            return button;
        }

        private           void
StartTour()
        {
            if
(_isStartingTour)
                return;


StartCoroutine(StartTourRoutin
e());
        }

        private    IEnumerator
StartTourRoutine()
        {
            _isStartingTour  =
true;
            if
(_startTourButton != null)

_startTourButton.interactable
= false;
```

```
ShowLoadingPanel("Menghubungkan ke server...");

            ArtworkSessionCache cache = ArtworkSessionCache.EnsureExists();
            yield return cache.Prefetch(apiUrl, SetLoadingStatus);

            if (!_isStartingTour)
                yield break;

            if (!cache.IsReady)
            {
                string error = string.IsNullOrEmpty(cache.LastError)
                    ? "Gagal memuat data karya."
                    : cache.LastError;

                SetLoadingStatus(error + "\nPeriksa koneksi / server API, lalu coba lagi.");
                yield return new WaitForSecondsRealtime(2.2f);

                HideLoadingPanel();

                _isStartingTour = false;
                if (_startTourButton != null)

                    _startTourButton.interactable = true;
                yield break;
            }

            SetLoadingStatus("Membuka tour...");
            yield return null;

            SceneManager.LoadScene(gallerySceneName);
        }

        private void CancelStartTour()
        {
            if (!_isStartingTour)
            {
                HideLoadingPanel();
                return;
            }

            StopAllCoroutines();

            ArtworkSessionCache.Instance?.Clear();

            HideLoadingPanel();
            _isStartingTour = false;
            if (_startTourButton != null)

                _startTourButton.interactable = true;
        }

        private void ShowLoadingPanel(string status)
        {
            if (_loadingPanel != null)

                _loadingPanel.SetActive(true);


            SetLoadingStatus(status);
        }

        private void HideLoadingPanel()
        {
            if (_loadingPanel != null)

                _loadingPanel.SetActive(false);
        }

        private void SetLoadingStatus(string status)
        {
            if (_loadingStatusText != null)
```

```
_loadingStatusText.text = status;
        }

        private void ShowSejarah()
        {
            ShowInfoPanel("Sejarah UKM Fokus", sejarahText, true);
        }

        private void ShowTentangPembuat()
        {
            ShowInfoPanel("Tentang Pembuat", tentangPembuatText, false);
        }

        private void ShowInfoPanel(string title, string body, bool showLogo)
        {
            if (_infoPanel == null)
                return;

            _infoTitle.text = title;
            _infoBody.text = body;

            if (_infoLogo != null)
            {
                if (_infoLogo.texture == null)
                    _infoLogo.texture = Resources.Load<Texture2D>("Logo_UKM_FOKUS");

                _infoLogo.gameObject.SetActive(showLogo && _infoLogo.texture != null);
            }

            _infoPanel.SetActive(true);
            Canvas.ForceUpdateCanvases();
            RefreshInfoContentHeight(showLogo && _infoLogo != null && _infoLogo.texture != null);
        }

        private void HideInfoPanel()
        {
            if (_infoPanel != null)
                _infoPanel.SetActive(false);
        }

        private void QuitApp()
        {
#if UNITY_EDITOR
            UnityEditor.EditorApplication.isPlaying = false;
#else
            Application.Quit();
#endif
        }

        private static void ApplyRoundedSprite(Image image, Sprite sprite, Color color, float pixelsPerUnitMultiplier = 1f)
        {
            if (image == null)
                return;

            if (sprite != null)
            {
                image.sprite = sprite;
                image.type = Image.Type.Sliced;
                image.pixelsPerUnitMultiplier = pixelsPerUnitMultiplier;
                image.useSpriteMesh = false;
            }

            image.color = color;
        }

        private static Sprite CreateRoundedSprite(int size, float cornerRadiusPixels)
        {
```

```
            Texture2D texture
= new Texture2D(size, size,
TextureFormat.RGBA32, false)
            {
                name         =
"RoundedUITexture",
                filterMode   =
FilterMode.Bilinear,
                wrapMode     =
TextureWrapMode.Clamp,
                hideFlags    =
HideFlags.HideAndDontSave
            };

            Color32[] pixels =
new Color32[size * size];
            float   radius   =
Mathf.Clamp(cornerRadiusPixels
, 4f, size * 0.45f);
            float  softEdge  =
1.25f;

            for (int y = 0; y
< size; y++)
            {
                for (int x = 0;
x < size; x++)
                {
                    float
alpha = RoundedRectAlpha(x, y,
size, size, radius, softEdge);
                    pixels[y *
size + x] = new Color32(255,
255,   255,   (byte)(alpha   *
255f));
                }
            }


texture.SetPixels32(pixels);

texture.Apply(false, true);

            int    border    =
Mathf.CeilToInt(radius       +
softEdge + 1f);
            return
Sprite.Create(
                texture,
                new   Rect(0f,
0f, size, size),
                new
Vector2(0.5f, 0.5f),
                100f,
                0,
SpriteMeshType.FullRect,
                new
Vector4(border,        border,
border, border));
        }

        private  static  float
RoundedRectAlpha(int x, int y,
int  width,  int  height,  float
radius, float softEdge)
        {
            float    left    =
radius;
            float   right   =
width - 1 - radius;
            float   bottom   =
radius;
            float top = height
- 1 - radius;

            float px = x;
            float py = y;
            float dx = 0f;
            float dy = 0f;

            if (px < left)
                dx = left - px;
            else   if   (px   >
right)
                dx   =   px   -
right;

            if (py < bottom)
                dy = bottom -
py;
            else if (py > top)
                dy = py - top;

            if (dx <= 0f && dy
<= 0f)
                return 1f;

            float    dist    =
Mathf.Sqrt(dx * dx + dy * dy);
            return
Mathf.Clamp01((radius  -  dist)
/ softEdge);
        }

        private  static  void
DestroyGeneratedSprite(ref
Sprite sprite)
        {
            if    (sprite    ==
null)
                return;
```

```
            Texture2D texture
= sprite.texture;
            Destroy(sprite);
            if (texture !=
null)

Destroy(texture);

            sprite = null;
        }

        private static Image
CreateImage(string name,
Transform parent)
        {
            GameObject go =
new GameObject(name,
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(Image));

go.transform.SetParent(parent,
false);
            return
go.GetComponent<Image>();
        }

        private static
TextMeshProUGUI CreateText(
            string name,
            Transform parent,
            string content,
            float fontSize,
            FontStyles style,
            Color color)
        {
            GameObject go =
new GameObject(name,
typeof(RectTransform),
typeof(CanvasRenderer),
typeof(TextMeshProUGUI));

go.transform.SetParent(parent,
false);

            TextMeshProUGUI
text =
go.GetComponent<TextMeshProUGU
I>();
            text.text =
content;
            text.fontSize =
fontSize;
            text.fontStyle =
style;
            text.color =
color;
            text.alignment =
TextAlignmentOptions.Center;

text.textWrappingMode =
TextWrappingModes.Normal;
            text.raycastTarget
= false;
            return text;
        }
        private static void
StretchFull(RectTransform
rect)
        {
            rect.anchorMin =
Vector2.zero;
            rect.anchorMax =
Vector2.one;
            rect.offsetMin =
Vector2.zero;
            rect.offsetMax =
Vector2.zero;
            rect.localScale =
Vector3.one;
        }
        private static void
SetTopCenter(RectTransform
rect, float x, float y, float
width, float height)
        {
            rect.anchorMin =
new Vector2(0.5f, 1f);
            rect.anchorMax =
new Vector2(0.5f, 1f);
            rect.pivot = new
Vector2(0.5f, 1f);

rect.anchoredPosition = new
Vector2(x, y);
            rect.sizeDelta =
new Vector2(width, height);
        }
        private static void
SetTopLeft(RectTransform rect,
float x, float y, float width,
float height)
        {
            rect.anchorMin =
new Vector2(0f, 1f);
            rect.anchorMax =
new Vector2(0f, 1f);
            rect.pivot = new
Vector2(0f, 1f);

rect.anchoredPosition = new
```

```
Vector2(x, y);                          new Vector2(width, height);
            rect.sizeDelta    =
        }
    }
}
```

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

PROPOSAL

Mahasiswa : CHAERUL ANWAR	Pembimbing I : Wahyuddin, S.Kom.,M.Kom.
NIM : 219280182	Pembimbing II : Andi Wafiah, S.Kom., M.Kom
RANCANG BANGUN VIRTUAL TOUR HASIL KARYA FOTOGRAFI UKM FOKUS UNIVERSITAS MUHAMMADIYAH PAREPARE	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Acc		Konsultasi 1 Perbaiki sistem diselesaikan	
Konsultasi 2		Konsultasi 2 Acc Proposal	
Konsultasi 3		Konsultasi 3	
Konsultasi 4		Konsultasi 4	
Konsultasi		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

HASIL

Mahasiswa : CHAERUL ANWAR	Pembimbing I : Wahyuddin, S.Kom.,M.Kom.
NIM : 219280182	Pembimbing II : Andi Wafiah, S.Kom., M.Kom
RANCANG BANGUN VIRTUAL TOUR HASIL KARYA FOTOGRAFI UKM FOKUS UNIVERSITAS MUHAMMADIYAH PAREPARE	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Acc		Konsultasi 1 Revisi sdh diselesaikan	
Konsultasi 2		Konsultasi 2 Acc ujian hasil	
Konsultasi 3		Konsultasi 3	
Konsultasi 4		Konsultasi 4	
Konsultasi		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

SKRIPSI

Mahasiswa : CHAERUL ANWAR	Pembimbing I : Wahyuddin, S.Kom.,M.Kom.
NIM : 219280182	Pembimbing II : Andi Wafiah, S.Kom., M.Kom
RANCANG BANGUN VIRTUAL TOUR HASIL KARYA FOTOGRAFI UKM FOKUS UNIVERSITAS MUHAMMADIYAH PAREPARE	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Acc		Konsultasi 1 Revisi sdh diselesaikan	
Konsultasi 2		Konsultasi 2 Acc ujian tutup	
Konsultasi 3		Konsultasi 3	
Konsultasi 4		Konsultasi 4	
Konsultasi		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali