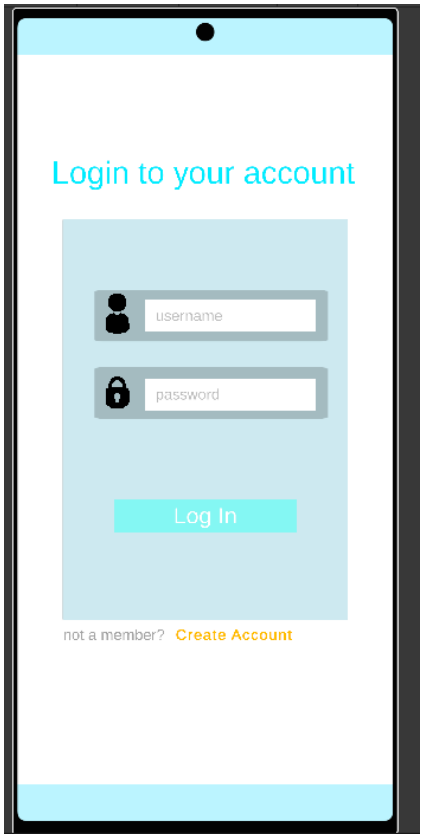
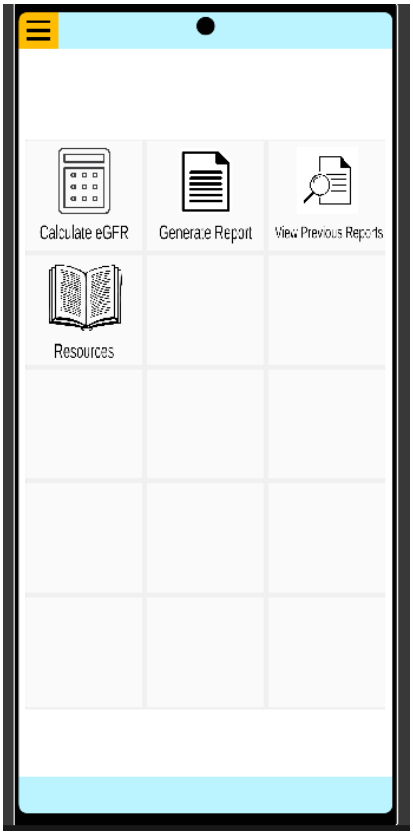
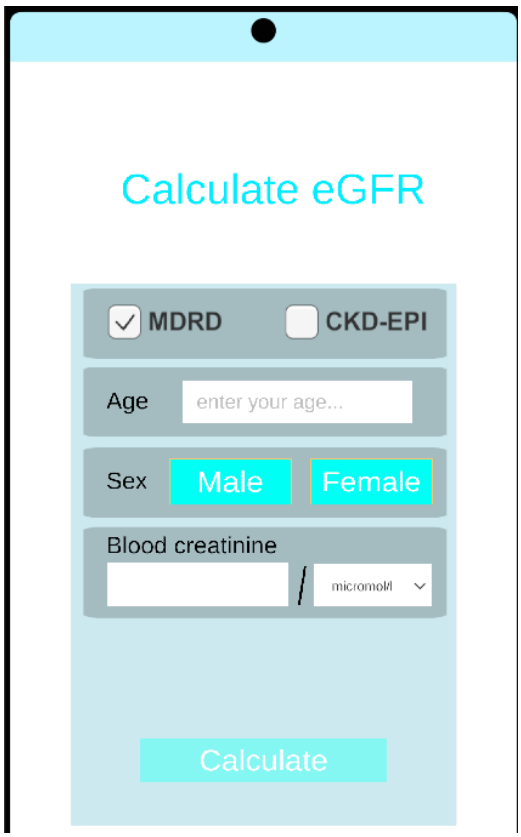
Iteration 1

The first portion of iteration 1 was spent developing the UX/UI layout and design for the main areas of the app before any actual functionality was put into place. These being the login screen, the main area of the app where users can select what they want to do, and then finally the eGFR calculator itself.







After that focus was divided into two separate areas, one on the front-end functionality working with the UI and actual elements that the user will interact, and the backend dealing with the database and all the interactions it may have with the front end. At first the database had many vulnerabilities with public variables and exposed pieces of data in other portions of the code, mostly in the login system where the database needed to be linked to check the inputted ids and passwords. But as more was learnt about connecting the database to Unity, these vulnerabilities were patched out and encapsulated only inside of the one class that has direct access to the database.

```
public class DatabaseManager : MonoBehaviour
{
    private string apiUrl = "https://apex.oracle.com/pls/apex/ckd/Nephro"; // Replace with your API URL
    1 reference
    public LoginData activeData { get; set; }
    private JSONData activeTable;
    private JSONData patientTable;
    private JSONData clinicianTable;
    8 references
    public static DatabaseManager instance { get; private set; }
```

```
Unity Script (1 asset reference) | 12 references
public class DatabaseManager : MonoBehaviour
{
    private string apiUrl = "https://apex.oracle.com/pls/apex/ckd/Nephro"; // Replace with your API URL
    private LoginData activeData;
    private JSONData activeTable;
    private JSONData patientTable;
    private JSONData clinicianTable;
    13 references
    public static DatabaseManager instance { get; private set; }
```

```
else if (userIdField.text.Contains("H1") || userIdField.text.Contains("2000"))
{
    IdUser = DatabaseManager.instance.FindItemById(userIdField.text);
    passUser = DatabaseManager.instance.FindItemByPassword(passwordField.text);
    Debug.Log("id and password");
    CheckData(IdUser, passUser);
}

else
{
    IdUser = DatabaseManager.instance.FindItemByUsername(userIdField.text);
    passUser = DatabaseManager.instance.FindItemByPassword(passwordField.text);
    Debug.Log("username and password");
    CheckData(IdUser, passUser);
}
```

```
else
{
    Debug.Log("match found");
    DatabaseManager.instance.activeData = IdUser;
    if (DatabaseManager.instance.GetCurrentTable() == "patients") SceneManager.LoadScene("PatientInterface");
    else SceneManager.LoadScene("ClinicianInterface");
}
```

```
2 references
public LoginData FindItemByUsername(string searchName)
{
    LoginData found = activeTable.items.Find(client => client.username.Contains(searchName));
    LoginData item = found;
    return item;
}

0 references
public LoginData FindItemByUsername(string searchName, string searchPassword)
{
    LoginData found = activeTable.items.Find(client => client.username.Contains(searchName) && client.password == searchPassword);
    LoginData item = found;
    return item;
}

2 references
public LoginData FindItemByPassword(string searchPassword)
{
    LoginData found = activeTable.items.Find(client => client.password == searchPassword);
    LoginData item = found;
    return item;
}
```

```
public bool IsIdInTable(string searchId)
{
    LoginData item = FindItemById(searchId);
    if (item == null) return false;
    else return true;
}

1 reference
public bool IsUsernameInTable(string searchName)
{
    LoginData item = FindItemByUsername(searchName);
    if (item == null) return false;
    else return true;
}

2 references
public bool IsPasswordInTable(string password)
{
    LoginData item = FindItemByPassword(password);
    if (item == null) return false;
    else return true;
}

1 reference
public bool DoesPasswordMatchUser(string username, string password)
{
    LoginData passItem = FindItemByPassword(password);
    LoginData nameItem = FindItemByUsername(username);
    if (passItem == nameItem) return true;
    else return false;
}

1 reference
public bool DoesPasswordMatchId(string id, string password)
{
    LoginData passItem = FindItemByPassword(password);
    LoginData idItem = FindItemById(id);
    if (passItem == idItem) return true;
    else return false;
}
```

After most of the general functionality left was for the logins and the MDRD calculator, development was dedicated to creating other calculators like the paediatric eGFR calculator for under 18s and the CKD-EPI calculator for other patients that may have different needs. Throughout this process testing was being done, which exposed a few features and points that could be implemented in the next iteration.

```
//TODO: put full GetPeGFR version and call it with instance on PeGFRCalculator class
1 reference
public float GetFullPeEGFR(float height, float creatinine, float cystatinC, float bun, bool isMale, bool isMGDL)
{
    if (!isMGDL) creatinine /= 88.4f; // Convert µmol/L to mg/dL if necessary
    float genderFactor = isMale ? 1.076f : 1f;
    float eGFR = 39.8f * Mathf.Pow(height / creatinine, 0.456f) *
    Mathf.Pow(1.8f / cystatinC, 0.418f) *
    Mathf.Pow(30f / bun, 0.079f) * genderFactor *
    Mathf.Pow(height / 1.4f, 0.179f);

    return eGFR;
}

//TODO: put CKD-EPI version and call it with instance on EGFRCalculator class
1 reference
public float GetCKDEPI(int age, float creatinine, bool isMale, bool isMGDL)
{
    if (!isMGDL) creatinine /= 88.4f; // Convert µmol/L to mg/dL if needed

    float K = isMale ? 0.9f : 0.7f;      // k coefficient 0.9 for males, 0.7 for females
    float alpha = isMale ? -0.302f : -0.241f; // alpha coefficient -0.302 for males, -0.241 for females

    // min and max serum creatinine (Scr) values in mg/dL
    float minScr = Mathf.Min(creatinine / K, 1);
    float maxScr = Mathf.Max(creatinine / K, 1);

    float eGFR = 142 * Mathf.Pow(minScr, alpha) * Mathf.Pow(maxScr, -1.200f) * Mathf.Pow(0.9938f, age);
    if (!isMale) eGFR *= 1.012f; // if female ending factor multiplier

    return eGFR;
}
```

Iteration 2

This iteration was focused on general quality of life features and adding additional security. One of the first features was the CSV file upload, allowing if logged in as a clinician to upload a csv file (with the correct formatting) to the app and batch calculate the eGFR for all patients in the file (barring any errors in values that would be highlighted per entry if so).

Following this was the addition of a few security features. The first of which was a lockout system for the login page. Where after a certain number of invalid tries the app would lock you out from trying again for a certain amount of time. Followed by an unfinished addition of password hashing, where each password would be hashed using SHA256, an irreversible process that makes the password unreadable unless the exact string or password is hashed again to compare, as hashing is deterministic.

Iteration	Key Focus	Features Delivered	Security Improvements
1	Core UI/UX, login system, database integration	Login page, main menu, MDRD calculator	Encapsulation of public variables and improved Unity-Oracle connection handling
2	Quality of life, data handling, security	CSV batch upload, paediatric calculator, CKD-EPI calculator	Login lockout system, SHA-256 password hashing

Each iteration closed with a review session and refactoring sprint to ensure that lessons from testing and security analysis directly informed subsequent development cycles.

```
public void SelectFile()
{
    NativeFilePicker.Permission persimssion = NativeFilePicker.PickFile(async (path) =>
    {
        if (!string.IsNullOrEmpty(path) && Path.GetExtension(path).ToLower() == ".csv")
        {
            resultText.text = string.Empty;
            try
            {
                Debug.Log("CSV file selected: " + path);
                filePath = path;
                List<eGFRData> fileContent = await ParseCSV();
                if (!resultCanvas.isActiveAndEnabled) resultCanvas.gameObject.SetActive(true);
                int elementCount = 1;


                foreach (eGFRData data in fileContent)
                {
                    if (elementCount == 11 || fileContent[fileContent.Count - 1] == data)
                    {
                        pages.Add(resultText.text);
                        resultText.text = string.Empty;
                        elementCount = 1;
                    }
                    FormatText(data);
                    elementCount++;
                }
                OnFormatComplete?.Invoke(pages);
                Debug.Log(transform.root.gameObject.name);
            }

            catch (Exception ex)
            {

                Debug.LogError("Error: fileContent is empty. CSV might not be parsed correctly.");
                resultText.text = "Please select a valid csv file with the correct formatting";
                if (!resultCanvas.isActiveAndEnabled) resultCanvas.gameObject.SetActive(true);
                if (resultCanvas.gameObject != transform.root.gameObject) transform.root.gameObject.SetActive(false);
            }
```

```
if (!lockoutStarted && incorrectCount >= maxIncorrect)
{
    lockoutStarted = true;
    userIdField.gameObject.SetActive(false);
    userIdField.text = string.Empty;
    passwordField.gameObject.SetActive(false);
    passwordField.text = string.Empty;
    await Task.Delay(lockoutSeconds * 1000);
    incorrectCount = 0;
    lockoutStarted = false;
    userIdField.gameObject.SetActive(true);
    passwordField.gameObject.SetActive(true);
}
```

```
private string HashData(string data)
{
    using (SHA256 sha256 = SHA256.Create())
    {
        byte[] hashedBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(data));
        return BitConverter.ToString(hashedBytes).Replace("-", "").ToLower(); // Convert to HEX
    }
}
```