Droeftoeters-ICT – Login Functionaliteit

Inlever datum: 04/04/2025

# 1 – Informatie

## 1.1 - Studenten

Christian Scott Horler - 2227807

Lilah Sanchez - 2233692

Martijn van Houwelingen - 2225486

Sophie Iersel - 2232301

Thomas Koman – 2226974

## 2.2 – Project

Groepsnaam: Droeftoeters

Projectnaam: Droeftoeters-ICT

# 2 - Inhoud

Contents

[1 – Informatie 1](#_Toc194664214)

[1.1 - Studenten 1](#_Toc194664215)

[2.2 – Project 1](#_Toc194664216)

[2 - Inhoud 1](#_Toc194664217)

[3 - Inleiding 2](#_Toc194664218)

[4 – Samenvatting 2](#_Toc194664219)

[5 – Diagrammen 3](#_Toc194664220)

# 3 - Inleiding

Dit document gaat vooral over informatie voor de programmeur. Met deze informatie zou je makkelijk de software kunnen uitbereiden. Dit document is daarom ook niet helemaal netjes geformatteerd als je vanuit het oogpunt kijkt van een product owner, dit is puur zodat programmeurs het begrijpen.

Dit document gaat over alles wat met login en registratie te maken heeft aan de front-end unity kant.

Verder is dit document geconvert van markdown naar word, dus als er hier en daar nog ` vliegende comma’s te zien zijn komt dat hierdoor.

# 4 – Samenvatting

De logica bestaat uit 2 Scenes, de FunctionalLogin Scene, en de ChildCreator scene. Ze gebruiken bijde het zelfde script. Het LoginScript.cs. Hier is voor gekozen omdat ze bijde grotendeels de zelfde functionaliteit gebruiken, en er geen tijd was om een child klasse aan te maken voor het kinder registratie pagina.

Verder heeft elke pagina het MainManager gameobject + script. En aan dit object hangt ook het apiconnector.cs. Dit hebben we allemaal nodig voor de loginscript.

# 5 – Diagrammen

Ik heb mijn best gedaan om in een flowchart weer te geven hoe het in de frontend zit met de logica. Ik heb het moeten opknippen in 2 foto’s omdat de canvas te groot is. Als je het helemaal wilt bekijken, ga dan naar Droeftoeters-ICT\documentation\Feature\login and register\Login Flowchart.canvas. Je kan deze canvas bekijken met de hulp van het programma [Obsidian](https://obsidian.md/).

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# 6 – In Depth Uitleg

## 6.1 - LoginScript Class

Dit script hendelt de validatie van de invoer, het afhandelen van de login request, en register request.

Bij startup checkt ie met MainManager script en api connector script of dat je bent ingelogd (hiervoor moet nog een endpoint komen op de api)

Als je niet bent ingelogd, haalt ie gelijk een nieuwe session token op als dat mogenlijk is waardoor je toch automatisch ingelogd wordt.

ClickButton(string) is hoe de buttons communiceren met de logica, Vul 1 van de volgende waarden in:

- ParentRegister

- ParentLogin

- ChildLogin

Als je iets anders invult wordt er een error gelogd wanneer je klikt op de knop.

SetPasswordValue(string) en SetUsernameValue(string) is hoe de invoer velden aangeven aan de logica wat de value is van de 2 velden, dit gebeurt bij de on value changed event van de input field.

Als je een 2e password veld hebt, dan gebruik je SetSecondPasswordValue(string), dit 2e wachtwoord veld is voor bij het registreren om te kijken of de wachtwoorden het zelfde zijn.

Er zijn heel veel public fields zoals parentRegisterErrorMessageLabel en childLoginErrorMessageLabel en parentRegisterUsernameField en parentLoginUsernameField etc. Deze fields zijn er zodat alle functionaliteit werkt op de elementen die geegeven zijn, aangezien alles werkt met de zelfde script, maar het niet allemaal in de zelfde scene zit.

### 6.1.1 – Het kinder account aanmaken

Het kinder account aanmaken is heel scuffed, we doen eerst 1 query om het aan te maken, en daarna doen we er nog 3 om het te koppelen, als 1 van die 3 faalt hebben we een ghost kinder account die niet gekoppeld is aan een ouder maar wel bestaat. We weten dat dit niet zo hoort, en dat we dit nooit in productie zouden doen, maar we hadden geen tijd om onze eigen registratie endpoint te maken. Vandaar dat we hiervoor hebben gekozen.

## 6.2 – MainManager Class

dit script zorgt ervoor dat data bewaard blijft als je switcht tussen scenes, hiervan bestaat er max 1. Dit wordt nu gebruikt zodat je vanaf elke scene een AUTH request kan sturen naar de api.

Verder haalt dit ook de login credentials op aan het begin uit je bestanden, en onthoudt hij die gegevens.

LoginDataSaveLocation is het pad waar de login van de user wordt opgeslagen.

Verder gebruiken we de mainmanager ook om te onthouden op welke pagina we waren als we opnieuw moeten inloggen, zodat je daarna teruggesturd kan worden.

## 6.3 - ApiConnecter Class

`public string baseUrl = "";` voor de correcte api address.

`public string defaultLoginScene = "TestFunctionalLogin";` is om aan te geven welke scene de login scene is.

/// <summary>

/// The method that connects the frontend with the backend.

/// </summary>

/// <param name="path">The path that you put behind the baseurl of the api.</param>

/// <param name="protocol">POST GET PUT or DELETE</param>

/// <param name="authorized">wether you need to be logged in for this endpoint or not.</param>

/// <param name="callback">the method that will get called with the result of the api request.</param>

/// <param name="body">the body that you provide for an PUT and POST request. JSON string.</param>

/// <param name="autoLogin">whether we send the user to the login page if we get an unauthorized error. If enabled it will try to auto login the user again.</param>

/// <returns></returns>

public IEnumerator SendRequest(string path, HttpMethod protocol, bool authorized, Action<string, string> callback, string body = "", bool autoLogin = true) {}

Hierboven zie je de methode die we aanroepen om een request te sturen, deze heb ik gemaakt om het process te versimplenen. Deze robuste methode zou voor elk scenario bruikbaar moeten zijn, en het handelt opnieuw inloggen automatisch af als dat nodig is.

Hier zien we een voorbeeld van de implementatie.

apiConnecter.SendRequest("account/checkAccessToken", HttpMethod.GET, true, (string response, string error) =>

{

if (error == null)

{

Debug.Log("yay");

}

else {

Debug.Log("errorrrrr");

}

}, "", false);

# 7 - Support Classes Etc

Dit gaat over classes die niet de main zijn, maar wel gebruikt worden in dit systeem.

## 7.1 – LoginResponse

Dit wordt gebruikt om op te slaan je data waarmee we automatisch weer kunnen inloggen.

public class LoginResponse

{

public string tokenType { get; set; }

public string accessToken { get; set; }

public int expiresIn { get; set; }

public string refreshToken { get; set; }

}

## 7.2 – Validator

Met deze simpele klasse kunnen we valideren of een email of wachtwoord voldoet aan onze eisen, hier is het voorbeeld met ingeklapte methodes:

public class Validator

{

public static bool IsValidPassword(string password) {}

public static bool IsValidEmail(string email) {}

}