*Bachelor Thesis*  
Leuphana Universität Lüneburg  
Major Wirtschaftsinformatik

Analyse, Entwurf und prototypische Realisierung einer Flutter-Anwendung mit Web3-Authentifizierungsverfahren und Autorisierung durch ERC-721 Token am Beispiel eines Minigames

Abschlussarbeit

zur Erlangung des akademischen Grades

|  |  |
| --- | --- |
| 1. Gutachter: | Prof. Dr. Guido Barbian |
| 2. Gutachter: | Prof. Dr. Jan Wilk |
| vorgelegt von: | Anton Stamme |
| vorgelegt am: | 15. 09. 2022 |

**Bachelor of Science (B.Sc.)**

# Abstract/Zusammenfassung

**Willkommen** und vielen Dank, dass Sie sich für diese Dokumentvorlage entschieden haben.

# Inhaltsverzeichnis

[Abstract/Zusammenfassung I](#_Toc108718747)

[Inhaltsverzeichnis II](#_Toc108718748)

[Abkürzungsverzeichnis IV](#_Toc108718749)

[Tabellenverzeichnis V](#_Toc108718750)

[Abbildungsverzeichnis VI](#_Toc108718751)

[1 Einführung 1](#_Toc108718752)

[1.1 Motivation 1](#_Toc108718753)

[1.2 Zielsetzung und Aufgabenstellung 1](#_Toc108718754)

[1.3 Aufbau und Arbeitsweise 1](#_Toc108718755)

[2 Grundlagen 2](#_Toc108718756)

[2.1 Blockchain Technologie 2](#_Toc108718757)

[2.1.1 Crypto-Accounts und -Wallets 2](#_Toc108718758)

[2.1.2 Smart Contracts 2](#_Toc108718759)

[2.1.3 NFTs 2](#_Toc108718760)

[2.1.4 Node Provider 2](#_Toc108718761)

[2.1.5 IPFS 2](#_Toc108718762)

[2.2 Ethereum 2](#_Toc108718763)

[2.2.1 ERC-721 2](#_Toc108718764)

[2.2.2 Solidity 2](#_Toc108718765)

[2.3 IT-Sicherheit 2](#_Toc108718766)

[2.3.1 Web3 Authentifizierung 2](#_Toc108718767)

[2.3.2 Autorisierung mittels NFTs 2](#_Toc108718768)

[2.4 Cross-Platform-Entwicklung mit Flutter 2](#_Toc108718769)

[3. Systemanalyse 3](#_Toc108718770)

[3.1 Flappy Bird 3](#_Toc108718771)

[3.1.1 Geschichte 3](#_Toc108718772)

[3.1.2 Konzept 4](#_Toc108718773)

[3.1.3 NFTs in Flappy Bird 4](#_Toc108718774)

[3.2 Anforderungsanalyse 5](#_Toc108718775)

[3.2.1 Smart Contract 6](#_Toc108718776)

[3.2.2 Flutter Anwendung 6](#_Toc108718777)

[4. Systementwurf 8](#_Toc108718778)

[4.1 Architektur 9](#_Toc108718779)

[4.2 Smart Contract 11](#_Toc108718780)

[4.3 Flutter App 13](#_Toc108718781)

[5. Prototypische Realisierung 16](#_Toc108718782)

[5.1 Smart Contracts 16](#_Toc108718783)

[5.1.1 Ganache 16](#_Toc108718784)

[5.1.2 Skins 16](#_Toc108718785)

[5.1.3 Perks 16](#_Toc108718786)

[5.2 Flutter App 16](#_Toc108718787)

[5.2.1 Flutter Bird 16](#_Toc108718788)

[5.2.2 Web3-Authentifizierung 16](#_Toc108718789)

[5.2.3 Autorisierung mit NFTs 16](#_Toc108718790)

[6. Vorführung der Anwendung 17](#_Toc108718791)

[7. Bewertung des entwickelten Prototyps 18](#_Toc108718792)

[8. Fazit 19](#_Toc108718793)

[Literaturverzeichnis 20](#_Toc108718794)

[Literaturverzeichnis als sortierbare Tabelle 20](#_Toc108718795)

[Standard-Literaturverzeichnis 20](#_Toc108718796)

[Index 21](#_Toc108718797)

[Eidesstattliche Erklärung 22](#_Toc108718798)

# Abkürzungsverzeichnis

Ein Abkürzungsverzeichnis ist nicht für jede Arbeit notwendig. Es enthält nur fachspezifische Abkürzungen, die nicht im Duden aufgeführt bzw. allgemein gebräuchlich sind.

Die Abkürzungen stehen hier in einer unsichtbaren Tabelle ohne Rahmen (Tabellenvorlage: Unsichtbare Tabelle). Mehr dazu weiter unten.

|  |  |
| --- | --- |
| Abk. 1: | Beschreibung |
| Abk. 2: | Beschreibung |
| Abk. 3: | Beschreibung |

IPFS

CID

HTTP

ETH

RPC

ERC

EIP

NFT

MVC

# Tabellenverzeichnis

[Tabelle 1: Zusätzliche Anforderungen an den Flutter Bird Skins Smart Contract 6](#_Toc108372171)

[Tabelle 2: Anforderungen an die Flutter Anwendung 7](#_Toc108372172)

# Abbildungsverzeichnis

[Abbildung 1: FlappyBird Screenshot 4](#_Toc108704622)

[Abbildung 2: Darstellung der Kernfunktionen von Flutter Bird durch ein Use Case Diagramm 7](#_Toc108704623)

[Abbildung 3: Verteilungsdiagramm des Gesamtsystems 10](#_Toc108704624)

[Abbildung 4: Klassendiagramm des Smart Contracts und der OpenZeppelin Implementierungen 12](#_Toc108704625)

[Abbildung 5: Vereinfachtes Klassendiagramm zur Entwurfslpanung der Flutter App 14](#_Toc108704626)

**Hinweis:** Auf diesen Absatz folgt ein Abschnittswechsel, der nicht gelöscht werden sollte. Er ermöglicht den Wechsel von römischen zu arabischen Seitenzahlen. Mehr Infos zum Abschnittswechsel weiter unten. Drücken Sie STRG+SHIFT+PLUS um unsichtbare Steuerzeichen und damit den Abschnittswechsel ein- und auszublenden.

# 1 Einführung

## 1.1 Motivation

## 1.2 Zielsetzung und Aufgabenstellung

## 1.3 Aufbau und Arbeitsweise

# 2 Grundlagen

## 2.1 Blockchain Technologie

### 2.1.1 Crypto-Accounts und -Wallets

### 2.1.2 Smart Contracts

### 2.1.3 NFTs

### 2.1.4 Node Provider

### 2.1.5 IPFS

## 2.2 Ethereum

### 2.2.1 ERC-721

### 2.2.2 Solidity

## 2.3 IT-Sicherheit

### 2.3.1 Web3 Authentifizierung

### 2.3.2 Autorisierung mittels NFTs

## 2.4 Cross-Platform-Entwicklung mit Flutter

# 3. Systemanalyse

Die Verfahren der Web3-Authentifizierung und der Autorisierung durch NFTs, sollen anhand eines Minigames demonstriert werden. In diesem Abschnitt wird ein Minigame-Konzept vorgestellt, an welchem sich genannte Verfahren demonstrieren lassen. Die genauen Anforderungen an das System werden festgelegt, und bieten die Grundlage für den darauf folgenden Entwurf.

## 3.1 Flappy Bird

Als Spielkonzept wird Flappy Bird gewählt. Da die Entwicklung des Spiels nicht im Mittelpunkt dieser Arbeit steht, bietet sich ein triviales und schnell verstandenes Minigame-Konzept wie Flappy Bird an. Der Entwicklungsaufwand ist nicht sonderlich groß, und das Spiel lässt sich durch Funktionalitäten ergänzen, an welchen sich die Autorisierung mit NFTs sowie die dafür erforderliche Web3-Authentifizierung, demonstrieren lässt.

### 3.1.1 Geschichte

Flappy Bird ist ein Minigame, welches im Jahr 2013 von dem vietnamesischen Entwickler Dong Nguyen für iOS und Android Geräte veröffentlicht wurde (source). Nachdem das Spiel für über ein halbes Jahr kaum Aufmerksamkeit erlangte, erlebte es Anfang 2014 eine enorme Hypewelle, welche das Spiel im Apple AppStore zwischenzeitlich auf den 1. Platz der am häufigsten heruntergeladenen Spiele brachte (source). Die ikonische Geschichte von Flappy Bird endete schon kurze Zeit später am 10. Februar 2014, als der Entwickler, sich trotz enormen – auch wirtschaftlichen – Erfolges, dazu entschied das Spiel aufgrund von Kritik aus dem AppStore und Google PlayStore zu entfernen (source). Seinen plötzlichen Erfolg hat das Spiel wohl seiner absoluten Simplizität zu verdanken, welche gleichzeitig einen gewissen Suchtfaktor mit sich brachte. Das Spiel, dessen Entwicklung laut Angaben von Entwickler Dong Nguyen nur zwei bis drei Tage dauerte (source), funktioniert wie folgt:

### 3.1.2 Konzept

Die Spielfigur ist ein kleiner gelber Vogel, welcher sich konstant durch eine zweidimensionale Spielwelt nach rechts bewegt. Auf dem Weg begegnet der Vogel in regelmäßigen Abständen Hindernissen, den es auszuweichen gilt. Das einzige was der Spieler hierfür tun muss, ist es, auf den Bildschirm zu tippen, um den Vogel „springen“ zu lassen. Dabei ist es völlig egal wo auf dem Bildschirm getippt wird. Die Hindernisse werden durch vertikale grüne Röhren dargestellt, welche auf einer zufälligen Höhe eine Öffnung zum Durchfliegen haben (siehe Abbildung 1). Sobald der Vogel eine Röhre berührt, stürzt er ab, und die Runde endet. Ziel des Spiel ist es, in einem Durchlauf durch möglichst viele Röhren zu fliegen, ohne dabei mit einer zu kollidieren. Die Anzahl der durchflogenen Röhren ergibt die Punktzahl, an der sich gemessen werden kann.

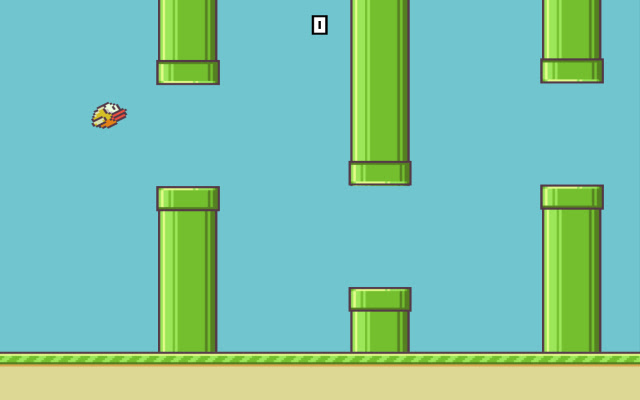


Abbildung : FlappyBird Screenshot

### NFTs in Flappy Bird

Damit an diesem Spielkonzept ein Autorisierungsverfahren mit NFTs veranschaulicht werden kann, wird es durch Funktionalitäten ergänzt, die erst nach erfolgreicher Autorisierung eines bestimmten NFTs freigeschaltet werden. Auf die folgenden Zwei Weisen, soll von NFTs Gebrauch gemacht werden können:

Skins

Bestimmte NFTs sollen dazu genutzt werden können im Spiel alternatives Aussehen für den Vogel („Skins“) freizuschalten. Wird der Besitz eines Skin-NFTs nachgewiesen, hat der Spieler vor dem Start der Runde die Möglichkeit dieses alternative Aussehen zu wählen. Diese „Skins“ stellen eine rein optische Veränderung des Spiels dar, und bieten keine tatsächlichen Vorteile.

Perks

Andere NFTs sollen dazu genutzt werden können bestimmte Vorteile im Spiel („Perks“) zu gewähren. Dazu könnten zum Beispiel Power-Ups wie ein Schild gehören, welches im Spiel eingesammelt werden kann, um die nächste Kollision mit einer Röhre zu vermeiden. Ein weiteres Power-Up könnte nach dem Einsammeln das Spiel für eine gewisse Zeit verlangsamen, was das Spiel vereinfacht.

## Anforderungsanalyse

Die Kernanforderungen an das Gesamtsystem liegen bereits vor: Es soll eine Flutter Anwendung umfassen, welche mindestens für die mobilen Plattformen iOS und Android und für die Benutzung im Web-Browser ausgelegt ist. Außerdem soll das System ein Verfahren zur Authentifizierung mit einem Crypto Wallet implementieren, sowie ein Verfahren welches die Autorisierung mittels ERC-721 Token (NFTs) demonstriert. Um diese Verfahren anschaulich zu demonstrieren, soll das Spielkonzept von Flappy Bird herangezogen und erweitert werden (siehe 3.1 Flappy Bird).

Es sind zwei entscheidende technische Komponenten gefragt: Zum einen die Flutter Anwendung, in welcher das Spielkonzept sowie das Authentifizierungsverfahren und das Autorisierungsverfahren realisiert werden. Und zum anderen der ERC-721 Smart Contract, welcher auf einer Blockchain hinterlegt wird und für die Verwaltung der NFTs zuständig ist. Die Flutter Anwendung und der Smart Contract sind nicht direkt voneinander abhängig und lassen sich jeweils als eigene Systeme betrachten. Aus diesem Grund bietet es sich an, in der weiteren Anforderungsanalyse, zwischen diesen Komponenten zu differenzieren.

Ziel der weiteren Analyse sind konkrete Listen der Anforderungen an den Smart Contract sowie an die Flutter Anwendung, sodass im Anschluss ein Entwurf des Gesamtsystems sowie der beiden Hauptkomponenten angefertigt werden kann. Das zu entwickelnde Gesamtsystem wird von nun an als „**Flutter Bird**“ bezeichnet, da es das Spielkonzept von *Flappy Bird* nachahmen soll, und mit *Flutter* entwickelt wird.

### 3.2.1 Smart Contract

Die grundlegenden Anforderungen an einen ERC-721 Smart Contract sind fest vorgeschrieben und gelten universell für jeden ERC-721 Smart Contract (siehe 2.2.1 ERC-721). Darüber hinaus sollen die von diesem Smart Contract verwalteten NFTs folgende Eigenschaften besitzen:

Tabelle : Zusätzliche Anforderungen an den Flutter Bird Skins Smart Contract

|  |  |
| --- | --- |
| Zufällig generierte Skins | Der Smart Contract soll jedem Token eine zufällig generierte Bilddatei zuordnen können, welche sich als Skin im Flutter Bird Spiel eignet. |
| Hohe Varianz an Skins | Der Smart Contract soll mindestens 500 Token verwalten, um eine hinreichende Varianz an Skins zu ermöglichen. |
| Metadaten | Der Smart Contract soll jedem Token individuelle Metadaten zuweisen können, welche die Eigenschaften und Merkmale des dazugehörigen Skins festhalten |
| Skins für Nutzer abfragen | Der Smart Contract soll eine Funktion bereitstellen, welche es erlaubt alle zu einem Ethereum Account gehörenden Skins, abzufragen |
| Bilddatein im dezentralen Datenspeicher | Die Metadaten eines jeden Tokens sollen eine Referenz zu der entsprechenden Bilddatei des Skins enthalten, welche im IPFS gespeichert ist. |

### 3.2.2 Flutter Anwendung

Um einen Überblick über mögliche Anwendungsszenarien zu schaffen, wird die Flutter Anwendung in einem UML Use Case Diagrammen dargestellt. Abbildung 2 zeigt einen imaginären Nutzer der App und die potentiellen Anwendungsszenarien, durch welche dieser Nutzer mit der App interagieren kann. Zentrales Anwendungsszenario wird es sein, *Flutter Bird* zu spielen.

Diagram

Description automatically generated

Abbildung : Darstellung der Kernfunktionen von Flutter Bird durch ein Use Case Diagramm

Das in Kapitel 3.1 Flappy Bird gezeigte Spielkonzept soll nachgeahmt werden, sodass es dem Ursprungsspiel im Aussehen und in der Funktionsweise ähnelt. Dieses Szenario wird durch die Möglichkeit erweitert, das Spiel mit einem Skin-NFT anstelle der Standard-Spielfigur zu spielen. Damit der Nutzer dazu autorisiert werden kann, mit einem solchen Skin-NFT zu spielen, ist es erforderlich, dass die Anwendung mit dem auf der Blockchain hinterlegtem ERC-721 Smart Contract interagieren kann, um den Besitz eines solchen Skin-NFTs nachzuweisen (siehe 2.3.2 Autorisierung mittels NFTs). Dieser Nachweis wiederum erfordert, dass der Spieler die Möglichkeit hat sich innerhalb der Anwendung mit einem Crypto Wallet zu authentifizieren (siehe 2.3.1 Web3 Authentifizierung). Es lassen sich die folgenden Kernanforderungen an die Flutter Anwendung ableiten:

Tabelle : Anforderungen an die Flutter Anwendung

|  |  |
| --- | --- |
| Flutter Bird spielen | Die Anwendung soll das Spielkonzept von Flappy Bird nachahmen. Auch ohne den Besitz eines Skin-NFTs soll ein Nutzer das Spiel spielen können. |
| Spielen mit Skin-NFT | Die Anwendung soll es autorisierten Nutzern erlauben mit einem alternativen Skin zu spielen. Hierzu kann der Nutzer vor dem Start des Spiel zwischen allen besessenen Skins wählen. |
| Autorisierung durch NFT | Das System soll Nutzer dazu autorisieren können mit einem Skin-NFT zu spielen, indem der Besitz eines solchen nachgewiesen wird. |
| Authentifizierung mit Crypto Wallet | Das System soll ein Verfahren implementieren, welches die Authentifizierung mit einem Crypto-Wallet ermöglicht. |

# 4. Systementwurf

In diesem Abschnitt werden die im vorangegangenen Kapitel ermittelten Anforderungen herangezogen, um die technische Umsetzung des Systems zu planen. Zunächst wird die Architektur des Gesamtsystems entworfen und anhand eines Verteilungsdiagrammes grob veranschaulicht. Das Verteilungsdiagramm zeigt alle erforderlichen Komponenten des Systems, sowie deren Zusammenspiel auf.

Anschließend werden die Hauptkomponenten des Systems, der ERC-721 Smart Contract und die Flutter Anwendung, genauer betrachtet und auf ihre Implementierung vorbereitet. Hierfür werden Kernprozesse im Detail beschrieben und relevante Teile der Datenstruktur erläutert. Darüber hinaus werden die notwendigen Schnittstellen, sowie die Kommunikation zwischen den Komponenten definiert.

## 4.1 Architektur

Ziel der Architektur des Gesamtsystems ist es, die erforderlichen Komponenten, so miteinander in Verbindung zu bringen, dass die in der Systemanalyse ermittelten Anforderungen erfüllt werden können. Abbildung 3 zeigt ein Verteilungsdiagramm des Gesamtsystems, welches alle benötigten Komponenten sowie deren Kommunikationswege darstellt. Die grün markierten Elemente repräsentieren Artefakte, welche es für die Umsetzung von *Flutter Bird* zu erstellen gilt. Alle weißen Kästen repräsentieren Soft- und Hardwarekomponenten von Dritten, welche von *Flutter Bird* beansprucht werden.

**Smart Contract**: Oben links ist die Ethereum Blockchain dargestellt, auf welcher der *Flutter Bird Skins* Smart Contract dezentral hinterlegt wird (siehe Abbildung 3). Da Speicherplatz auf der Ethereum Blockchain ein äußerst begehrtes, und damit auch teures, Gut ist (vgl. 2.1.5 IPFS), sollen die tatsächlichen Daten der *Skins* im dezentralen Datenspeicher, dem IPFS, hinterlegt werden (siehe Abbildung 3, rechts oben). (Erläutern das, und warum ein Testnet verwendet wird für den Prototyp)

Diagram

Description automatically generated

Abbildung : Verteilungsdiagramm des Gesamtsystems

**Flutter App**: Am unteren Rand von Abbildung 3 ist die Flutter Anwendung dargestellt („Flutter Bird App“), welche sich über die drei Plattformen streckt, auf denen sie ausführbar sein soll. Der Übersichtlichkeit halber, ist die Flutter Anwendung hier als ein Softwareartefakt dargestellt. In Realität generieren sich aus dem Quellcode der Flutter Anwendung drei separate Artefakte für die jeweiligen Plattformen (vgl. 2.4 Cross-Platform-Entwicklung mit Flutter).

**Authentifizierung**: Damit der Nutzer sich mit einem Crypto Wallet innerhalb der App authentifizieren kann, ist es erforderlich, dass sich die Flutter Anwendung mit einem Crypto Wallet auf demselben Gerät verbinden kann. Auf den mobilen Plattformen iOS und Android sind Crypto Wallets meist als App verfügbar, und im Browser als Browser Extension. Aufgabe dieser Wallet Anwendungen ist es, zu bestätigen, dass der Nutzer den privaten Schlüssel zu einer entsprechenden öffentlichen Adresse auf der Ethereum Blockchain besitzt, um so den Nutzer als Besitzer des dazugehörigen Blockchain Accounts zu authentifizieren (vgl. 2.1.1 Crypto-Accounts und -Wallets).

**Autorisierung**: Um anschließend zu prüfen, ob der Nutzer einen *Flutter Bird Skin*, also ein ERC-721 Token aus dem *Flutter Bird Skins* Smart Contract*,* in seinem Blockchain Account hält, ist es erforderlich, dass die App Daten aus diesem Smart Contract lesen kann. Die Blockchain ist ein dezentrales System, welches sich nicht, wie ein zentralisierter Server, durch eine eindeutige IP-Adresse erreichen lässt. Aus diesem Grund muss der Dienst eines Node Providers beansprucht werden: Ein Service, welcher eine oder mehrere Ethereum Nodes betreibt, u nd diese über eindeutige IP-Adressen und per HTTP-Protokoll erreichbar macht. Somit kann die Flutter App per HTTP Protokoll (bzw. JSON-RPC over HTTP), Anfragen an die Ethereum Node des Node Providers stellen, über welche der Aufruf von Smart Contract Funktionen ermöglicht wird (vgl. 2.1.4 Node Provider). Auf diesem Wege lässt sich abfragen, ob und wenn ja welche *Flutter Bird Skins* von dem zuvor authentifizierten Ethereum Account besessen werden.

**Skins laden**: Zu jedem der besessenen Token lässt sich anschließend eine Referenz abfragen, welche die zu dem Token gehörigen Metadaten im IPFS eindeutig identifiziert (CID). Ähnlich wie bei einer Blockchain, ist auch das IPFS ein dezentrales System, welches sich nicht über eine eindeutige IP-Adresse erreichen lässt. Ein IPFS-Gateway erfüllt die selbe Aufgabe für das IPFS, wie ein Node Provider für die Blockchain. Es betreibt IPFS Nodes, und macht diese über eindeutige IP-Adressen und HTTP-Protokoll erreichbar. Mit Hilfe eines IPFS-Gateways, lassen sich anhand einer vom Smart Contract erhaltenen Referenz (CID), die zu dem Skin gehörigen Metadaten aus dem IPFS erhalten. Die Metadaten erhalten wiederum eine Referenz (CID) zu der Bilddatei des Skins, welche vom der Flutter App benötigt wird, um das Spielen mit einem Skin zu ermöglichen.

## 4.2 Smart Contract

Die grundlegende Struktur und das grundlegende Verhalten eines Smart Contracts, der dem ERC-721 entspricht, ist durch das IERC-721 Interface vordefiniert (vgl. 2.2.1 ERC-721). Dabei schreibt das IERC-721 Interface lediglich vor, welche Methoden mit welchen Methodensignaturen ein Smart Contract zu implementieren hat, nicht aber, wie diese Implementierung auszusehen haben. Da diese Implementierungen für die meisten Anwendungsfälle von ERC-721 Smart Contracts in etwa gleich auszusehen hat, hat sich die Organisation „*OpenZeppelin*“ zur Aufgabe gemacht Standardimplementierungen zu entwickeln. Diese Implementierungen sind modular strukturiert, sodass verschiedene Bedürfnisse berücksichtigt werden können und stetig garantiert ist, dass der Smart Contract dem IERC-721 Interface entspricht. Sie sind öffentlich zugänglich und gelten als effizient und sicher (source), weshalb auch der *Flutter Bird Skins* Smart Contract von den *OpenZeppelin* Implementierungen Gebrauch machen wird. Abbildung 4 zeigt ein Klassendiagramm des *Flutter Bird Skins* Smart Contracts mit den verwendeten Implementierungen und Interfaces von *OpenZeppelin*.

Diagram

Description automatically generated

Abbildung : Klassendiagramm des Smart Contracts und der OpenZeppelin Implementierungen

(Folgende Erklärung in Kapitel 2.2.2??) Wie in Kapitel 2.2.2 erläutert, werden Klassen in Solidity als Contract bezeichnet. In Abbildung 4 sind also 5 Contracts (blau und grün) und 4 Interfaces (gelblich) dargestellt. Es ist wichtig zu verstehen, dass das Konzept des Contracts in Solidity nicht mit dem Konzept eines Smart Contracts auf der Blockchain gleichzusetzen ist. Das gesamte in Abbildung 4 dargestellte Konstrukt mit allen Contracts und Interfaces ergibt auf der Blockchain einen Smart Contract. Um Unklarheiten zu vermeiden werden die Contracts im Kontext von Solidity im Folgenden als Klassen bezeichnet.

Grün dargestellt ist die FlutterBirdSkins Klasse, welche die einzige ist, die erstellt werden muss. Alle blauen Elemente repräsentieren die von *OpenZeppelin* benutzten implementierten Klassen. Und gelblich markiert sind die verwendeten Interfaces. „IREC721“ und „IERC165“ sind Standards die einem Ethereum Improvement Proposal (EIP) entspringen. „IERC721Metadata“ und „IERCEnumerable“ sind von *OpenZeppelin* entworfene Standards, welche die erforderlichen Kernfunktionen eines ERC-721 Smart Contracts um weitere optionale Funktionen erweitern. Das „IERC721Metadata“ Interface erweitert das „IERC721“ Interface durch drei weitere Methoden: „name()“ und „symbol()“, die der Identifizierung eines Smart Contracts dienen, und „tokenURI(tokend)“, welche jedem Token des Smart Contracts einen eindeutige Kennung (Uniform Resource Identifier) zuordnet. Der Name des soll „FlutterBirdSkins“ heißen und das Symbol entsprechend „FBS“. Wie in den Anforderungen an den Smart Contract festgehalten, soll der Smart Contract einem jeden Token eindeutige Metadaten zuweisen können, welche im IPFS hinterlegt sind. Die „tokenURI“ Methode wird benutzt, um für jeden Token die Referenz (CID) zu den sich im IPFS befindenden Metadaten zurückzugeben.

## 4.3 Flutter App

Die Architektur der Flutter Anwendung wird einem Model-View-Controller (MVC) Muster folgen. Im folgenden Entwurf wird ein deutlicher Fokus auf die zu demonstrierenden Verfahren der Authentifizierung und Autorisierung gelegt. Das Spielkonzept und die Spiellogik steht im Hintergrund. Abbildung 5 zeigt ein vereinfachtes Klassendiagramm, welches der Entwurfsplanung dienen soll. (Folgende Erklärung von MVC in Kapitel über Flutter?) Die blau markierten Klassen repräsentieren die Benutzeroberfläche (*Views*) der App. Die Aufgabe der *Views* besteht zum einen darin, die im *Model* festgehaltenen Informationen grafisch darzustellen, als auch darin Nutzerinteraktionen festzustellen und zur Verarbeitung an den *Controller* weiterzuleiten. Der *Controller* wird von den gelb markierten Klassen repräsentiert (siehe Abbildung 5). Dieser ist zum einen dafür verantwortlich, dass die *Views* über Veränderungen im *Model* benachrichtigt werden, als auch für die Ausführung von Geschäftslogik, die durch Nutzerinteraktionen in *Views* veranlasst wird und die die Daten im *Model* verändern kann. Letztlich, stellen die grün markierten Klassen das *Model* dar, also die für die Umsetzung erforderlichen Daten.

**Views**: Für den Nutzer soll die App so einfach wie möglich zu bedienen sein. Aus diesem Grund wird es, genau wie bei dem originalen Flappy Bird Spiel, nur zwei Ansichten (Views) geben: Das Hauptmenü („MainMenuView“), und das Spiel selbst („GameView“). Wenn die App startet, öffnet sich das Hauptmenü. Hier hat der Spieler die Möglichkeit das Spiel zu starten. Wenn der Spieler sich zuvor authentifiziert hat, hat er im Hauptmenü die Möglichkeit zwischen allen von ihm besessenen Skins zu wählen. Ist er noch nicht authentifiziert, hat er die Möglichkeit die Authentifizierung zu veranlassen. Wird das Spiel gestartet, öffnet sich die Spielansicht („GameView“). Hier ist die Spielwelt zusehen und die Spiellogik implementiert. Der Spieler kann mit der Standardspielfigur oder einem zuvor ausgewählten Skin spielen. Kollidiert der Spieler mit seinem Vogel, endet die Runde. Die Spielansicht schließt sich, und das Hauptmenu ist wieder zu sehen.

Diagram

Description automatically generated

Abbildung : Vereinfachtes Klassendiagramm zur Entwurfslpanung der Flutter App

**Controller**: Die Logik der App, welche auch Authentifizierungs- und Autorisierungsverfahren umfasst, wird in drei Klassen aufgeteilt: Der *FlutterBirdController*, der die Schnittstelle zu den *Views* darstellt, und der *AuthenticationService* sowie der *AuthorizationService*, die, wie sich den Namen entnehmen lässt, für Authentifizierung und Autorisierung verantwortlich sein werden. Der *FlutterBirdController* dient als Fassade. Er stellt die Funktionalität der Service Klassen durch Methoden bereit, sodass sie durch die Views aufrufbar ist. Außerdem Informiert der *FlutterBirdController* die Views über Veränderungen im *Model*. Hierfür wird das, in Flutter stark etablierte, Provider Pattern (Erläuterung nötig?) verwendet. Die beiden „Service“ Klassen lassen sich als Teil des *FlutterBirdControllers* verstehen. Der *AuthenticationService* ist für den Authentifizierungsprozess verantwortlich, der von einem Nutzer aus dem Hauptmenü angestoßen werden kann. Hierfür muss Verbindung zu einem Crypto Wallet auf dem Gerät des Nutzers aufgebaut werden, sodass dieses Crypto Wallet bestätigen kann, dass der Nutzer Besitzer eines privaten Schlüsseln zu einem bestimmten Ethereum Account ist. Der *AuthorizationService* führt den Autorisierungsprozess durch, indem geprüft wird, ob der zuvor authentifizierte Ethereum Account im Besitz eines oder mehrerer Skin-NFTs ist. Wird der Besitz festgestellt, so werden die zu den besessenen Skins gehörenden Bilddatein aus dem IPFS geladen. Es ist also der *AuthorizationService* der über einen Node Provider mit der Blockchain, als auch über ein IPFS Gateway mit dem IPFS, in Verbindung treten wird, um Daten abzufragen (vgl. Abbildung 3).

**Model**: Das Model (Die Fachklassen) der Anwendung setzen sich aus zwei Typen zusammen: Dem *Account*, der einen Ethereum Account darstellt, sowie die dazugehörigen *Skins*. Sobald ein Nutzer erfolgreich vom *AuthenticationService* authentifiziert wird, wird ein Objekt vom Typ *Account* erstellt. Diese Änderung wird durch den *FlutterBirdController* den *Views* mitgeteilt, sodass der authentifizierte *Account* auf der Benutzeroberfläche dargestellt wird. Der *AuthorizationService* erstellt während des Autorisierungsprozesses zu jedem vom *Account* besessenen Skin-NFT ein Objekt des Typs *Skin* und ordnet diese dem Account zu. Auch diese Änderung wird durch den *FlutterBirdController* den Views mitgeteilt, sodass alle dem authentifizierten Account zur Verfügung stehenden Skins im Hauptmenu zur Auswahl angezeigt werden können.

# 5. Prototypische Realisierung

## 5.1 Smart Contracts

### 5.1.1 Ganache

### 5.1.2 Skins

### 5.1.3 Perks

## 5.2 Flutter App

### 5.2.1 Flutter Bird

### 5.2.2 Web3-Authentifizierung

### 5.2.3 Autorisierung mit NFTs

# 6. Vorführung der Anwendung

# 7. Bewertung des entwickelten Prototyps

# 8. Fazit

# Literaturverzeichnis

Entscheiden Sie sich für eine Variante des Verzeichnisses. Sie sind beide ähnlich formatiert, das eine ist eine sortierbare Tabelle, die manuell befüllt wird, das andere ist ein Standard-Verzeichnis. Formatiert wird das Verzeichnis mit der Absatzformatvorlage „Literaturverzeichnis“.

## Literaturverzeichnis als sortierbare Tabelle

|  |
| --- |
| Eco, Umberto: Wie man eine wissenschaftliche Abschlussarbeit schreibt. UTB, 2007. |
| Fahnenstich, Klaus/Haselier, G. Rainer: Microsoft Home and Student 2007: Microsoft Press Deutschland, 2007. |
| Karmasin, Matthias/Ribing, Rainer: Die Gestaltung wissenschaftlicher Arbeiten. UTB, 2008. |
| Nicol, Natascha/Albrecht, Ralf: Wissenschaftliche Arbeiten schreiben mit Word 2007. Addison-Wesley, 2007. |
| Ravens, Tobias: Wissenschaftlich mit Word arbeiten. Pearson Studium, 2004. |

## Standard-Literaturverzeichnis

Eco, Umberto: Wie man eine wissenschaftliche Abschlussarbeit schreibt. UTB, 2007.

Fahnenstich, Klaus/Haselier, G. Rainer: Microsoft Home and Student 2007: Microsoft Press Deutschland, 2007.Karmasin, Matthias, und Rainer Ribing. Die Gestaltung wissenschaftlicher Arbeiten. UTB, 2008.

Karmasin, Matthias/Ribing, Rainer: Die Gestaltung wissenschaftlicher Arbeiten. UTB, 2008.

Nicol, Natascha/Albrecht, Ralf: Wissenschaftliche Arbeiten schreiben mit Word 2007. Addison-Wesley, 2007.

Ravens, Tobias: Wissenschaftlich mit Word arbeiten. Pearson Studium, 2004.

# Index

Abbildungen 18

Abkürzungsverzeichnis V

Abstände 14

Arial 9

Automatische Nummerierung 12

Beschriftung 18

Bindestriche 16

Dokumentstruktur 9

Einzüge 15

Formatvorlagenleiste 8

Fußnoten 17

Gedankenstriche 16

Geschützte Leerzeichen 16

Inhaltsverzeichnis 22

Kopfzeilen 15

Nummerierungen 16

Satzspiegel 8

Shortcuts 10, 27

Steuerzeichen 9

Tabellen, unsichtbar 12

Überschriften 14

Überschriften, mehrere 14

Zitate 17

# Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich diese Bachelorarbeit selbstständig ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen sind als solche einzeln kenntlich gemacht.

Diese Arbeit ist bislang keiner anderen Prüfungsbehörde vorgelegt worden und auch nicht veröffentlicht worden.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, Datum, Unterschrift