STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Rise of Pilgrims

David Domkář
Pardubický kraj

Pardubice 2021

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Rise of Pilgrims

Rise of Pilgrims

Autoři: David Domkář
Škola: DELTA – Střední škola informatiky a ekonomie, s.r.o.
Kraj: Pardubický kraj
Konzultant: Bc. Vlaďka Janů

Pardubice 2021

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Pardubicích dne 31. 3. 2021 ……

David Domkář

Poděkování

Chtěl bych poděkovat Bc. Vlaďce Janů za odborné vedení projektu. Dále děkuji Michalovi Špitálskému za grafické zpracování projektu a Karlovi Koudelkovi za vytvoření soundtracku do aktivní části hry. V neposlední řadě také děkuji svým spolužákům, kteří pomáhali s testováním projektu.

Anotace

Cílem projektu je vytvořit arkádovou mobilní hru typu „endless runner“, ve které se více hráčů snaží se svou herní postavou dosáhnout co nejvyššího umístění v globálním žebříčku. Kromě mobilní aplikace se samotnou hrou je součástí projektu také webová administrace sloužící k editování hodnot jednotlivých hráčů, blokaci hráčů a zobrazení statistik administrátory hry.

Klíčová slova

Mobilní hra; webová administrace; Flutter; NuxtJS

Annotation

The goal of this project is to create an „endless runner“ arcade mobile game in which the players are trying to get their in-game character to the highest place in the global leaderboard. In addition to the mobile application with the game, there is a web administration where the game admins can edit each player’s values, block players or display statistics about the game.

Keywords

Mobile game; web administration; Flutter; NuxtJS

Obsah

1 Úvod

Mobilní hry mají již několik let většinový podíl na trhu s videohrami a jejich podíl neustále roste. Většina je přitom v principu velice jednoduchá a s jasným cílem. Slouží z pravidla ke zkrácení dlouhé chvíle. Tyto hry kvůli své jednoduchosti zpravidla velice rychle omrzí, protože hráč často hraje pouze sám se sebou a hra ho nenutí se ve hraní zdokonalovat.

Cílem projektu Rise of Pilgrims je, aby hra byla stále velice jednoduchá, ale zároveň si udržovala hráče díky globálnímu žebříčku. Žebříček ukazuje nejlepší hráče, mezi které se hráč neustále snaží dostat. Tento systém lze do budoucna rozšířit o sezóny, kdy se globální žebříček po nějaké době automaticky resetuje a odmění nejlepší hráče, aby ostatní hráči měli novou šanci se umístit a znovu měli chuť hru hrát.

2 Technologie

Následující část se věnuje použitým technologiím v projektu. Pro mobilní aplikaci je použit framework Flutter a jazyk Dart. Backend aplikace tvoří platforma Firebase a webová administrace je napsaná ve frameworku Nuxt.js.

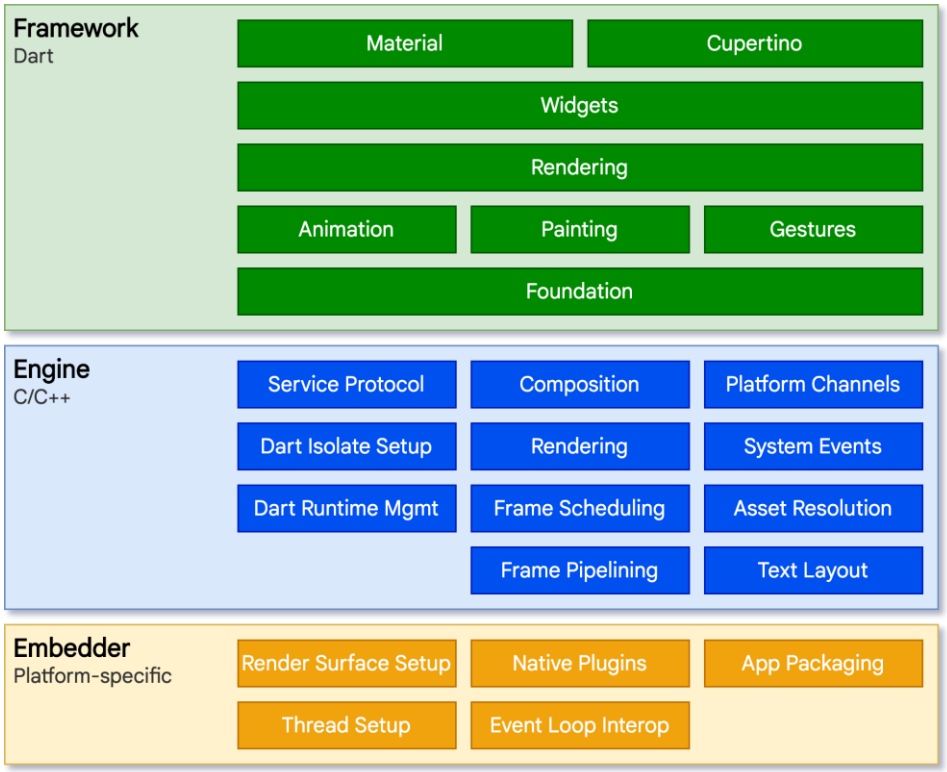
2.1 Flutter

Flutter je framework pro vytváření multiplatformních aplikací vyvinutý společností Google. Jedná se o poměrně novou technologii, která se ale poměrně rychle adoptuje v celém průmyslu. Jeho hlavní výhoda je již zmíněná multiplatformnost, umožňuje aplikaci sestavit pro více platforem bez nutnosti měnit kód. Flutter je také open source, takže se jeho kód může volně šířit a modifikovat.

Obrázek 1 - Flutter Logo (zdroj: https://flutter.dev) (Google LLC, n.d.)

2.1.1 Architektura Flutteru

Flutter se skládá z několika částí. Hlavní částí je samotný Flutter Engine, je napsaný převážně v jazyce v C++ a stará se o hlavní funkce aplikace, jako je vykreslování pomocí grafické knihovny Skia nebo komunikace s cílovou platformou aplikace přes Embedder – specifický kód pro danou platformu, který Enginu umožňuje na dané platformě běžet. Nakonec je tu Framework, set knihoven napsaných v jazyce Dart, které se využívají k psaní samotné Flutter aplikace. Dart kód je ve výsledku zkompilován do nativního kódu, takže výsledná Flutter aplikace je velice rychlá, podobně jako aplikace napsaná pro danou platformu přímo. Toto je velká výhoda Flutteru oproti konkurenčním multiplaftormním frameworkům, jako je například React Native.



Obrázek 2 - Architektura Flutteru (zdroj: https://flutter.dev/docs/resources/architectural-overview)

Díky tomu, jak je Flutter navržen, v něm není problém vytvářet i graficky a výpočetně náročnější aplikace, jako jsou například hry.

2.1.2 Widgety

Flutter aplikace se skládá z widgetů, jsou to jednotlivé prvky v aplikaci. Každé tlačítko, text, i samotná aplikace je widget. Widgety postupně tvoří robustní hierarchii aplikace. Programátor si může definovat widgety vlastní nebo si upravit již existující, které má Flutter předpřipravené k rychlému vývoji aplikace. Předpřipravené widgety jsou dostupné ve dvou designech. Material Design (Android) nebo Cupertino (iOS).

2.1.3 Dart

Dart je programovací jazyk od společnosti Google. Je open source a Flutter ho používá jako svůj jazyk pro psaní aplikací a knihoven. Jazyk je optimalizován pro programování rychlých aplikací na jakékoli platformě.

2.1.3.1 Hot-reload

Dart disponuje funkcí hot-reload, znamená to, že dokáže dynamicky nahrazovat programátorem modifikované části programu za běhu. Tato funkčnost se přenáší na Flutter. Během vývoje tak programátor nemusí aplikaci po každé změně restartovat, ale vidí změny okamžitě.

2.1.4 Pub.dev

Pub.dev je webové rozhraní registru knihoven pro Flutter a Dart, lze zde vyhledat spoustu knihoven pro přidání do aplikace. Pub.dev nazývá knihovny packages neboli balíčky. Do aplikace lze knihovny přidat pomocí souboru pubspec.yaml v kořenové složce Flutter projektu.

```
easy_localization: 2.3.4-nullsafety.1

flame: 1.0.0-rc5
flame_forge2d: 0.5.0-rc1
flame_gamepad: 0.0.2

flutter_modular: 3.0.0-nullsafety.25

sortedmap: 0.4.2+1
ordered_set: 2.0.0
json_annotation: 3.1.1

flutter_mobx: 1.1.0+1
mobx: 1.2.1+3

validators: 2.0.1

firebase_core: 0.5.3
firebase_analytics: 6.3.0
firebase_auth: 0.18.4+1
cloud_firestore: 0.14.4
cloud_functions: 0.7.2

google_fonts: 1.1.1

auto_size_text: 2.1.0
keyboard_avoider: 0.1.2

flutter_facebook_auth: 1.0.2+2
google_sign_in: 4.5.5

package_info_plus: 0.5.0

lint: 1.5.1
```

Obrázek 3 - Ukázka pubspec.yaml

2.1.5 Flame

Flame je knihovna napsaná v jazyce Dart, funguje jako nadstavba nad Flutter frameworkem a plní funkci herního enginu. Obsahuje užitečné abstrakce nad Flutter Canvas API, díky které lze ve Flutteru provádět vlastní grafické operace. Také se stará o načítání herních assetů.

Flame poskytuje widget GameWidget, kterému programátor jako parametr v konstruktoru poskytne svou třídu dědící ze třídy Game a implementuje metody jako je například update nebo render, kde se odehrává samotná logika hry.

Organizaci herních objektů Flame řeší systémem hierarchie komponent, tento systém je však použit jen okrajově, protože projektu nevyhovoval. Více je popsáno v kapitole 3 Architektura.

Obrázek 4 - Logo Flame (zdroj: https://pub.dev/packages/flame/versions/1.0.0-rc8)

Knihovna je dostupná v registru pub.dev a je rovněž open source. Více o Flame se lze dozvědět na webových stránkách enginu: https://flame-engine.org

2.1.5.1 Forge 2D

Součástí Flame je také knihovna Forge 2D, jde o fyzický engine postavený nad Dart implementací populárního Box2D. V aplikaci je použit na detekci kolizí objektů.

2.2 Firebase

Firebase je **backend as a service** platforma od společnosti Google. Jedná se o kompletní serverless řešení backendu, takže programátor nemusí řešit škálování a údržbu vlastních serverů nebo databází. Firebase nabízí pro různá řešení různé produkty, se kterými aplikace komunikuje přes jejich SDK. V projektu je jich použito hned několik a jsou sepsané níže.

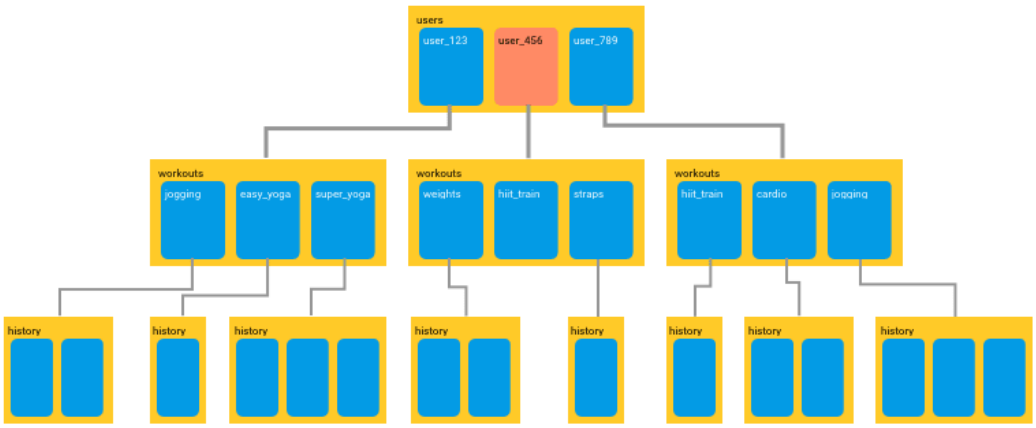
Obrázek 5 - Logo Firebase (zdroj: https://firebase.google.com)

2.2.1 Firebase Authentication

Firebase Authentication je řešení pro přihlašování a registraci uživatelů. Umožňuje konfiguraci přihlašování přes různé poskytovatele, jako je například Facebook nebo Google. Také lze nastavit přihlašování anonymně nebo přes email. Dále produkt řeší ověřování emailu, obnovu hesla a změnu emailové adresy. Vše lze ovládat přímo z aplikace pomocí Firebase SDK.

2.2.2 Cloud Firestore

Cloud Firestore je flexibilní škálovatelná cloudová NoSQL databáze, která podporuje synchronizaci dat s připojenými klienty v reálném čase. Data se ve Firestore strukturují do kolekcí a dokumentů. Kolekce může obsahovat více dokumentů a do každého dokumentu lze uložit jakákoliv data ve formátu JSON.



Obrázek 6 - Struktura Firestore databáze (zdroj: https://proandroiddev.com/working-with-firestore-building-a-simple-database-model-79a5ce2692cb)

Dotazování na databázi probíhá pomocí Firebase SDK přímo z klientské aplikace. Není tak nutné programovat vlastní API, která slouží jako prostředník mezi databází a aplikací.

Bezpečnost je ve Firestore řešena přes Firebase Security Rules. Jazyk určený přímo pro Firestore databázi, ve kterém si lze napsat vlastní pravidla pro přístup do jednotlivých kolekcí a dokumentů na základě informací od klienta nebo dat v databázi.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /characters/{userId} {
      allow get: if request.auth != null && request.auth.uid == userId;
      allow list: if request.auth != null;
    }
  }
}
```

Obrázek 7 - Ukázka security rules

Jedna z nevýhod Firestore je limitovaná API pro dotazování se na data. Často se musí data na různých místech v databázi denormalizovat, aby se předešlo nutnosti dotazovat se na data vícekrát, což násobí počet čtení, který je jedním z předmětů ceníku databáze. Nicméně díky těmto limitům je databáze schopna operovat v podstatě zdarma, protože dotazy jsou velice jednoduché a nevyžadují tak velké výpočetní prostředky. Navíc má oproti jiným řešením pro

projekt značné výhody, jako je již zmíněná synchronizace dat v reálném čase nebo absence backendového kódu s API, který by se musel udržovat.

2.2.3 Hosting

Hosting je produkt, který umožňuje hostovat statické weby přímo ve Firebase. Lze mít libovolný počet webů a napojit na ně svou vlastní doménu. Nahrání souborů webu lze docílit pomocí nástroje Firebase CLI.

Hosting automaticky stránku ukládá do mezipaměti různých CDN (Content Delivery Network) serverů všude po světě. Stránky se tak vždy načtou velice rychle nezávisle na lokaci cílového uživatele.

Hosting lze zároveň použít i jako prostředník mezi uživatelem jinými službami Google Cloudu, které mají veřejný endpoint. Například Cloud Run nebo Cloud Functions.

V projektu je **Hosting** použit pro hostování administrace hry - https://admin.riseofpilgrims.com a také experimentální webové verze aplikace běžící přes Flutter Web - https://riseofpilgrims.com

2.2.4 Cloud Functions

Cloud Functions je další produkt pod **Firebase**. Umožňují přímo v cloudu spouštět váš vlastní kód jako odpověď na různé události. Například registrace uživatele, změna dat v databázi nebo vrátit data po tom co se klient dotáže na http endpoint dané funkce.

Cloud Funkce lze psát v jazyce TypeScript nebo JavaScript a jejich nahrání do cloudu lze učinit pomocí nástroje Firebase CLI.

V projektu se **Cloud Functions** využívají hlavně pro pokročilou logiku na serveru, například při uložení hry nebo při vytváření herní postavy, kdy se musí validovat data před uložením do databáze a jedná se o větší transakci s vlastní logikou, na kterou dříve popisované Firestore Rules nestačí kvůli svým limitům.

2.2.5 App Distribution

App Distribution je produkt, díky kterému lze přímo ve webovém rozhraní Firebase spravovat verze mobilních aplikací jak pro Android, tak iOS a distribuovat je testerům.

Testery lze do App Distribution přidat přes jejich email. Po nahrání sestaveného balíčku aplikace vývojářem pak Firebase informuje testery emailem o nové verzi, kterou si mohou nainstalovat do svého zařízení přes aplikaci App Tester od Firebase.

App Distribution projekt využívá pro testování aplikace během vývoje.

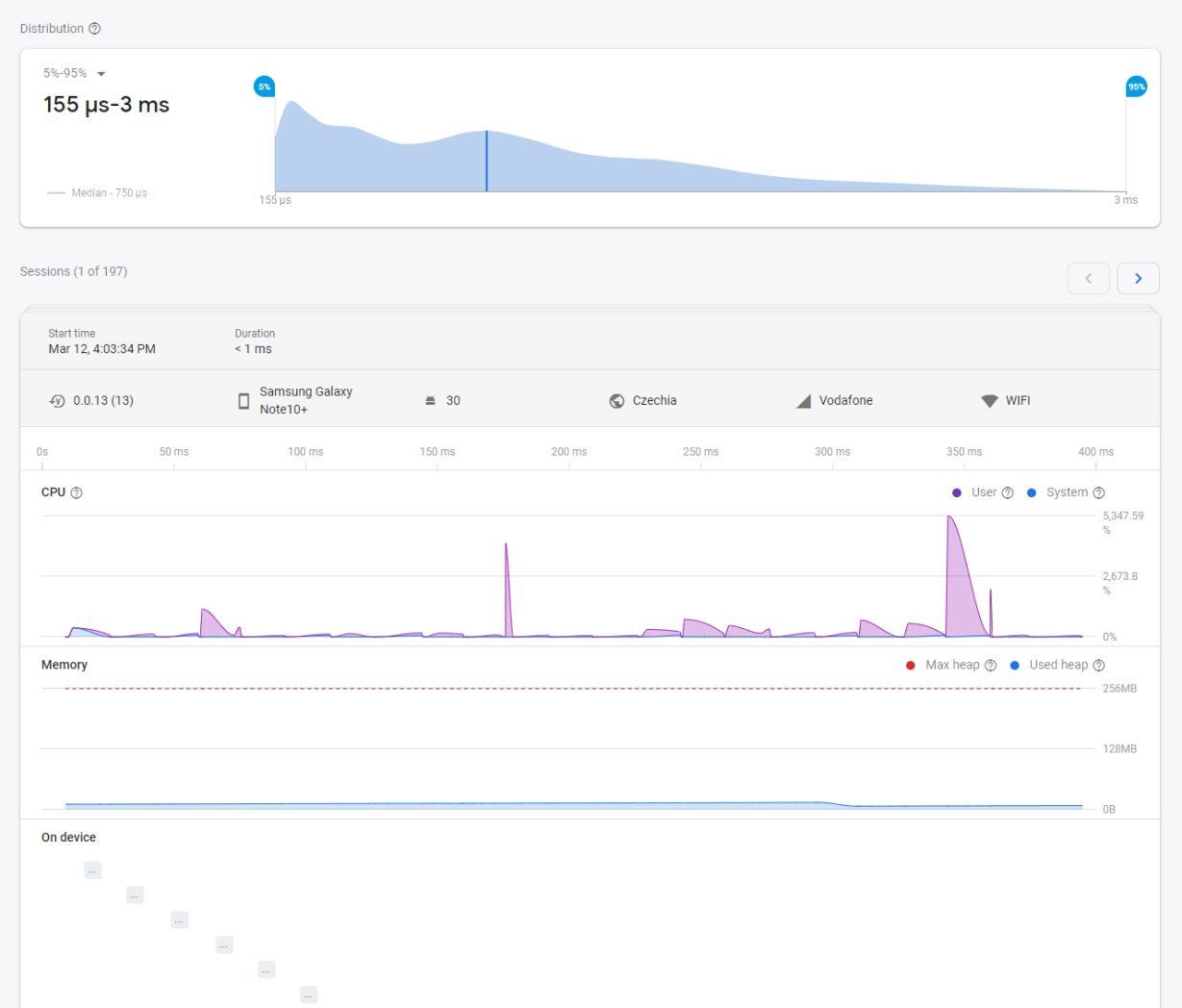
2.2.6 Crashlytics

Crashlytics je nástroj, který umožňuje po instalaci SDK do aplikace odesílat data o pádech aplikace přímo do Firebase konzole. Vývojář má tak přehled o pádech aplikace napříč různými zařízeními a verzemi operačního systému. **Crashlytics** poskytne v reportu o pádu údaje o tom, kde v kódu aplikace pád nastal. Lze tak velice rychle odhalit a opravit zásadní chyby v aplikaci bez nutnosti nahlášení chyby uživatelem, kde se ne vždy dá přesně dohledat, co pád způsobilo.

2.2.7 Perfomance Monitoring

Perfomance Monitoring je další Firebase produkt. Slouží k monitorování doby trvání určitých částí aplikace nebo síťových dotazů. Ve Firebase konzoli je pak možné vidět statistiky jako například doba spouštění aplikace. Data lze zobrazit podle typů zařízení, operátora nebo verze operačního systému.

V projektu je performance monitoring použit v metodách update a render, které jsou kritickými body aplikace a potřebují být co nejrychlejší. Takto bylo možné analyzovat, jak si tyto kritické body vedou napříč různými zařízeními a na základě toho provést optimalizace.



Obrázek 8 - Ukázka performance monitoring

2.3 NuxtJS

NuxtJS je framework pro vytváření webových aplikací. Je postavený na frameworku VueJS a je open source. NuxtJS má modulární architekturu. Existuje velký počet již vytvořených modulů pro různé funkce aplikace. Například pro integraci s Firebase. Není proto potřeba veškeré funkce implementovat od začátku. Další výhodou Nuxtu je možnost statické generace stránek (SSG) nebo serverové renderování stránek (SSR). Obě možnosti zajistí, že je webová aplikace lépe indexovatelná prohlížečem, protože se nečeká na vykreslení stránky JavaScriptem po načtení prohlížečem. Ale prohlížeči se servuje již hotové HTML.

NuxtJS je v projektu použitý pro webovou administraci.

2.3.1 Komponenty

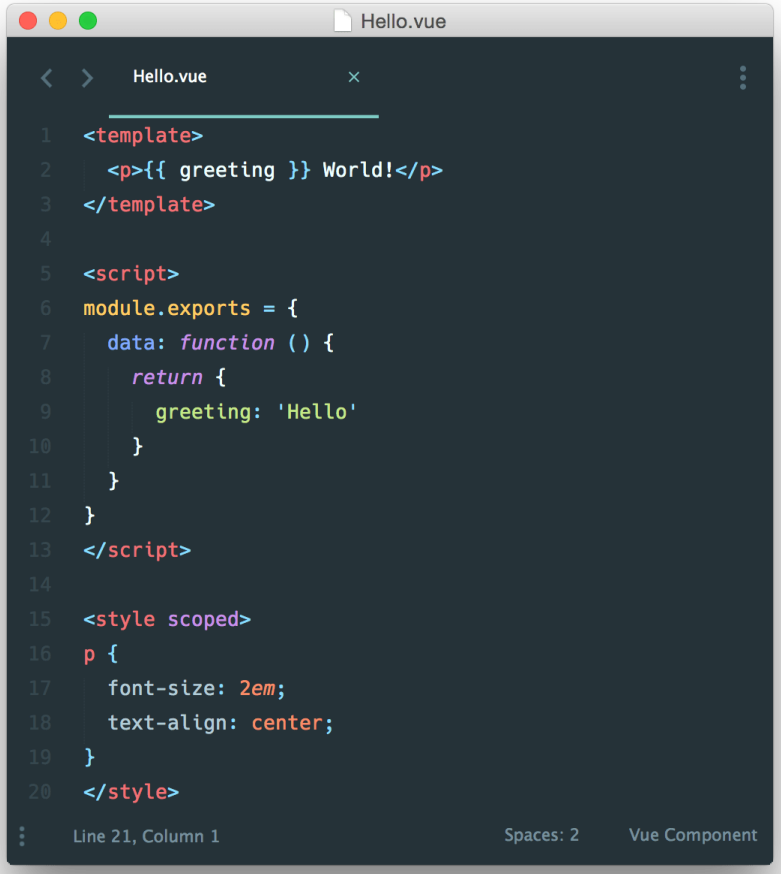
Podobně jako jiné frameworky i NuxtJS používá komponenty ke členění kódu. Jelikož Nuxt vychází z VueJS. Používá se pro zápis komponent Vue SFC (Single File Component). Jedná se o formát souboru s koncovkou .vue, ve kterém je veškerý kód k danému komponentu, ať už se

jedná o styly, html nebo skript. Sekce jednotlivých částí v souboru jsou definovány top-level párovými tagy, mezi které se vždy píše daná část. V základu jsou k dispozici tři hlavní části.

- **<template>** - HTML kód rozšířený o specifické direktivy pro propsání hodnot proměnných definovaných v kódu. Template dovoluje specifikování atributu lang, pro možnost psaní komponentu v jiném jazyce, pokud je v dependencies projektu nainstalovaný webpack loader pro tento jazyk. Příkladem je například jazyk Pug, který je v projetku pro zápis komponent využit.

- **<style>** – část, ve které je zapsán styl. V základu je v jazyce css, ale opět lze díky atributu lang specifikovat například sass nebo postcss. Style sekce má navíc atribut scoped, kterým lze říci, aby se styly aplikovaly pouze na tento komponent. Bez atributu scoped se styly aplikují globální v celé aplikaci.

- **<script>** – část s kódem komponentu. V základu v jazyce JavaScript, ale lze psát i například v jazyce TypeScript opět specifikací atributu lang.



Obrázek 9 - Ukázka Vue SFC (zdroj: https://vuejs.org/v2/guide/single-file-components.html)

Single file components jsou programovatelné o další části. To se může hodit u větších projektů, kde lze například dokumentaci komponentu zapisovat přímo do souboru s komponentem do vlastní sekce.

2.3.2 Struktura souborů

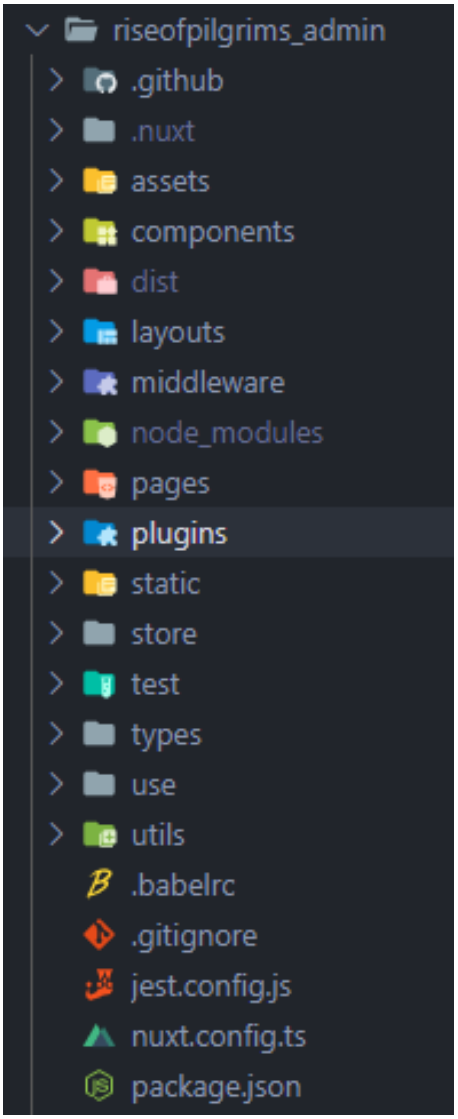
NuxtJS má specifickou strukturu složek a souborů. Níže jsou popsané některé složky a soubory v projektu a jejich funkce.

- **/pages** – hlavní složka v NuxtJS projektu. Jsou zde umístěny Vue komponenty, podle jejichž názvu se generují URL adresy jednotlivých stránek. Komponenty v této složce jsou rozšířeny o vlastnosti specifické pro framework NuxtJS, jako je například middleware pro modifikaci přístupu nebo layout pro specifikaci rozložení, ve kterém se má stránka vykreslit.

- **/layouts** – zde jsou umístěny Vue komponenty, které slouží pro určení rozložení stránek. Stránky ze složky /pages si pak mohou pomocí vlastnosti layout určit, jaké rozložení ze složky layout chtějí použít.

- **/components** – složka pro klasické VueJS komponenty, nejsou rozšířeny o žádné vlastnosti NuxtJS frameworku. Komponenty v této složce se automaticky globálně zpřístupní do celé aplikace, lze je proto pak kdekoliv použít bez nutnosti importu.

- **/assets** – složka pro veškeré assety projektu (obrázky, fonty, globální styly, atd…), které budou manipulovány v průběhu sestavování aplikace. Např. styly ve formátu sass, ze kterých se stane css nebo obrázky, které se sestavením automaticky zkompresují.



Obrázek 10 - Ukázka struktury NuxtJS projektu

- **/middewares** – složka pro middlewary. Middleware umožňuje spustit kód při načítání stránky nebo přesměrování. Obvykle se používá při kontrole oprávnění uživatele, zda má na danou stránku přístup. Middleware lze použít buď globální nebo specifický pro stránku nebo layout.

- **/plugins** – kód, který rozšiřuje Nuxt o další funkce. Obvykle se používá pro instalaci pluginů původně určených pouze pro VueJS.

- **nuxt.config.ts** – soubor pro konfiguraci celého Nuxt projektu. Existuje zde spousta konfiguračních možností od jména aplikace až po konfiguraci samotného sestavování projektu přes webpack.

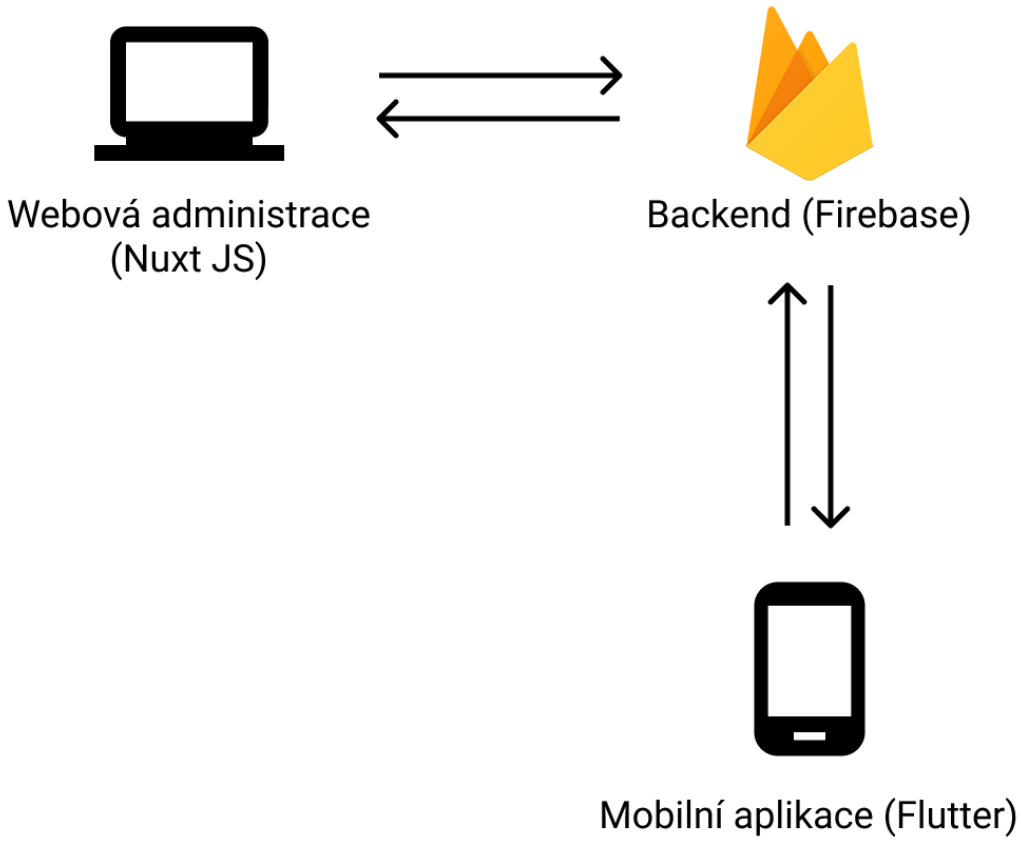
2.3.3 Vuetify

Vuetify je knihovna komponent pro VueJS a NuxtJS implementující Material Design specifikaci. Knihovna nabízí spoustu předpřipravených komponent pro rapidní vývoj aplikací.

Šetří tak čas při vývoji. Vzhled komponent si lze přizpůsobit jejich atributy nebo přes sass api. Vuetify je plně open source a do NuxtJS lze integrovat pomocí komunitního modulu.

3 ARCHITEKTURA

Architekturu aplikace tvoří 3 základní prvky – Mobilní aplikace, webová administrace a backend. Níže je ukázaný obrázek znázorňující propojení jednotlivých prvků.



Obrázek 11 - Architektura projektu (Firebase logo: https://firebase.google.com, ikony: https://material.io/icons)

Z obrázku lze vidět, že architektura je opravdu jednoduchá, v případě aplikace i administrace se jedná o komunikaci klient-server, kde roli serveru plní Firebase. Backend se dále dělí na 3 jednotlivé části – server pro přihlašování, databázi a backendovou logiku pomocí Cloud Functions. Všechny tyto části spolu vzájemně komunikují a aplikace i administrace mohou s jakoukoliv z nich komunikovat napřímo přes Firebase SDK.

3.1 Přihlašovací server

Roli přihlašovacího serveru plní Firebase Authentication. V aplikaci je možné přihlásit se přes sociální sítě (Google a Facebook) nebo využitím emailu. Po prvotní registraci Firebase Authentication uloží uživatele s jeho daty do své interní databáze, ke které se lze dotazovat přes REST API nebo jednodušeji přes Firebase SDK.

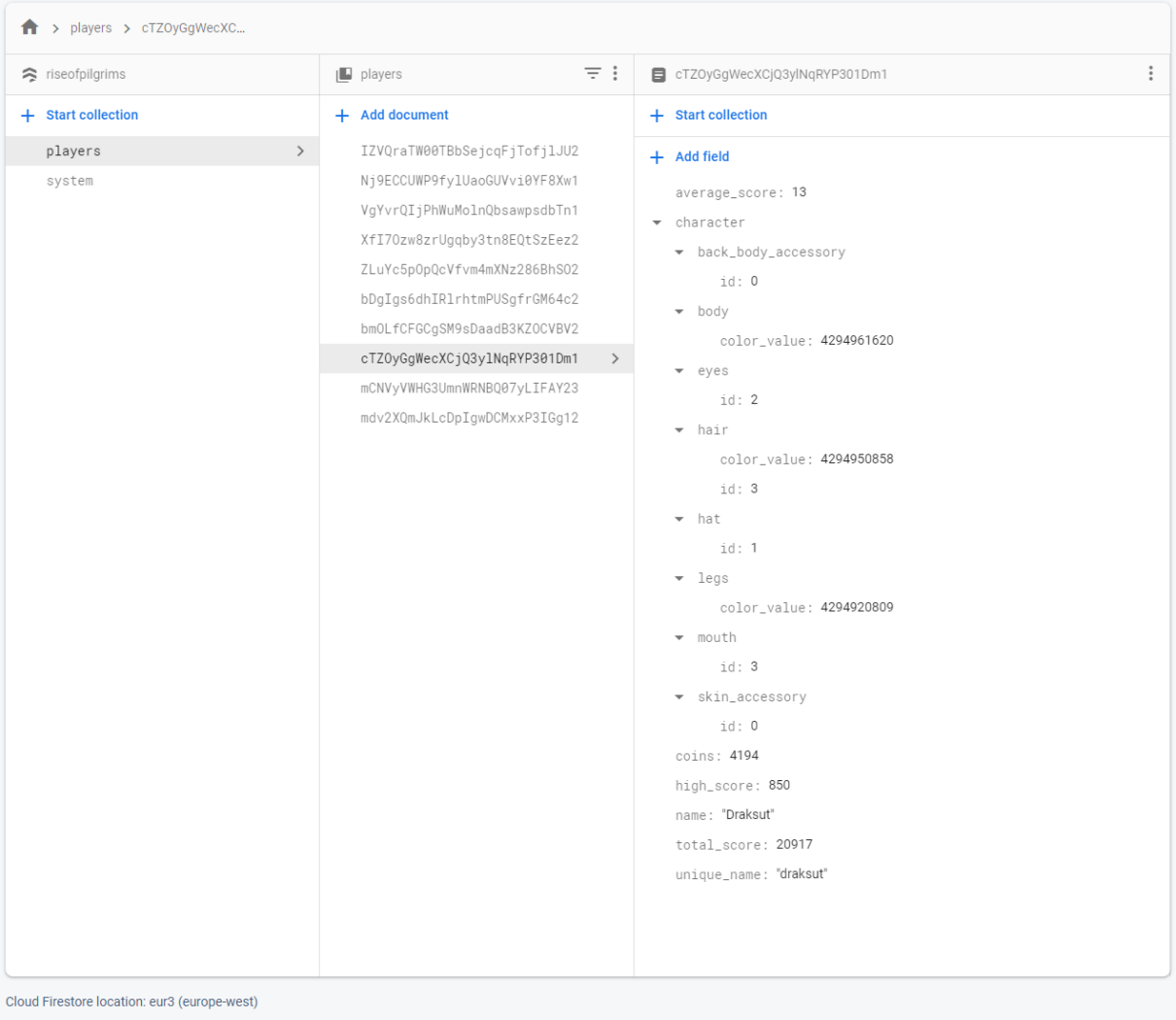
Přihlašování přes Firebase Authentication je založené na JWT tokenech. Tokeny jsou časově omezené a existují 2 hlavní typy. **Refresh token**, který má expirační dobu jednoho roku a využívá se k obnově druhého typu tokenu zvaného **access token**, který má expirační dobu pouze jednu hodinu. Access token je potřeba připojovat ke každému dotazu na ostatní části architektury, ať už se jedná o přihlašovací server, databázi nebo cloud funkce. Access token v sobě má uložené veškeré informace o uživateli a nelze ho modifikovat kvůli jeho podpisu serverem. Lze ho tedy pouze číst a používat ke komunikaci. Server tak vždy jednoznačně ví, od kterého uživatele dotaz pochází a má všechny jeho data k dispozici rovnou z tokenu, jehož pravost si ověří kontrolou podpisu.

Oproti přihlašování přes standardní session mají tokeny několik výhod. Největší výhoda je v zabezpečení. Pokud by se nějakému útočníkovi podařilo odchytit dotaz na server a ukrást access token, může ho používat jen po dobu další hodiny, což výrazně sníží napáchání škod. Bez refresh tokenu totiž nový access token nezíská. Další výhoda je v tom, že přihlašování je „stateless“. To znamená, že si server neuchovává žádná data o relaci uživatele. Všechna data lze totiž vždy spolehlivě získat rovnou z tokenu. Firebase Admin SDK umožňuje do tokenu zapisovat vlastní data, dokud nepřekročí 1 MB. To je v projektu využito při nastavování rolí uživatele v administraci. Výhodou je, že k datům v tokenu lze přistoupit odkudkoliv ve Firebase. Například z logiky bezpečnostních pravidel přístupu k databázi.

Nevýhoda tohoto systému je, že tokeny musí klient pravidelně obměňovat, aby nedošlo k chybnému dotazu z důvodu expirace tokenu. Také si klient někde musí ukládat samotný refresh token, aby se mohl později využít k získání nového access tokenu, když je aplikace nějakou dobu nepoužívaná. Tuto logiku ale naštěstí řeší Firebase SDK a programátor se tak z většiny nemusí u tuto funkcionalitu starat.

3.2 Databáze

Jako databáze je použitý Cloud Firestore. Byl zvolen hlavně kvůli své jednoduchosti a integraci s Firebase. Cloud Firestore je NoSQL databáze, takže nebylo potřeba vymýšlet databázový model typický pro relační SQL databáze. Místo toho je v databázi pouze jedna hlavní kolekce, ve které jsou uložena data všech hráčů v jednotlivých dokumentech, jejichž id je vždy UID hráče z Firebase Authentication. ID dokumentu se dá přirovnat k primárnímu klíči v relačních databázích. V dokumentech jsou pak data hráče uložena ve formátu JSON. Dále je v databázi druhá kolekce s názvem „system“, kde je například dokument s uživateli, co mají přístup do databáze nebo dokument, kam se ukládají statistiky o počtu hráčů v databázi pomocí Cloud Funkcí, co reagují na změny v kolekci s hráči.



Obrázek 12 - Struktura databáze projektu

Databáze komunikuje s připojenými klienty v reálném čase a informuje je o všech změnách v kolekcích a dokumentech, kde klienti poslouchají změny. Tímto způsobem má klient vždy aktuální data k zobrazení s maximálně pár milisekundovým zpožděním.

3.3 Logika na backendu

Backend by se neobešel bez obslužné logiky. K tomu jsou využity Cloud Functions. Plní hned několik funkcí. Jednou je reakce na změny ve Firestore databázi. Pokaždé, když se vytvoří nová uživatelská data nebo se smažou, tak se aktualizuje počítadlo v dokumentu se statistikami, stejně je tomu u reakcí na registrace a mazání uživatelů. Při smazání uživatele je zde zároveň cloud funkce, co smaže jeho příslušná hráčská data v databázi.

Dalším úkolem cloud funkcí v projektu je fungovat jako http endpoint pro nějakou akci. Nahrazují tak dedikovaný API server, který by se musel zvlášť spravovat a škálovat. Logiku

cloud funkcí mohou klienti spouštět přes Firebase SDK. Níže je seznam všech cloud funkcí, které projekt využívá.

Function	Trigger	Region	Runtime	Memory	Timeout
addAdminUser	HTTP Request https://europe-west1-riseofpilgrims.cloudfunctions.net/addAdminUser	europe-west1	Node.js 14 Beta	256 MB	60s
changePlayerName	HTTP Request https://europe-west1-riseofpilgrims.cloudfunctions.net/changePlayerName	europe-west1	Node.js 14 Beta	256 MB	60s
createPlayerData	HTTP Request https://europe-west1-riseofpilgrims.cloudfunctions.net/createPlayerData	europe-west1	Node.js 14 Beta	256 MB	60s
decrementPlayerCount	document.delete players/{playerId}	europe-west1	Node.js 14 Beta	256 MB	60s
decrementUserCount	user.delete	europe-west1	Node.js 14 Beta	256 MB	60s
deleteUserData	user.delete	europe-west1	Node.js 14 Beta	256 MB	60s
getUserInfo	HTTP Request https://europe-west1-riseofpilgrims.cloudfunctions.net/getUserInfo	europe-west1	Node.js 14 Beta	256 MB	60s
incrementPlayerCount	document.create players/{playerId}	europe-west1	Node.js 14 Beta	256 MB	60s
incrementUserCount	user.create	europe-west1	Node.js 14 Beta	256 MB	60s
listAdminUsers	HTTP Request https://europe-west1-riseofpilgrims.cloudfunctions.net/listAdminUsers	europe-west1	Node.js 14 Beta	256 MB	60s
removeAdminUser	HTTP Request https://europe-west1-riseofpilgrims.cloudfunctions.net/removeAdminUser	europe-west1	Node.js 14 Beta	256 MB	60s
saveGame	HTTP Request https://europe-west1-riseofpilgrims.cloudfunctions.net/saveGame	europe-west1	Node.js 14 Beta	256 MB	60s
setUserRole	HTTP Request https://europe-west1-riseofpilgrims.cloudfunctions.net/setUserRole	europe-west1	Node.js 14 Beta	256 MB	60s

Obrázek 13 - Seznam Cloud Funkcí v projektu

3.4 Zabezpečení

Komunikace všech částí je šifrovaná pomocí protokolu HTTPS. Přístup k databázi je řešený přes Security Rules a endpointy cloud funkcí validují příchozí data vlastní logikou.

Otázkou zabezpečení je ale i podvádění ve hře. Momentálně lze velice jednoduše poslat na endpoint pro uložení hry jakákoliv data o skóre hráče, validace proběhne bez problému a data se zapíšou do databáze. Tomuto problému nelze úplně předejít v žádné online hře. I kdyby byl endpoint nějakou logikou zabezpečen, nelze předejít tomu, aby si hráč nenapsal program, který bude číst paměť hry a ovládat postavu hráče sám.

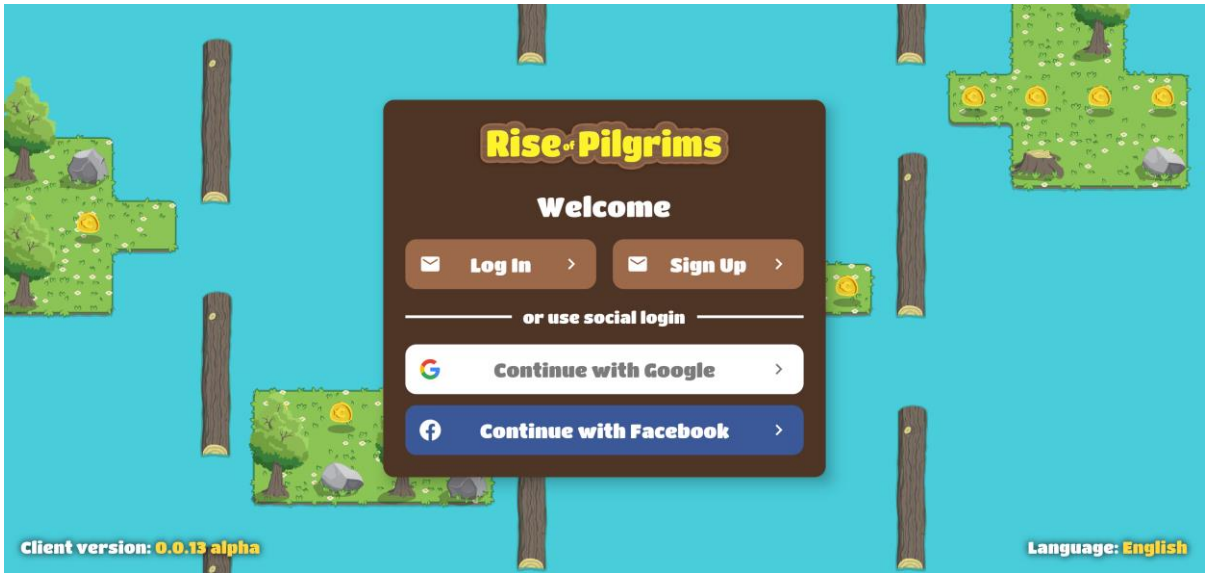
Podvádění lze ale udělat výrazně složitější, aby se o to pokoušelo mnohem méně lidí. Algoritmus, který se bohužel nepovedlo kvůli jeho složitosti implementovat do termínu odevzdání projektu by fungoval například tak, že by se na endpoint pro uložení hry spolu s výsledkem hry posílaly také data o tom, jak se k tomuto výsledku došlo. Takovými daty jsou například pohyby hráče. Server by pak mohl podle pohybů hráče a seedu hry (hodnota, která vždy vygeneruje stejný herní svět) celou hru přehrát u sebe a zkontrolovat, že pohyby sedí k danému světu. V tomto případě by ale server musel nejdříve před začátkem hry poslat hráči hodnotu seedu a očekávat stejnou hodnotu na konci hry nazpět. Tímto by se předešlo opětovnému spamování endpointu se stejným seedem a jeho řešením. Díky možnosti kompilace Dartu do JavaScriptu by šlo i jednoduše sdílet logiku generátoru světa mezi serverem a klientem. Do budoucna je tedy tento problém částečně řešitelný.

4 MOBILNÍ APLIKACE

Tato část se věnuje funkcím mobilní aplikace, jejich implementaci a napojením na Backend. Také popisuje interakci uživatele s aplikací.

4.1 Přihlášení / Registrace

Po prvním otevření aplikace se uživateli zobrazí vítací stránka s možnostmi přihlášení nebo registrace.



Obrázek 14 - Přihlašovací stránka

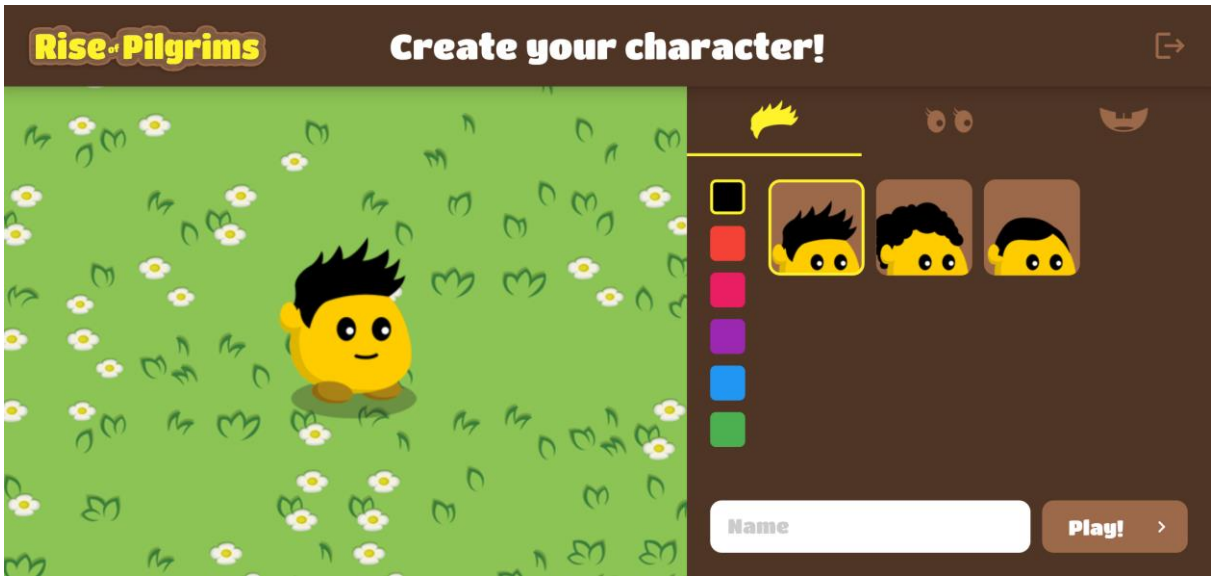
Uživatel má možnost přihlásit se přes sociální sítě nebo přes email. Pokud ještě nemá účet, může se přes email registrovat. V pravém dolním rohu obrazovky si může změnit jazyk. Na výběr je čeština nebo angličtina.

Aplikace si po uživateli při prvním úspěšném přihlášení nebo registraci vyžádá ověření emailu. Předejde se tak registraci uživatelů s falešným emailem.

Přihlašování v aplikaci je plně řešené přes Firebase Authentication.

4.2 Vytvoření postavy

Po úspěšném přihlášení je uživatel vyzván k vytvoření své herní postavy. Uživatel si může vybrat z různých částí těla v různých kategoriích a také upravovat barvy těla, vlasů a bot. Po zvolení jména postavy může uživatel pokračovat do hry.



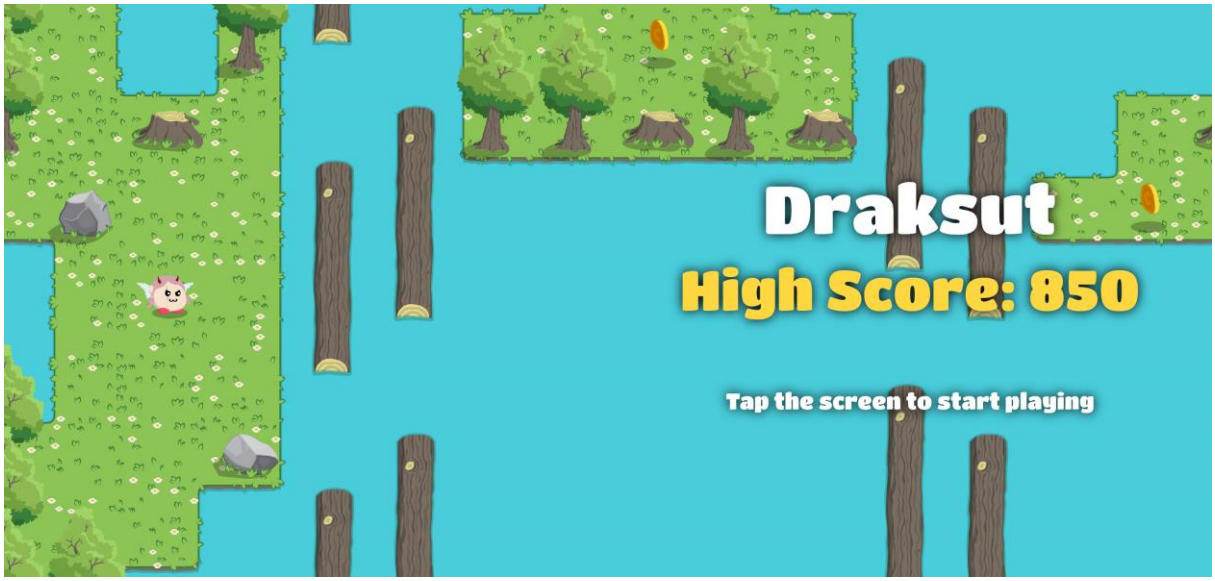
Obrázek 15 - Stránka vytvoření postavy

Na postavu bude možné v budoucnu ve hře kupovat různé předměty, které vylepšují vzhled, ale nenahrazují prvky postavy vybrané hráčem při vytvoření postavy. Předměty se v momentální fázi projektu nedají koupit, ale je implementované jejich vykreslování a lze je hráči přiřadit skrze administrační rozhraní. Předmětů je několik druhů na různé části postavy. Výsledná postava se potom skládá z doplňků a částí těla v určitém pořadí. Níže jsou znázorněné vybrané jednotlivé části a popsané jejich skládání.

Obrázek 16 - Popis postavy

4.3 Hratelná (aktivní) část

Hratelná část je hlavní a nejkomplexnější část projektu. Do hratelné části hry se hráč dostane po vytvoření postavy kliknutím na tlačítko play nebo z hlavního menu hry, opět tlačítkem play. Po vstupu do hratelné části se na obrazovce objeví hráčovo jméno, jeho nejvyšší skóre a výzva pro započatí hry klepnutím na obrazovku. Na pozadí se nachází vygenerovaný začátek herního světa spolu s postavou hráče, kterou následně bude hráč ovládat.



Obrázek 17 - Hratelná část

4.3.1 Herní svět

Herní svět je v podstatě mřížka, kde každá buňka představuje jednu souřadnici. Na každé souřadnici může být umístěno několik herních objektů. Objektů je několik základních typů. Níže je podrobněji popsaný každý z nich.

4.3.1.1 Dlaždice (tiles)

Dlaždice jsou základním a nejjednodušším objektem ve světě. Na každé souřadnici může být pouze jedna dlaždice určitého druhu. Ve hře jsou 2 druhy dlaždic - voda a tráva. Dlaždice se nemohou hýbat, ale lze je dynamicky v průběhu ničit nebo vytvářet nové. Ostatní objekty mohou zjistit, na které se momentálně nacházejí dlaždici, a podle toho reagovat. Například entita hráče se na dlaždici s vodou utopí, ale na trávě může stát bez úhony.

4.3.1.2 Překážky (obstacles)

Překážek je ve hře také několik druhů a platí pro ně stejná pravidla jako pro dlaždice – lze mít pouze jednu překážku jednoho druhu na dané souřadnici. Překážky se dále dělí na průchozí a neprůchozí. Mezi neprůchozí překážky patří strom, keř, kámen a pařez. Entita hráče na ně nemůže vstoupit, pohyb na jejich souřadnici je zablokován. Průchozí překážou je naopak například past. Hráč může vstoupit na souřadnici s pastí, ale past následně sklapne a entitu hráče zničí, což znamená konec hry.

4.3.1.3 Efekty (effects)

Efekty jsou objekty, které většinou nejsou vidět, ale slouží pro uchovávání dat na určité souřadnici, ke kterým mají potom přístup ostatní objekty. Efektů je opět několik druhů, ale na rozdíl od dlaždic může být na jedné souřadnici více efektů, podmínkou je, že musí být různého druhu. Příkladem efektu je například proud vody. Entity klády a leknínu se podle proudu na souřadnici hýbou. Dalším efektem je generátor klád. Jedná se o efekt, který podle efektu proudu na stejné souřadnici generuje entity klád, které následně plují po proudu pryč.

4.3.1.4 Entity (entities)

Entity jsou nejkomplexnějším objektem ve hře. Entity nejsou vázané na souřadnici, může jich být neomezeně mnoho a neomezeně druhů na jednom místě zároveň. Jsou také jediným objektem, který může libovolně dynamicky měnit svou pozici ve hře a jelikož není vázaný na souřadnici, tak lze entitu umístit například i na rozhraní 4 dlaždic. Mezi entity patří leknín, kláda, penízek a hráč. Každá entita má vlastní logiku toho, jak se má chovat vzhledem k dlaždicím, překážkám, efektům a jiným entitám, se kterými sdílí souřadnice. Mimo to lze také dvou entitám nastavit, co se má dít při jejich kolizi. Například při kolizi entity hráče a entity klády se hráč připojí ke kládě a jeho souřadnice se odvíjí relativně od entity klády. Tímto je hráč přichycen k pohybující se kládě. Stejná logika je platná pro leknín. Při kolizi hráče a penízku se zase zničí entita penízku a hráčovi se připíše 1 k počtu sesbíraných penízků.

4.3.2 Chunky

Herní svět se dělí na menší části (chunky). Každý chunk představuje oblast 5x9 souřadnic ve světě a obsahuje všechny dlaždice, překážky a efekty v této oblasti. Chunky slouží k jednodušší manipulaci se světem. Například k ohraničení oblasti pro vykreslování nebo k hromadnému smazání objektů, co jsou již mimo oblast kamery. V budoucnu lze tento systém také využít k případné serializaci dat rozehrané hry a uložení na disk.

4.3.3 Vykreslování světa

Vykreslování světa (rendering) hra řeší přes Flutter Canvas API a knihovnu Flame. Kombinací těchto dvou rozhraní implementuje vlastní algoritmus renderování specifický pro tento projekt. Samotné vykreslování probíhá v několika krocích, které jsou níže popsány.

4.3.3.1 Získání chunků pro render

Nejdříve má algoritmus za úkol zjistit, co bude vykreslovat. Svět je z již dříve vysvětlených důvodů rozdělený do chunků. Každý chunk je objekt, který v oblasti 5x9 souřadnic ve světě obsahuje všechny dlaždice, překážky a efekty. V rámci úspory výpočtů se vždy vykreslují jen chunky, které jsou v zorném poli kamery. Ty se získají podle souřadnic a velikosti objektu kamery ze světa, který má všechny chunky uložené v tabulce chunků podle jejich souřadnic. Tabulka je tvořená vnořenými kolekcemi typu HashMap s indexací podle typu int pro jednotlivé souřadnice.

4.3.3.2 Vykreslování dlaždic

Po získání chunků se jako první vykreslí všechny dlaždice z vybraných chunků. Vykreslování dlaždic je velmi náročná operace, protože každá dlaždice se skládá z 16 poddlaždic, které zaručují správný tvar výsledného terénu v návaznosti na ostatní dlaždice téhož typu. Každá poddlaždice má také několik variací, díky kterým výsledný terén nevypadá repetetivně. Toto znamená, že jen pro dlaždice se musí každý snímek vykreslit stovky obrázků. Toto byl z počátku problém pro knihovnu Flame, každá poddlaždice totiž byla vlastní sprite (obrázek ve hře) a pro vykreslení takového objektu Flame na pozadí zavolá Flutter Canvas API metodu drawImageRect. Tato metoda ale není nejrychlejší, protože každé její zavolání způsobí nový vykreslující příkaz na nižší úrovni SKIA, nad kterou je Flutter postavený. Tyto příkazy jsou velice pomalé a je prioritou je v aplikaci minimalizovat. Hra byla při vykreslování dlaždic tímto způsobem v podstatě nehratelná.

Jako řešení se nabídl objekt z knihovny Flame s názvem SpriteBatch, který na rozdíl od objektu Sprite dokáže vykreslit více objektů v jednom příkazu. Podmínkou ale je, že se všechny objekty musí nacházet na stejné textuře. Na pozadí SpriteBatch při vykreslení volá metodu z Flutter Canvas API s názvem drawAtlas. Ta přijímá list s transformacemi a velikostmi objektů na textuře, které následně všechny vykreslí v jednom jediném příkazu. Ve výsledku zredukuje stovky příkazů z původního řešení pouze na jeden jediný příkaz na chunk, takže asi 4 na snímek.

4.3.3.3 Vykreslování překážek, efektů a entit

Jako poslední krok se vykreslí zbytek všech herních objektů. Jelikož je zbytek objektů vykreslován naprosto totožně bez ohledu na typ objektu, všechny objekty proto implementují rozhraní (v dartu mixin) Renderable, které definuje 3 vlastnosti – vrstvu, pozici v sloupci a pozici v řádku. Pozice v sloupci a řádku se v základu odvodí od skutečné pozice objektu, lze je však přepsat, vrstva je v základu 0.

Vrstva je číslo, které určuje, jaký objekt se má vykreslit nad jiným objektem. Například pokud má entita hráče vrstvu 1 a entita klády vrstvu 0, entita klády bude vždy vykreslena pod entitou hráče. Jinými slovy, entita klády se vykreslí dříve než entita hráče, protože má nižší vrstvu.

Pozice v sloupci je číslo, které určuje řazení objektu v rámci sloupců. Ve hře platí, že čím nižší tato hodnota je, tím dříve bude objekt vykreslen.

Pozice v řádku je číslo, které určuje řazení objektu v rámci řádků. Ve hře platí, že čím nižší tato hodnota je, tím dříve bude objekt vykreslen.

Překážky a efekty si algoritmus vytáhne z již získaných chunků. Dále k nim přidá entity, které se namísto z chunků vytahují z listu definovaném přímo v objektu herního světa. Entity totiž díky svým vlastnostem nejsou vázané na chunk a vždy se vykreslí všechny existující.

Algoritmus všechny objekty seřadí podle vlastností popsaných výše, nejdříve řadí podle vrstvy, dále podle pozice ve sloupci, a nakonec podle pozice v řádku. Tímto je jednoznačně definované, jak se budou objekty vykreslovat. Seřazené objekty se postupně vykreslí a algoritmus se tak pro daný snímek ukončí.

4.3.4 Generátor světa

Generátor světa má na starosti vytvářet svět v průběhu hraní hry hráčem. Vždy, když kamera zjistí, že ve své oblasti nemá žádné další chunky, pošle signál generátoru, aby nějaké vytvořil. Algoritmus generátoru pracuje se segmenty. Segment je objekt, který definuje určitý způsob generování – například proud s kládami nebo travnaté ostrůvky. Každý segment má zároveň list navazujících segmentů s procentuálními šancemi na jejich vygenerování po momentálním segmentu. Generátor tedy tímto způsobem skládá segmenty za sebe.

Dělení na segmenty je důležité hlavně z důvodu synchronizace entit. Entity ve stejném segmentu se vygenerují zároveň, tudíž jsou spolu časově synchronizované a nestane se například, že by se za sebou vygenerovalo několik leknínů, které nepůjde proskákat, protože jejich pohyby nebudou navazovat z důvodu desynchronizace.

Každý segment může po svém vygenerování předat dalšímu segmentu metadata, podle kterých následující segment může upravit pravidla generování. Typicky se předávají souřadnice, u kterých se ví, že z nich hráč může opustit momentální segment, aby následující segment na těchto místech uvolnil hráčovi cestu.

Generátoru lze v programové logice předat seed pro generování. Seed je počáteční hodnota generátoru, od které se pak odvíjejí další náhodné hodnoty. Stejný seed tak vždy vygeneruje stejný svět.

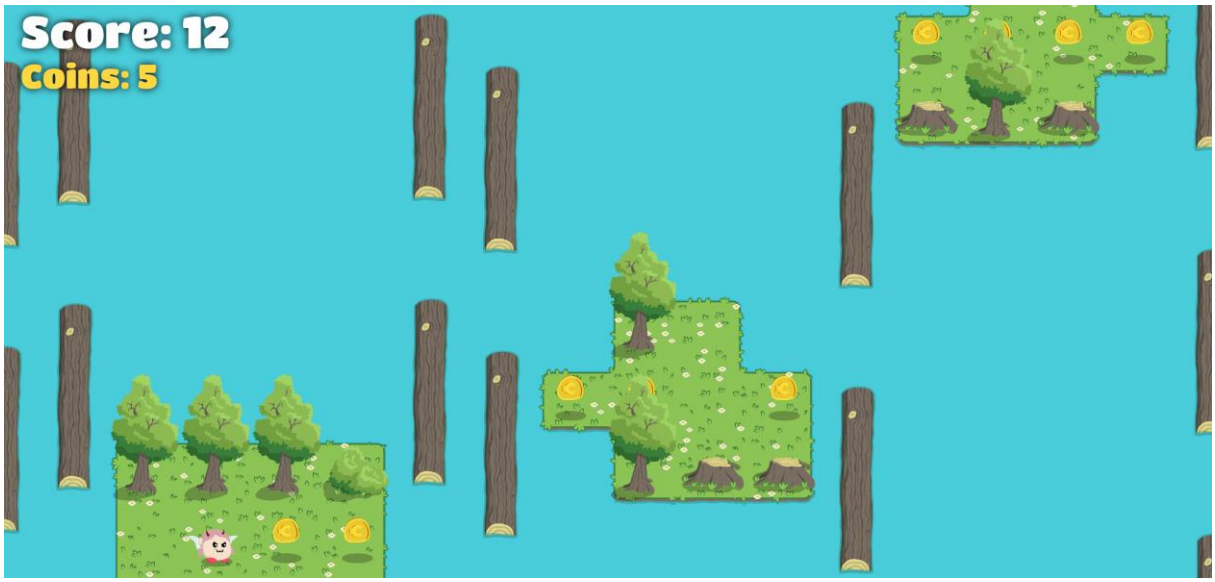
4.3.5 Průběh hry

Po klepnutí hráčem na obrazovku se spustí samotná hra. Klepnutí zároveň způsobí posunutí hráčovy postavy o jednu souřadnici doprava. Cílem hráče je dostat se v herním světě co nejdále do pravé strany, každý skok směrem doprava přičte hráči 1 bod.

Hráč může svou postavu ovládat dvěma způsoby – klepnutím nebo tažením prstem. Tažením postava skočí buď horizontálně nebo vertikálně do směru tažení na vedlejší souřadnici. Klepnutím vždy skočí o jednu souřadnici doprava.

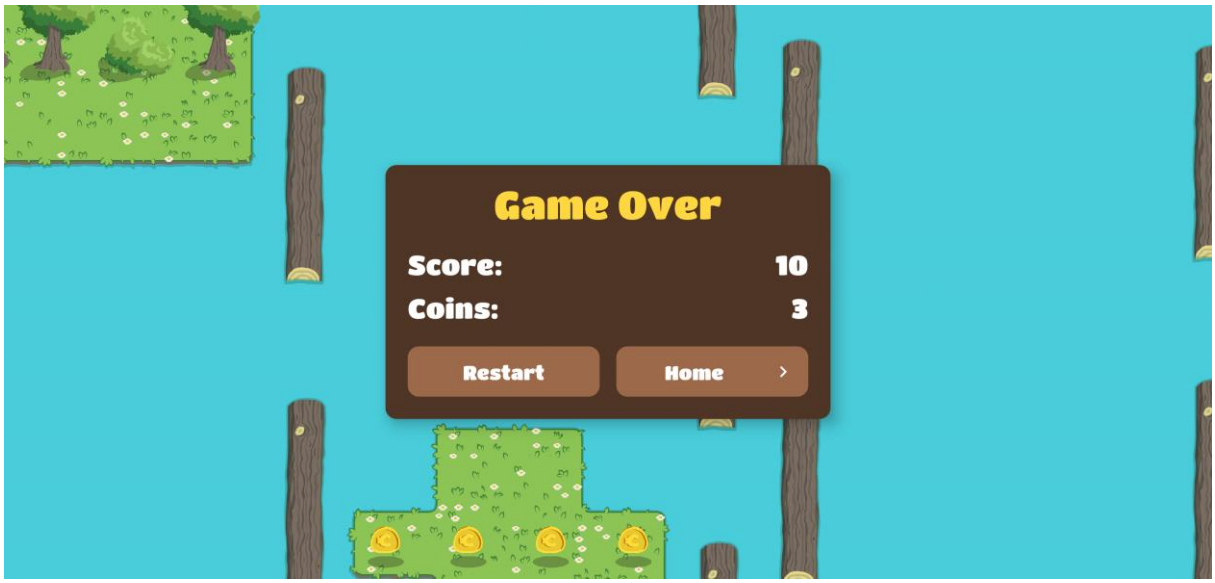
Okamžikem započatí hry se také začne pohybovat doprava kamera, když bude hráč moc pomalý a kamera (okraj obrazovky) ho dožene, tak se hra ukončí. Ukončení hry nastane i v případě, že hráč skočí do vody, šlápne na past nebo ho kláda, na které se nachází, odnese mimo obrazovku.

Hráč v průběhu hry sbírá penízky – měnu ve hře, která půjde v budoucnu využít k nakupování kosmetických doplňků na postavu. Počet penízků a své skóre může hráč vidět v levém horním rohu obrazovky po celou dobu hraní.



Obrázek 18 - Ukázka zobrazení skóre a penízků

Po ukončení hry se hráčovi zobrazí dialogové okno s výsledky hry. Výsledky se zároveň přičtou k předchozím hodnotám v databázi a hráč poté může hrát znovu tlačítkem restart nebo se vrátit do hlavního menu tlačítkem home.



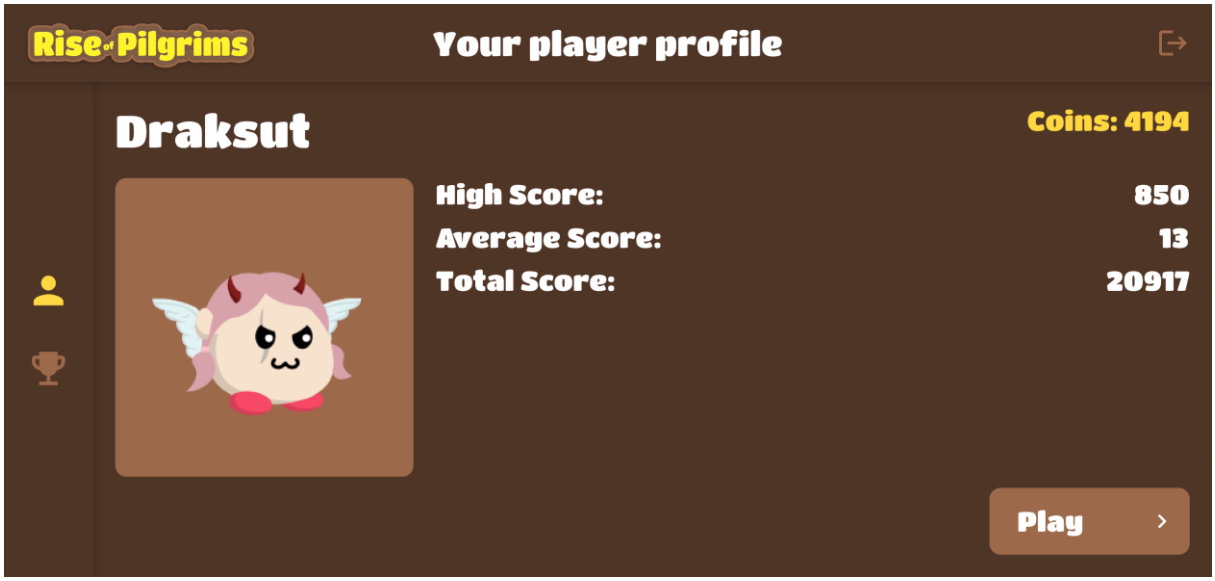
Obrázek 19 - Konec hry

4.4 Hlavní menu

Hlavní menu je hlavní stránka aplikace mimo herní část. Má několik podstránek, mezi kterými lze navigovat pomocí menu po levé straně. Z hlavního menu se jde pomocí tlačítka play přemístit do herní části hry nebo se odhlásit pomocí tlačítka pro odhlášení.

4.4.1 Profil hráče

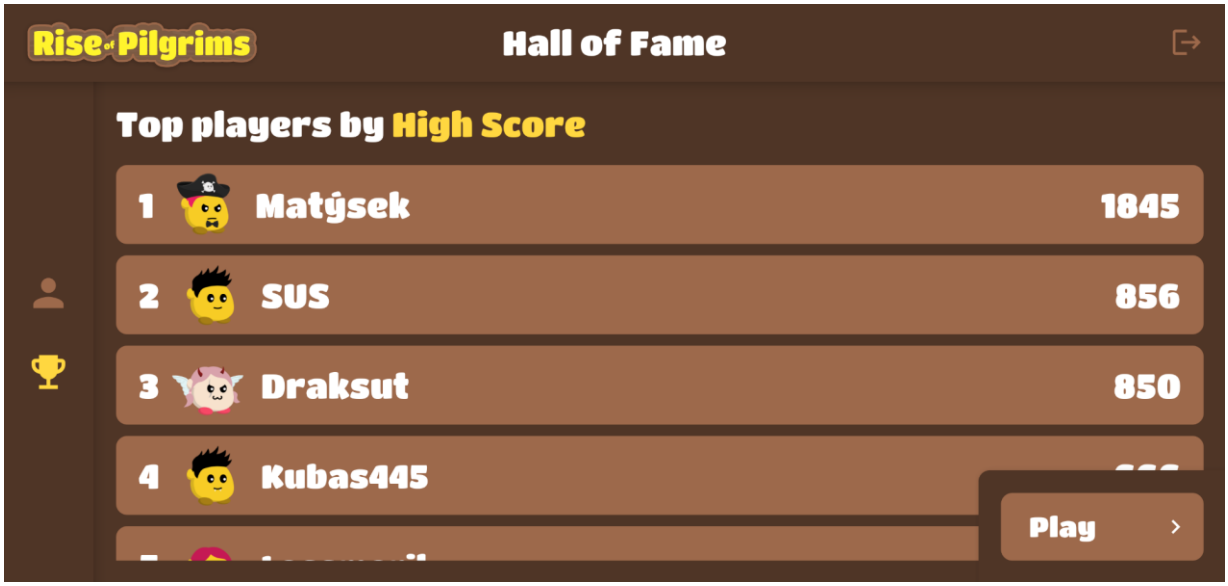
Profil hráče je hlavní podstránka hlavního menu. Je zde vidět hráčova postava, jméno, jeho celkové skóre, průměrné skóre, nejvyšší skóre a také počet nasbíraných penízků.



Obrázek 20 - Profil hráče

4.4.2 Síň slávy

Síň slávy je podstránka hlavního menu, kde lze vidět globální žebříček hráčů. Řadí se podle nejvyššího skóre hráče. Každý záznam obsahuje jméno, obrázek postavy a hodnotu skóre.



Obrázek 21 - Síň slávy

5 WEBOVÁ ADMINISTRACE

Následující část se věnuje webové administraci projektu. Jedná se o webovou aplikaci napsanou ve frameworku NuxtJS, která slouží administrátorům hry pro zobrazení statistik a ke správě hráčů. Do administrace se mohou dostat pouze uživatelé, kteří mají přístup přidělený jiným uživatelem, který má do administrace přístup. Uživatelé v administraci mohou mít buď roli editora nebo administrátora. Administrace se nachází na webové adrese https://admin.riseofpilgrims.com/

5.1 Přihlášení

Uživatel se nejdříve musí do administrace přihlásit. Přihlašuje se zde stejnými údaji jako v samotné mobilní aplikaci hry. Tudíž uživatel musí být nejprve ve hře zaregistrován skrz mobilní aplikaci. Pokud uživatel nemá jiným uživatelem přidělený přístup do administrace, bude po přihlášení přesměrován na stránku se zprávou, že nemá přístup. V opačném případě bude přesměrován na výchozí stránku administrace se statistikami.

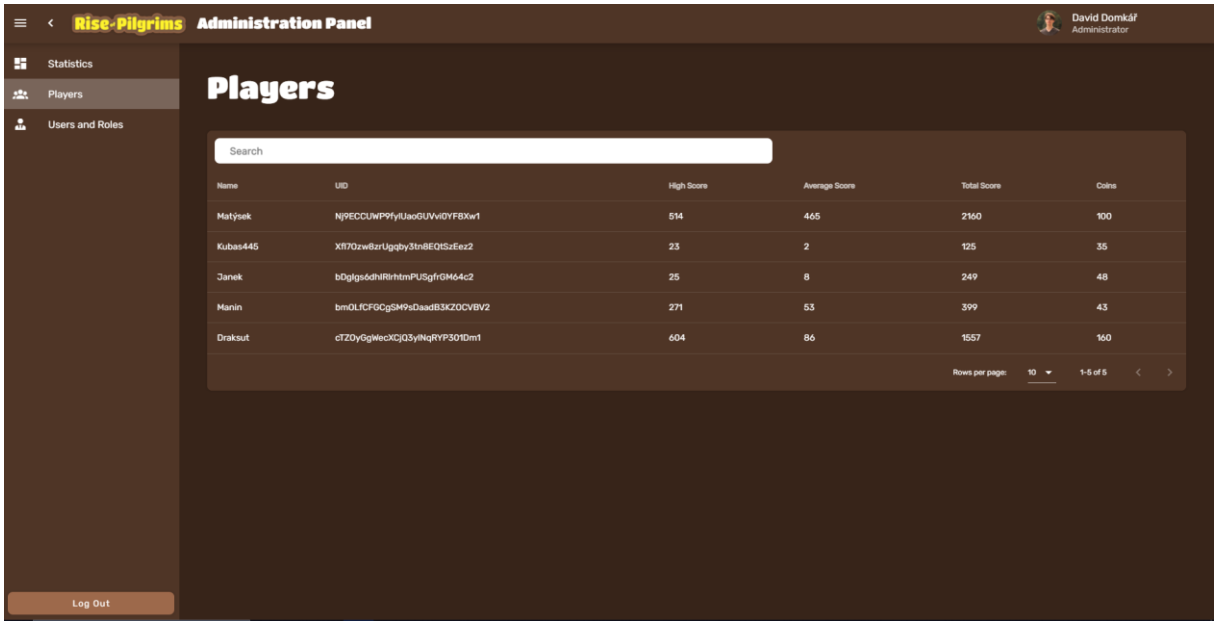
5.2 Zobrazení statistik

První sekcí v administraci je zobrazení statistik. Jsou zde vidět základní informace, jako například počet registrovaných hráčů. Dále jsou zde 3 žebříčky hráčů řazené podle nejvyššího, průměrného a celkového skóre hráče. Po kliknutí na hráče v žebříčku je uživatel přesměrován na stránku s detailem daného hráče. Všechny zobrazené informace se v reálném čase synchronizují se změnami v databázi, tudíž mají administrátoři vždy nejaktuálnější přehled o stavu hry.

Obrázek 22 - Zobrazení statistik

5.3 Seznam hráčů

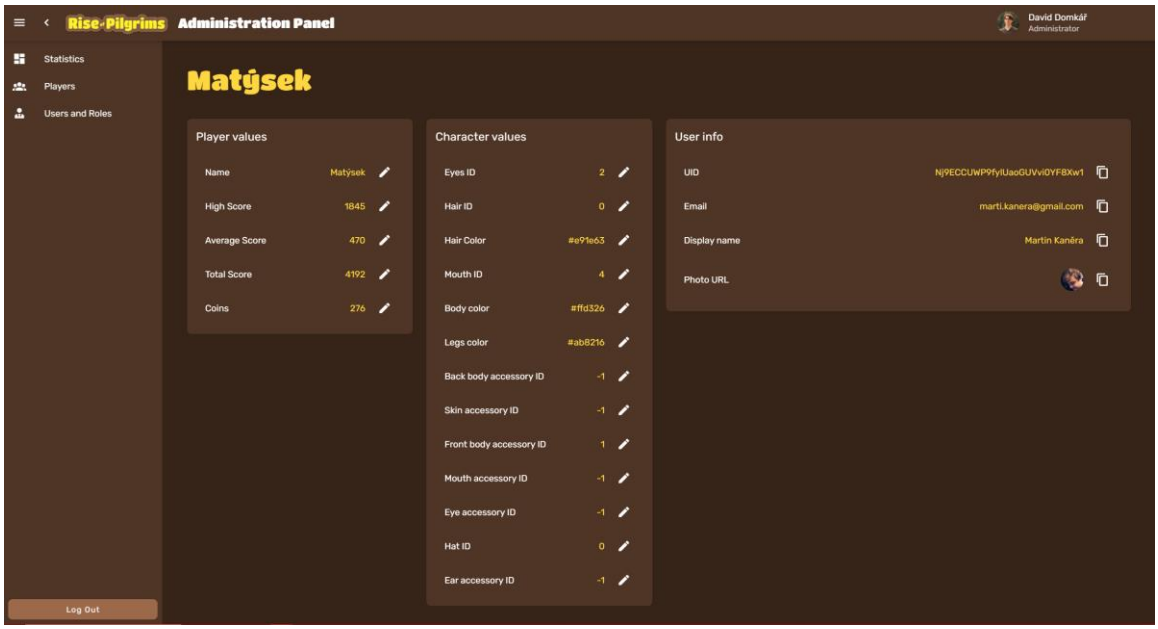
V seznamu hráčů lze vidět tabulku všech hráčů ve hře. Lze je řadit a vyhledávat podle různých kritérií, jako je například jméno hráče nebo dosažené skóre. Po kliknutí na záznam hráče v tabulce je uživatel přesměrován na stránku s detailem hráče.



Obrázek 23 - Seznam hráčů

5.4 Detail hráče

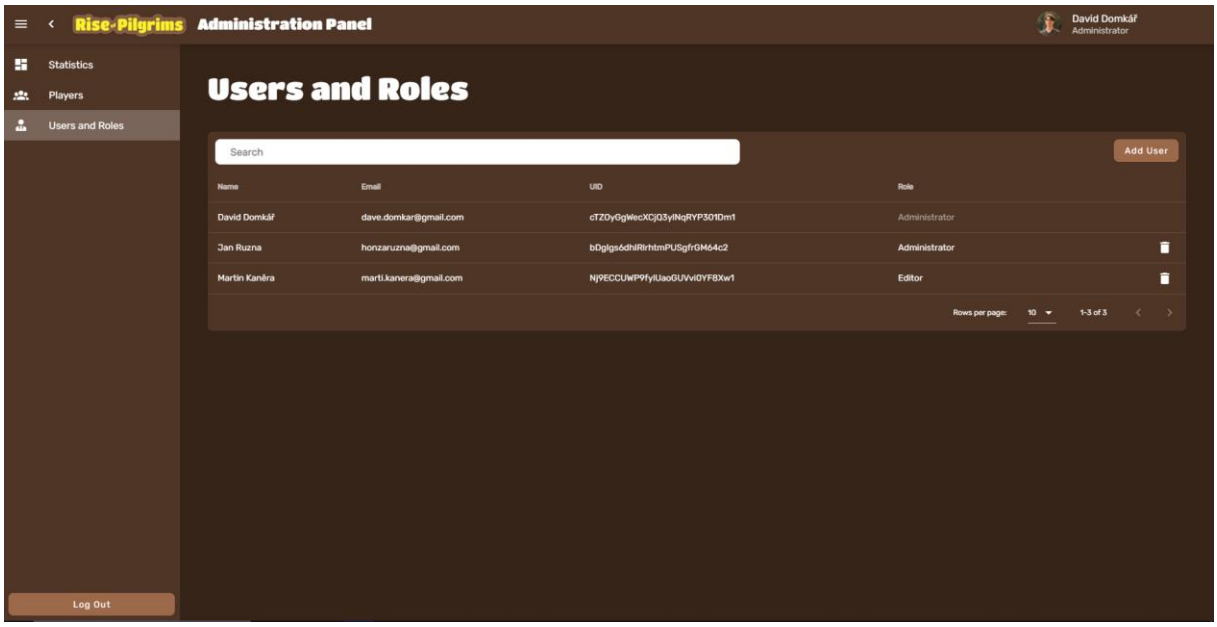
Stránka s detailem hráče slouží k pokročilé správě hráče. V případě nutnosti tu lze hráči upravit jakékoliv hodnoty, jako například jméno, skóre nebo vzhled postavy, kliknutím na editační tlačítko vedle dané hodnoty. Také jsou zde k vidění informace, které hráč poskytl při registraci do hry – email, jméno a profilový obrázek, které lze jednoduše zkopírovat do schránky kliknutím na ikonu vedle dané hodnoty.



Obrázek 24 - Detail hráče

5.5 Uživatelé a role

Poslední sekcí v administraci je správa uživatelů administrace a jejich oprávnění. Toto je jediná sekce administrace, kam mohou jen uživatelé s rolí administrátor. Lze zde vyhledávat a řadit existující uživatele administrace podle různých kritérií. Existující uživatele je možné smazat nebo jim změnit roli. Také lze přidat nového uživatele kliknutím na tlačítko „Add User“ v pravém horním rohu tabulky. Každý uživatel může mít buď roli editor nebo administrátor. Role editor umí vše, co role administrátor, ale uživatelé s touto rolí nemají přístup do sekce uživatelé a role, tudíž nemohou měnit oprávnění jiným uživatelům. Roli momentálně přihlášeného uživatele je možné vidět na horní liště administrace po pravé straně.



Obrázek 25 - Uživatelé a role

6 ZÁVĚR

Podařilo se vytvořit základní verzi mobilní hry spolu s administrační částí. Projekt má v současné chvíli implementované potřebné funkce pro registraci, vytvoření postavy, hraní samotné hry a porovnání svého skóre s ostatními uživateli. Administrátoři hry mají kontrolu nad hrou z administračního rozhraní.

Do budoucna bude potřeba projekt rozšiřovat o další funkce, jako je například obchod s doplňky a dostat projekt do fáze veřejného beta testování na platformách pro distribuci aplikací. Tyto věci jsou v plánu do konce roku 2021.

Po úspěšném testování bude následovat propagace projektu k získání většího povědomí o hře veřejností.

7 REFERENCE

Cherepanov, I. (14. září 2019). *What I Learned Developing Games from Scratch with Flutter | by Ivan Cherepanov | ITNEXT*. Získáno 19. března 2021, z Medium: https://itnext.io/what-i-learned-developing-games-from-scratch-on-flutter-a2eda59460e5

FireSlime. (nedatováno). Získáno 19. března 2021, z Flame engine: https://flame-engine.org/

flame-engine.org. (nedatováno). *flame 1.0.0-rc8 | Flutter Package*. Získáno 19. března 2021, z Dart packages: https://pub.dev/packages/flame/versions/1.0.0-rc8

flame-engine.org. (nedatováno). *forge2d | Dart Package*. Získáno 19. března 2021, z Dart packages: https://pub.dev/packages/forge2d

Google LLC. (nedatováno). Získáno 19. března 2021, z Dart packages: https://pub.dev/

Google LLC. (nedatováno). Získáno 19. března 2021, z Dart programming language | Dart: https://dart.dev/

Google LLC. (nedatováno). Získáno 19. března 2021, z Flutter - Beautiful native apps in record time: https://flutter.dev/

Google LLC. (nedatováno). Získáno 19. března 2021, z Skia Graphics Library: https://skia.org/

Google LLC. (nedatováno). *Documentation | Firebase*. Získáno 19. března 2021, z Firebase: https://firebase.google.com/docs

Google LLC. (nedatováno). *Flutter architectural overview - Flutter*. Získáno 19. března 2021, z Flutter - Beautiful native apps in record time: https://flutter.dev/docs/resources/architectural-overview

NuxtJS. (nedatováno). *Nuxt.js - The Intuitive Vue Framework*. Získáno 19. března 2021, z https://nuxtjs.org/

Sierra, F. G. (19. května 2018). *Working with Firestore: Building a simple database model | by Francisco García Sierra | ProAndroidDev*. Získáno 19. března 2021, z Medium: https://proandroiddev.com/working-with-firestore-building-a-simple-database-model-79a5ce2692cb

Sullivan, M. (4. ledna 2019). *Flutter: Don't Fear the Garbage Collector | by Matt Sullivan | Flutter | Medium*. Získáno 19. března 2021, z Medium: https://medium.com/flutter/flutter-dont-fear-the-garbage-collector-d69b3ff1ca30

Vuetify. (nedatováno). Získáno 19. března 2021, z Vuetify — A Material Design Framework for Vue.js: https://vuetifyjs.com/

Wenger, J. (5. listopadu 2018). *Demystifying Firebase Auth Tokens | by Jacob Wenger | Medium*. Získáno 19. března 2021, z Medium: https://medium.com/@jwngr/demystifying-firebase-auth-tokens-e0c533ed330c

You, E. (nedatováno). *Single File Components — Vue.js*. Získáno 19. března 2021, z Vue.js: https://vuejs.org/v2/guide/single-file-components.html

8 SEZNAM OBRÁZKŮ