УДК 004

Использование Google Firebase в приложении на Unity

Фатеенков Данила Витальевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье описан процесс интеграции Google Firebase с Android-приложеним, разработанном с использованием Unity. Рассматривается предоставляемый функционал платформой Firebase, а также реализация программного кода на языке программирования C# для работы с Google Firebase.
Ключевые слова: Unity, Google Firebase, C#

Using Google Firebase in a Unity app

Fateenkov Danila Vitalievich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article describes the process of integration of Google Firebase with Android application developed using Unity. The functionality provided by Firebase platform is considered, as well as the implementation of programming code in C# programming language to work with Google Firebase.
Keywords: Unity, Google Firebase, C#

1. Введение

1.1 Актуальность

Google Firebase – платформа, поставляющая услуги в сфере облачных технологий, была разработана в 2011-м году. Данная платформа используется большим количеством разработчиков, так как позволяет надёжно интегрировать серверную составляющую приложения. Платформа предоставляет большое количество услуг: от баз данных до хранилищ и модулей машинного обучения.

Платформа тесно связана с Android-разработкой и является частью функционала программного обеспечения Android Studio. Также Firebase доступен для интеграции в приложения, которые были разработаны с использованием Unity.

Google Firebase продолжает развиваться, улучшая уже существующий функционал, и продвигаться в Android-сегменте разработчиками. Для Android-сферы данная платформа является одной из наиболее актуальных среди разработчиков за счёт удобства использования и доступности предоставляемых услуг.

1.2 Обзор исследований

Е.Д. Куликова и Н.А. Косов описали реализацию аутентификации в Android-приложении с использованием Firebase Auth, которое создано в Unity [1]. К.О. Атеев описал процесс создания для аутентификации в веб-приложении, для реализации был использован JavaScript [2]. М.К. Долматова описала картографическое приложение и процесс его создания на основе свободных данных OpenStreetMap с использованием Google Firebase [3]. И.М. Минвалеев описал разработку мобильного приложения контроля и управления доступом, использовался язык программирования Java и Firebase Realtime Database для контроля доступа [4]. И.М. Минвалеев провел исследование на наличие уязвимостей и выявил преимущества Firebase Realtime Database перед другими БД [5].

1.3 Цель исследования

Цель – описать работу и реализацию функционала платформы Google Firebase в приложении на Unity, которое разрабатывается для Android. Написанный код должен работать с базой данных, которая предоставляется платформой Firebase.

2. Материалы и методы

Для реализации поставленной задачи использован язык программирования C# и среда разработки Unity.

3. Результаты и обсуждения

Одним из главных этапов интеграции Google Firebase в Android-приложение, которое было разработано на Unity, это настройка конфигураций приложения и установка расширений, предоставляющие функционал для интеграции.

Firebase предоставляет разработчику обширный набор инструментов для разработки: облачное хранилище, базы данных, контроль доступа к приложения (методы аутентификации), модели машинного обучения, отчёты об ошибках, аналитику и много других функций, который может задействовать разработчик.

После создания проекта в Google Firebase, необходимо зарегистрировать приложение. В регистрацию входит один пункт: платформа, под которую разрабатывается приложение (необходимо также указать название приложения и название пакета (или ID) приложения).

ID приложения доступно в настройках проекта в Unity и имеет вид “com.*разработчик*.*название приложения*”. Расположено ID во вкладке “Player” настроек проекта (см. рис. 1).

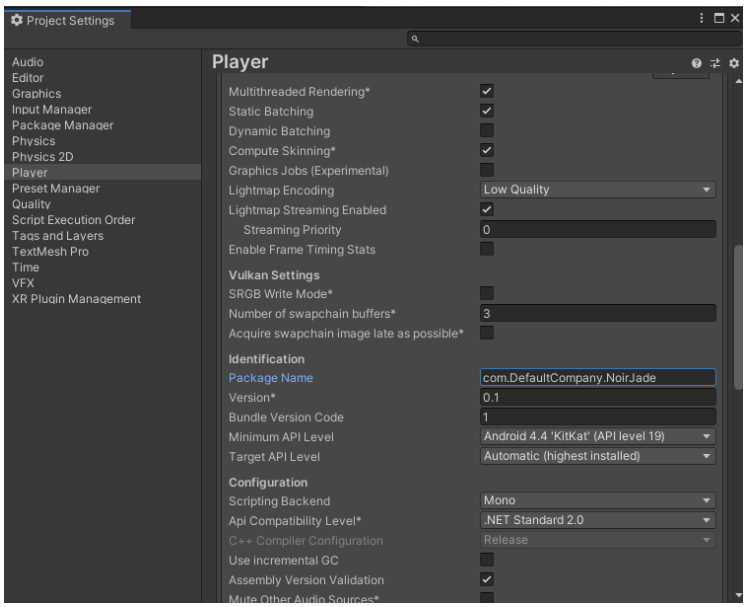

Рисунок 1. Имя пакета приложения в настройках проекта

После регистрации приложения, генерируется json файл, который представляет собой конфигурацию приложения. Данный файл необходим для корректной работы приложения с Google Firebase. Он помещается в корень проекта, а именно в папку “Assets”. Данный файл содержит необходимую, но не секретную информацию: API-ключи, ID клиента для OAuth протокола, а также URL вашего проекта в Firebase и ID проекта.

После добавления файла конфигурации, необходимо скачать Google Firebase SDK. Данный пакет включает в себя:

1. Firebase Analytics – пакет, реализующий соединение и передачу данных модулю, который предназначен для аналитики работоспособности приложения, а также для отчётности и статистики.
2. Firebase Auth – пакет, необходимый для реализации функциоанала авторизации в приложении. Следует использовать, если в приложении осуществляется авторизация и регистрация пользователей через соответствующий модуль платформы Firebase (данный модуль расположен в консоли разработчика и называется “Authentication”).
3. Firebase Crashlytics – пакет, который позволяет получать отчёты об ошибках, возникших при работе с Firebase. Crashlytics позволяет оперативно отслеживать возникшие баги при установке соединения с базой данных и выполнений запросов к серверу.
4. Firebase Database – пакет, содержащий в себе необходимый для выполнения запросов к базам данных на сервере в реальном времени. Realtime Database – один из двух типов БД, предоставляемых платформой. Отличительной особенностью является представление данных – БД представлена в виде json и синхронизируется с клиентом в реальном времени.
5. Firebase Dynamic Links – пакет для работы с динамическими ссылками. Динамические ссылки позволяют адаптировать и упростить работу с приложением, если пользователь находится вне него, но ссылка ссылается на это приложение (например, если пользователь открывает ссылку в браузере на

Android устройстве, то ему будет предложено открыть/установить приложение, для которого данная ссылка предназначена).

6. Firebase Firestore – пакет для работы с базами данных Firestore. БД Firestore представлены в виде таблиц и более привычны для разработчиков (реляционная модель БД). Являются одним из основных компонентов при работе с Google Firebase.

7. Firebase Functions – пакет, содержащий функции для работы с Firebase. Функции задаются разработчиком, и они передаются в приложение, которое синхронизировано с проектом в Firebase. Необходим пакет в случае безопасности передачи информации, так как код функции сохранён в панели разработчика и доступ к нему никто другой получить не может.

8. Firebase Messaging – API для реализации кросс-платформенной системы передачи уведомлений и сообщений между разработчиком и пользователями приложения. Модуль Firebase Cloud Messaging (FCM) позволяет загружать код для передачи информации пользователям, после чего система обрабатывает его и выполняет запрос.

9. Firebase Remote Config – пакет для реализации дистанционного управления конфигурацией приложения.

10. Firebase Storage – API для работы с хранилищем файлов на сервере. Позволяет загружать и выгружать файлы и данные в разных форматах для дальнейшей работы (например, загрузка файлов игры с сервера, которые могут занимать много места при скачивании apk файла).

Необязательно загружать в проект все представленные пакеты, но рекомендуется установить: Messaging, Crashlytics и Remote Config. Также отдельно можно загрузить пакет Admob, предназначенный для монетизации приложения путём добавления рекламы.

После загрузки всех нужных для разработки приложения пакетов, необходимо добавить зависимости и проверку на наличие сервисов Google Play на устройстве, на котором запускается приложение.

Одним из важнейших модулей является Cloud Firestore. На данный момент API для Unity позволяет создавать, изменять и удалять записи в базах данных Firestore. Данные представлены в виде документов, в которых описываются основные характеристики. Данные по одинаковым характеристикам могут и должны быть собраны в коллекции (но коллекции могут и не иметь однородные данные). Но работать с коллекциями на данный момент нельзя.

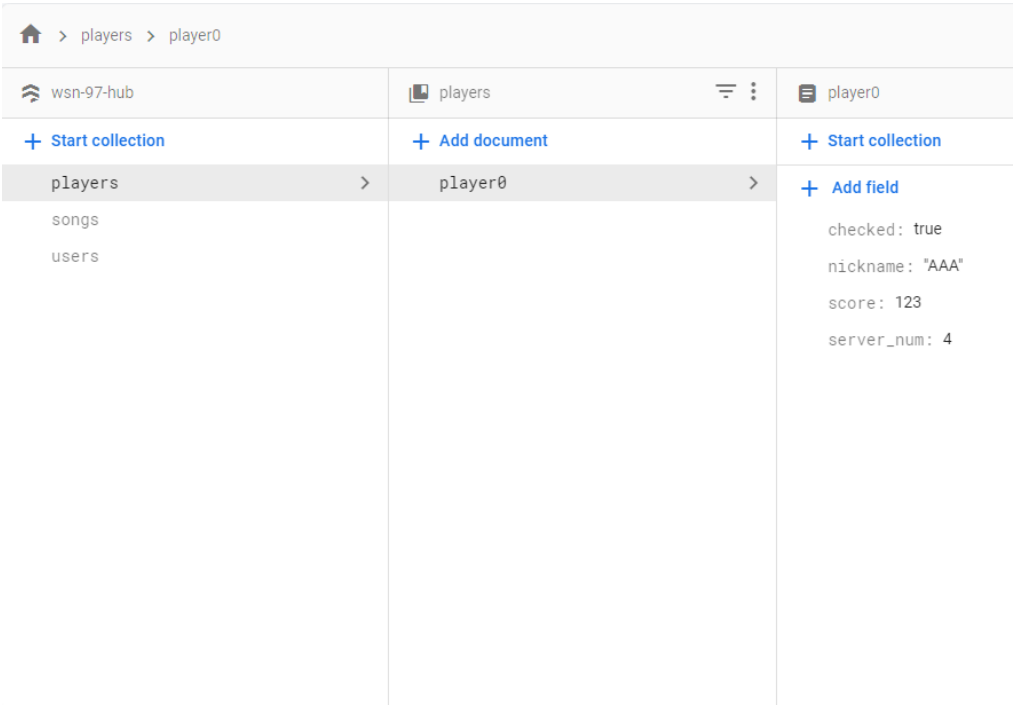

Рисунок 2. Структура Firestore

Для добавления документа, необходимо создать ссылку на коллекцию, в которую он должен быть помещён. Для этого создаётся новый объекта типа данных "DocumentReference", который ссылается на коллекцию ранее заданной БД. Задать базу данных необходимо заранее следующим образом: вводится новый объект вида "FirebaseFirestore", который является ссылкой на базу данных:

```
FirebaseFirestore DB_inst=FirebaseFirestore.DefaultInstance;
```

После создания ссылки на БД, можно сослаться на коллекцию и документ, применяя методы "Collection()" и "Document()":

```
DocumentReference
doc=DB_inst.Collection("players").Document("player5");
```

В скобках метода "Collection()" указывается название коллекцию, на которую необходимо сослаться, а в скобках "Document()" название документа из коллекции. Если документа в коллекции нет с указанным названием, то он будет создан при добавлении данных.

Для загрузки данных в документ используются словари, состоящие из двух ключевых характеристик: ключ/название параметра и значение данного параметра. Ключ всегда имеет строковый тип данных, а его значение представлено в виде объекта (так как для разных параметров значение может быть представлено в разных типах данных):

```
Dictionary<string,object> player = new
Dictionary<string,object> {
```

```
        {"nickname", PlayerName},
        {"score", PlayerScore},
        {"server_num", PlayerServer},
        {"checked", false}
    };
```

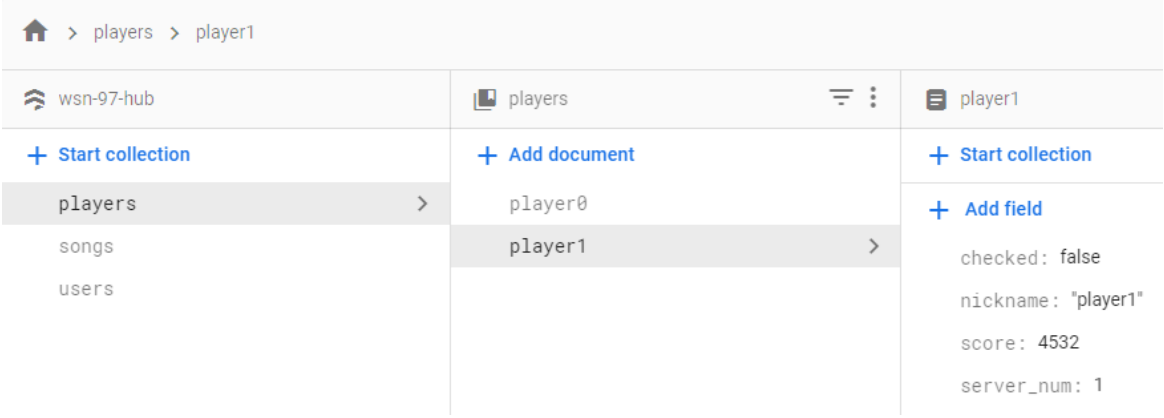

Рисунок 3. Новый документ в Firestore, после выполнения C# скрипта

Для сохранения изменений в существующем документе или нового документа в целом, используется функция "SetAsync()". Данная функция является частью модуля "Firebase.Firestore.SetOptions" и предназначена для внесения изменений в БД Firestore. "SetOptions" включает в себя несколько основных параметров, которые могут быть использованы при вызове функции "SetAsync()":

1. MergeAll - объединить новые поля с существующими в документе.
2. Overwrite - переписать уже заданные характеристики в документе.

Данные параметры используются при работе с уже существующими документами.

Пример записи нового документа: doc.SetAsync(player);

Пример изменения уже существующего документа: doc.SetAsync(player, SetOptions.MergeAll);

Для изменения составляющей документа необходимо применить функцию "UpdateAsync()". Для этого необходимо сослаться на уже существующий документ в коллекции. В самой функции указывается словарь с характеристиками и новыми значениями для них:

```
    DB_inst.Collection("players").Document("player1").UpdateAsyn
c(player);
```

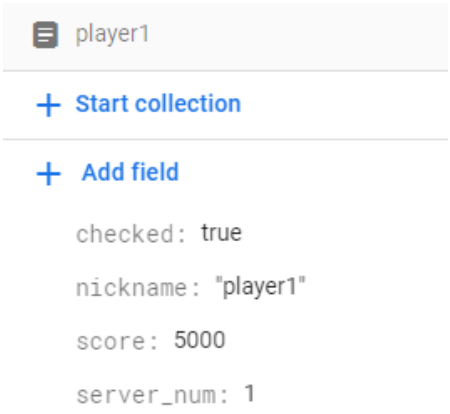

Рисунок 4. Документ “player1” после изменения значения полей “checked” и “score” через “UpdateAsync()”

Также есть возможность добавлять экземпляры классов в коллекции. Для этого необходимо сослаться на сам экземпляр и передать значения его параметров в словарь. Экземпляры удобны для создания нескольких документов, так как API на данный момент не поддерживает передачу массивов данных, поэтому легче через внутренний цикл загрузить все необходимые экземпляры классов.

Кроме добавления новых данных в БД, есть возможность получать уже существующие данные и использовать их для дальнейшей работы. Для этого используется функция "GetSnapshotAsync()". Результат запроса будет возвращён в виде типа данных DocumentSnapshot, содержащий всю информацию, которая сохранена в документе. DocumentSnapshot легко конвертируется в словарь, для этого есть метод "ToDictionary". Пример получения документа из коллекции Firestore.

Можно получать с коллекции сразу несколько документов:

```
    Query query =
DB_inst.Collection("players").WhereEqualTo("score",1000);
```

Для этого вместо "Document" можно использовать параметры запроса, по которым выбираются документы (например "WhereEqualTo"), а также тип возвращаемого значения будет не DocumentReference, а Query. Существуют следующие методы, используемые в API для задания запросов к БД:

1. WhereEqualTo(*параметр*,*значение параметра*) – возвращает все документы, в которых необходимый параметр (указанный в функции) соответствует заданному значению.
2. WhereLessThan() – возвращает все документы, в которых необходимый параметр (указанный в функции) меньше заданного значения.
3. WhereLessThanOrEqualTo() – все документы, в которых необходимый параметр (указанный в функции) меньше заданного значения или равен ему.

4. WhereGreaterThan() – все документы, в которых необходимый параметр больше заданного значения.

5. WhereGreaterThanOrEqualTo() – все документы, в которых необходимый параметр больше заданного значения или равен ему.

6. WhereNotEqualTo() – все документы, в которых необходимый параметр не равен заданному значению (не поддерживается в Unity на данный момент).

7. WhereArrayContains() – документы в массиве, значения нужного параметра соответствует указанному в методе. Функция принимает 2 входных параметре – характеристика/поле документа и значение, которому поле должно соответствовать.

8. WhereIn() – все документы, значение указанного поля входит в заданный массив. Функция принимает также 2 входных параметра: поле документов и массив, в котором проверяется наличие значения поля.

9. WhereArrayContainsAny() – документы, значение заданного поля которых соответствует любому значению массиву, который также задан в функции.

Главным ограничением при использовании данных методов состоит в том, что нельзя использовать методы, определяющие определённый промежуток (то есть операции сравнения <, <=, >, >= и !=) для разных полей документа (но можно использовать данные методы для одного поля).

Также можно задавать ограничения и пределы при поиске и выводе данных с БД. Предел задаётся с помощью функции "Limit()", в скобках которой указывается количество документов. Для сортировки документов используется функция "OrderBy()", в скобках которой указан параметр, по которому сортируется список документов.

"Limit()", "OrderBy()" и операции сравнения "Where" можно комбинировать и формировать более комплексный запрос:

```
    Query query =
DB_inst.Collection("players").WhereGreaterThan("score",1000).Ord
erBy("score").Limit(10);
```

Ограничение при работе с "Limit" и "OrderBy" такое же, что и с операциями сравнения: нельзя выполнять эти функции в одном запросе для отличного от операций сравнения поля документа.

Также можно удалять документы с коллекции. Для этого существует фукнция "DeleteAsync()". Данная функция удаляет документ полностью из коллекции, но перед этим необходимо задать название документа, который нужно удалить:

```
    DocumentReference doc =
DB_inst.Collection("players").Document("player0");
    doc.DeleteAsync();
```

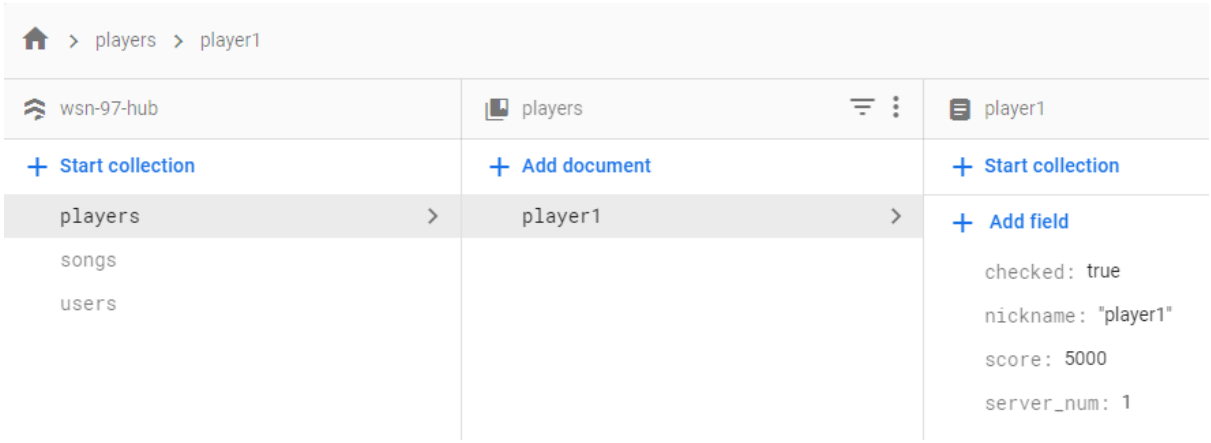

Рисунок 5. Коллекция “players” после выполнения C# скрипта, который удаляет документ с названием “player0”

Помимо удаления документов, можно удалять отдельные поля в документах. Для этого используется метод "Delete" класса "FieldValue":

```
    DocumentReference doc =
DB_inst.Collection("players").Document("player3");
    Dictionary<string,object> delData = new
Dictionary<string,object> {
        {"nickname", FieldValue.Delete}
    };
```

Удалять коллекции на данный момент нельзя. API для Unity не предоставляет функционала для работы с коллекциями, можно только ссылаться на них.

Firestore также позволяет импортировать и экспортировать данные, но это можно сделать только в консоли разработчика. Данный модуль позволяет эффективно работать с пользовательскими данными и создавать необходимый функционал в приложении. Одним из примеров использования Firestore может служить доска лидеров в игре:

```
    FirebaseFirestore db = FirebaseFirestore.DefaultInstance;
    Query q = db.Collection("players").WhereEqualTo("checked",
true).OrderBy("score").Limit(10);
```

Данный запрос вернёт 10 игроков с наивысшим количеством очков, у которых параметр “checked” принимает истинное значение.

Второй модуль, который является одним из наиболее значимых для разработчика - Cloud Storage. Облачное хранилище позволяет загружать файлы и использовать для дальнейшей разработки. При работе с проектом на Unity, это полезно, если итоговый проект занимает слишком много места и для удобства можно разделить приложение на 2 части: apk содержит необходимый функционал для работы приложения, а все файлы (музыка, текстуры, модели и т.д) расположены в облачном хранилище. Файлы при первом запуске

скачиваются с облачного хранилища, и пользователь может продолжить пользоваться приложением.

API для Unity позволяет производить следующие действия с хранилищем:

1. Скачивать файлы;
2. Загружать файлы;
3. Удалять файлы;
4. Получать информацию о файлах;

Перед началом работы с хранилищем задаётся подключение:

```
FirebaseStorage storage = FirebaseStorage.DefaultInstance;
```

Для получения файлов используется функция, задающая ссылку на файл “GetReference()”. После введения ссылки, можно скачать файл 4-мя способами:

1. Скачать файл по ссылке. Для этого необходимо использовать метод “GetDownloadUrlAsync()”. Также необходимо использовать WWW или UnityWebRequest для обработки запроса.
2. Скачать массив, данные в котором представлены в типе данных byte. Необходимо задействовать метод “GetBytesAsync()”, предварительно задав максимальный размер файла. В массив сохраняется результат выполнения запроса.
3. Использование потока для скачивания файлов. Для этого применяется метод “GetStreamAsync()”. Метод позволяет выполнять скачивание параллельно другим потокам в приложении, что может помочь повысить работоспособность приложения.
4. Загрузка локальных файлов. Используется метод “GetFileAsync()”, который загружает файл с устройства в приложение.

Перед началом загрузки задаётся ссылка на файл:

```
StorageReference imgReference =
storage.GetReference(“picture.png”);
```

Для загрузки файлов в хранилище также необходимо задать ссылку на файл, только в этот раз это будет ссылка для хранилища – где и под каким именем необходимо сохранить файл. Для реализации такой ссылки используется метод “Child()”, который в качестве параметра принимает путь, по которому будет сохранено изображение (также можно сохранять в корне хранилища, если указать только название файла). Также необходимо вызвать метод “PutFileAsync()” или метод “PutBytesAsync()”:

```
string fileLoc = “images/holder.png”;
StorageReference imgReference =
storage.Child(“images/holder.png”);
imgReference.PutFileAsync(fileLoc);
```

Удаление файлов происходит схожим образом: создаётся ссылка на файл и вызывается метод “DeleteAsync”, после которого файл с хранилища удаляется:

```
StorageReference imgReference =
storage.Child(“images/holder.png”);
imgReference.DeleteAsync();
```

Таким образом, выше было показано, как устроена работа с двумя основными модулями Firebase: Firestore и Cloud Storage. На примере этих двух модулей был представлен код и показана работа API в приложении, созданном в Unity.

В статье был рассмотрен API, предоставляемый Google Firebase, для приложений, которые созданы с использованием Unity для Android устройств.

Библиографический список

1. Куликова Е.Д., Косов Н.А. Реализация аутентификации с помощью Firebase в Unity3d // Актуальные научные исследования в современном мире. 2021. № 6-1 (74). С. 72-75.
2. Атеев К.О. Создание простой аутентификации с использованием технологий React и firebase // Инновации. Наука. Образование. 2020. № 23. С. 322-334.
3. Долматова М.К. Мобильное приложение для визуализации маршрутов по городу и планов зданий университетов // Сборник избранных статей научной сессии ТУСУР. 2018. № 1-3. С. 20-22.
4. Минвалеев И.М. Разработка системы контроля и управления доступом // Научно-технический вестник Поволжья. 2018. № 4. С. 140-142.
5. Минвалеев И.М. Сравнительная характеристика различных видов баз данных СКУД // Научно-технический вестник Поволжья. 2018. № 4. С. 143-145.
6. Unity - Manual: Unity User Manual 2020.3 (LTS) URL: https://docs.unity3d.com/Manual/index.html. – Дата доступа: 15.06.2022.
7. Firebase URL: https://firebase.google.com. – Дата доступа: 13.06.2022.