BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

MOBILE GAME FOR PRACTICING ENGLISH VOCABULARY

MOBILNÍ HRA PRO PROCVIČENÍ ANGLICKÉ SLOVNÍ ZÁSOBY

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **ANTONÍN HUBÍK**
AUTOR PRÁCE

SUPERVISOR **Ing. TOMÁŠ DYK**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2021/2022

Bachelor's Thesis Specification

Student: **Hubík Antonín**
Programme: Information Technology
Title: **Mobile Game for Practicing English Vocabulary**
Category: Information Systems

Assignment:

1. Familiarize yourself with modern education trends using mobile devices to enforce learning capabilities, mobile applications development, and cloud services as a support framework for mobile game infrastructure.
2. Design a multi-platform mobile game for practicing English based on the rules of the game "word soccer", meaning to help users expand their vocabulary. The game will have an online mode for two players.
3. Implement the designed mobile game using tools from the .NET ecosystem.
4. Test the implemented game on a selected range of users and analyze their feedback.

Recommended literature:

- Van der Vegt, W., Nyamsuren, E. and Westera, W., 2016, June. RAGE reusable game software components and their integration into serious game engines. In *International Conference on Software Reuse* (pp. 165-180). Springer, Cham.
- Reynolds, M., 2014. *Xamarin Essentials*. Packt Publishing Ltd.
- Fustino, R., 2018. *Azure and Xamarin Forms: Cross Platform Mobile Development*. Apress.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Dyk Tomáš, Ing.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 11, 2022
Approval date: November 3, 2021

Abstract

The goal of this thesis is to design and implement a multi-platform educational mobile game for practicing and expanding the English vocabulary. The game is based on the rules of word chain and similar word games and implemented as a minimal prototype of an online player versus player mode. The application is implemented in C# and XAML, making use of the .NET platform. Parts of the solution, which were ready for production environment, were deployed making use of suitable Microsoft Azure Cloud services . An Android version of the application was published locally and tried on a selected group of users in closed testing. Results of the testing have determined the state of the solution upon completing the thesis and have been used as basis for proposing further improvements above current technical shortcomings and limitations.

Abstrakt

Cílem této práce je navrhnout a implementovat multiplatformní vzdělávací mobilní hru pro procvičování a rozšiřování anglické slovní zásoby. Hra je založena na pravidlech slovní kopané a jí podobných her se slovy a obsahuje minimální prototyp režimu online hry dvou hráčů proti sobě. Aplikace využívá platformu .NET, kde je implementována v jazycích C# a XAML. Části řešení dostatečně připravené pro produkční prostředí byly nasazeny s využitím vhodných cloudových služeb Microsoft Azure. Aplikace na Android byla publikována lokálně a ozkoušena na vybrané skupině uživatelů v rámci uzavřeného testování. Výsledky testování posloužily k určení stavu řešení v okamžiku dokončení práce a na na jejich základě byla navržena možná vylepšení nad rámec technických nedostatků a omezení.

Keywords

Web service, mobile application, educational game, English, vocabulary, word chain, Android, C#, .NET, Xamarin, cloud, Azure, serverless

Klíčová slova

Webová služba, mobilní aplikace, vzdělávací hra, angličtina, slovní zásoba, slovní kopaná, Android, C#, .NET, Xamarin, cloud, Azure, serverless

Reference

HUBÍK, Antonín. *Mobile Game for Practicing English Vocabulary*. Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Tomáš Dyk

Mobile Game for Practicing English Vocabulary

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Dyk. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Antonín Hubík
May 10, 2022

Acknowledgements

Chtěl bych tímto poděkovat panu inženýru Dykovi za trpělivé vedení této závěrečné práce, a dále panu inženýru Pluskalovi za dříve poskytnuté konzultace.

Contents

Chapter 1

Introduction

Use of digital devices in language learning is by no means a new concept. Having been around for more than half a century, it has undercome significant evolution since its beginnings, most notably in the past two decades, when not only the use of personal computers, but also the of cell phones and later smart mobile devices broke into mainstream. These new technologies have inspired and enabled a broad spectrum of multimedia, including digital games, however lately perhaps the most significant development has been seen on the aforementioned smart platforms. With smartphones or tablets being perceived as increasingly affordable, widely available, practical and high-performance, the mobile app market has grown significantly in the past decade and with this a trend has been set of attempts at evaluating possible benefits of mobile applications, often games, in learning languages.

With that, various solutions have been developed, meant to assist learners with their English, many of them aim to balance engaging features with educational goals. This thesis aims to determine presence of proof of this attitude's possible benefits, explore existing implementations of similar-to-target scale and complexity and underlying technologies and finally design and prototype a simple mobile game for practicing English vocabulary.

The core idea of this proposed application is player versus player matches with rules based on the English language game of word chain, accompanied by a dictionary/get definition feature. By the very nature of this game, focus is on providing a compact option of interacting with English vocabulary in a comparatively rather unstructured way, based largely on association, creativity and motivation by competition.

This report opens with a chapter about game-based and digital game-based learning and its use and benefits when learning English. The game of word chain is described.

In the following chapter, a selection of existing mobile games of similar-to-target scale are elaborated on, in order to provide a more complete picture of the current situation.

Chapter 4 presents an overview of languages, tools and versions integrated together as the .NET ecosystem.

Further on, basic theory on cloud technologies and a selection of viable cloud options are described. Argumentation for utilization of cloud technologies is presented.

Chapter 6 then summarizes design of the application. Target audience is specified, application and game concepts are elaborated on, a high-level overview of architecture is shown and selected implementation technologies are listed.

In chapter 7 development process and an overview of the resulting solution, including more in-depth architecture overview, implementation specifics and current limitations.

To close the topic, chapter 8 provides information about the current state of deployment. User feedback is collected and state of the solution is evaluated.

Chapter 2

Digital Game-based Learning

According to [21] Game-based learning (GBL) can be defined as *"an environment where game content and game play enhance knowledge and skills acquisition, and where game activities involve problem solving spaces and challenges that provide players/learners with a sense of achievement."*

The above definition favors a broad variety of practical implementations, such as simple single-player or small-group card and board games on one hand, opposed to complex massively multiplayer online games and alternate reality games on the other [18].

The latter approach to implementing and deploying educational games falls under a subcategory of digital game-based learning (DGBL), which refers to the usage of entertaining power of digital games to serve an educational purpose [20]. These games are composed of multimedia elements, such as text, voice and images and it is desirable that they offer high level of interactivity. They can enhance attention, interest and creativity of users in relation to the practiced subject. Studies have shown that digital game-based learning can also increase the ability to engage in critical thinking [36]. Because of these findings and constant technological progress, the utilization of digital game-based learning for education purposes is on the rise [36], with its advantages and limitations being constantly explored and evaluated.

2.1 Game-based Learning vs. Gamification

It may be useful to immediately point out the differences between *game-based learning* and *gamification*, as these two terms are often incorrectly perceived as interchangeable and used in each other's stead [37]. In contrast with the above provided definition of game-based learning, that suggests, that the idea of game-based learning is "learning through playing", gamification is defined as *"the application of typical elements of game playing (e.g. point scoring, competition with others, rules of play) to other areas of activity, typically as an online marketing technique to encourage engagement with a product or service"* [29] or *"the process of adding games or game-like elements to something (such as a task) so as to encourage participation"* [17].

For us the important difference there is that in *game-based learning* learners achieve their educational goals by playing a game, while gamification for the purposes of education focuses on combining traditional approach to learning with non-essential experience-enhancing elements such as competition and reward [37]. Since the goal of this work is to create a game, we are interestend in *game-based learning*, not *gamification*.

2.2 Elements and Aspects Of Digital Games

Not all games that are able to some degree serve the purpose of DGBL can be considered educational. Some of them may have been created solely for the purpose of entertainment [2]. The below subsections briefly highlight some the most important factors and frequent elements of both these general categories, but with more focus on educational games, where these have to be selected and utilized in a way fitting the projected learning and other outcomes. The highlighted aspects were selected based on literature [24, 12] and own experience with digital games.

2.2.1 Game Mechanisms

Game mechanisms are what enables smooth functioning of the game world and promotes player actions under designer goals. They include freedom and interaction [24]. The latter is where educational games have the upper hand above traditional learning (see beginning of this chapter).

Interaction can be broadly distinguished as interaction with the game and interaction between players. The presence and ratio of these two is an important design choice that ties into design choices in other game factors. In addition, educational games must take into consideration the aforementioned projected outcomes when deciding how to promote them through encouraging individual effort versus cooperation or competition.

2.2.2 Game Fantasy and Progress

Feeling of immersion in game fantasy along with meaningful and well-signified in-game progress have proven to be essential for player engagement. There are several components, which can be combined to achieve this sense of goal completion or game and personal progression and provide the player with strong incentive to keep playing. The most prominent ones are:

- **Narrative** – Progressing through narrative and experiencing its story and visuals is provenly effective to retain many players' interest. In addition, it may not only signify progress, but also help with language skills – it has been noted that learners respond to language in narrative context with more permanent results than with classic lessons and grammar/vocabulary drills [6]. Some studies on the other hand point out that complex narrative may result in a heavy cognitive load that drives focus away from learning [12].

- **Level progression** – Linear progression through game levels, usually while gradually increasing difficulty and complexity, is yet another well established means for players to keep track of their progress, while being constantly pushed forward toward more formidable challenges, building upon what they've previously learned. This form is typical for entertainment arcade games, but can also translate into working toward educational goals. Level progression in educational games may resemble a lesson-based approach, but unlike lesson structure, it aims to keep in player sensation.

- **Score/Experience points** – serve as a measure of players' in-game success. When combined with narrative or level progression, points are often additionally used to modify speed of progression or directly influence game mechanics, for example availability and quality of player's actions or traits - experience points (XP). When used

in Player-versus-Player (PvP) games with focus on competition and user-interaction, their importance is greatly increased as it determines a victor and/or becomes the main metric of players' skill and to-date improvement (score points). Either the context, points may serve for determining progress **players' personal levels/ranks**. Score and XP can complement each other in shaping player experience.

2.2.3 Rewards

Existing games commonly use additional reward to boost player engagement. Reward particularly is also frequently used for gamification (see 2.1). Widespread and arguably effective rewards are **badges**. These can be put in a game to be on permanent display in a locked or otherwise incomplete state, teasing players to collect and complete them. The conditions for acquiring such badges range broadly from completing specific one-time tasks (achievements), to amassing points and levelling up (levels/ranks) or retaining badges through long-term tasks that are forfeit by breaking a routine (status badges).

Another quite common example of reward is access to **non-essential** content (e.g. cosmetic changes), either directly or through purchase using received **in-game currency**.

2.2.4 Social Elements

In present-day age of interconnectivity, many games are either based around or heavily incorporating social elements, with the trend even no longer exclusive to multiplayer titles. Social elements in games can have important game mechanics built around them, but on top of that, advantageously for learning, they facilitate sense of community.

Friend/following lists may enable smoother cooperative as well as competitive player interactions and bring notion of company.

Groups and guilds further expand on this experience with creating more closely bound player relationships, in which players may help each other and work together to achieve in-game goals.

Finally, dynamically changing **Rankings and leaderboards** allow players or groups to compare themselves to others based on different criteria, while fostering their competitive side or encouraging them to get in touch with more experienced players for help and advice. Access to these options is all the more desirable in context of improving real-life knowledge and skills.

2.3 (Digital) Game-based Approach To Learning English

Learning English is often considered stressful, especially by those who study it as a second or foreign language, which, together with the general perception of memorizing vocabulary and grammar as a boring activity, affects motivation and achievements of learners [36]. GBL and DGBL are possible remedies for these issues and are being increasingly put in practice and experimented with. Positive effects of these attitudes have for instance been witnessed in traditional education systems, where they have the power to change the role of students from passive participants of teaching and learning process to active [18].

Furthermore, it was concluded that DGBL especially has the potential to increase students' motivation and make the process more enjoyable [36]. One of the reasons for this may be that learners often base their GBL expectations on home use of games [25] and thus begin with more open, excited attitude.

2.3.1 DGBL and Mobile Assisted Language Learning

Digital games can be played on a variety of platforms, multi-purpose or dedicated ones. Because of rapid development of wireless technology and mobile electronic devices, especially over the past decade, DGBL is also often combined with the idea of mobile-assisted language learning (MALL) [36].

Mobile assisted language learning is a subset of computer assisted language learning that utilises any form of handheld mobile device. Its possibilities range from attitudes like (arguably inconvenient and outdated) SMS learning [35] to incomparably more advanced language training mobile applications meant for smartphones, that make use of wide high-resolution screens, big storage and memory and fast processing chips [35]. Part of the latter attitude's appeal to MALL today is the constantly demonstrated convenience and practicality of smartphones, that enables learning to become a part of life [36]. It is also there, where ideas of MALL and DGBL most often come together.

2.3.2 Word Chain

The game of word chain also reffered to as "grab on behind" or "last and first" (and commonly known as "slovní kopaná" in Czech language) is a simple game with words based around the idea of coming up with a start word and then thinking about another word starting with a letter or a sequence of letters from the end of the current one and continuing this way for an unspecified number of repetitions, thus creating a chain. In theory it can be played in a group of any size consisting of people of any age, as long as they can agree on a language to use, or even individually as a self-challenge mental exercise activity.

Apart from these basic principles, the exact rules of the game vary from instance to instance and may contain elements such as a time limit for coming up with the next word or a (usually applied) ban on repeating the same words in the same game. It is also possible to restrict the words accepted to only certain parts of speech (e.g. nouns) and include/exclude given, family or local names, etc.

Word chain is commonly utilised when teaching English to both children and adults, especially when taught as their second language, and on that account can be found among tips for in-class activities for practicing and expanding students vocabulary and improving spelling abilities [11, 5]. Positive influence of word chain on student engagement and English literacy was also documented in experiments - for instance a study conducted in Indonesia on a group of 39 8th grade students, where presented output data showed that interest, activity and joy accompanied the learning process [1].

It is also worth noting that this game can alternatively be restricted to only vocabulary from a specific field of interest (geography, famous contemporary/historical figures, ...) and in this form serve as a tool for practicing a range of different subjects.

There are various existing implementations for smart devices based on the idea of word chain or other similar word games. Apart from mobile games, examples of which are going to be discussed in chapter 3, there are also solutions for smart voice assistants, such as two analogous implementations for Amazon Alexa and Google Assistant, both simply called *Word Chain*.

Chapter 3

Analysis of Existing Mobile Language Games In English

There is a significant range and amount of language-focused mobile applications in English, from very comprehensive lesson-based ones, such as the vastly popular Duolingo[1], to language games like puzzles, brain teasers or competitive multiplayers. Applications of the former kind often put to use many game elements, resulting in strongly gamified learning environments, however as long as they are built around the idea of systematically passing through lessons, they are not games above all else [37]. They are therefore omitted in this chapter, as it is focused on applications that are more arguably mainly gameplay-driven.

From 10 researched mobile English language games with educational aspects, four were selected for the purpose of this thesis. All of them share a common trait of being at least partially focused on vocabulary and spelling, in a similar or even identical (3.4) way to the game of word chain (see section 2.3.2). Each game was played, analyzed and finally assessed in search of positive and possibly negative aspects.

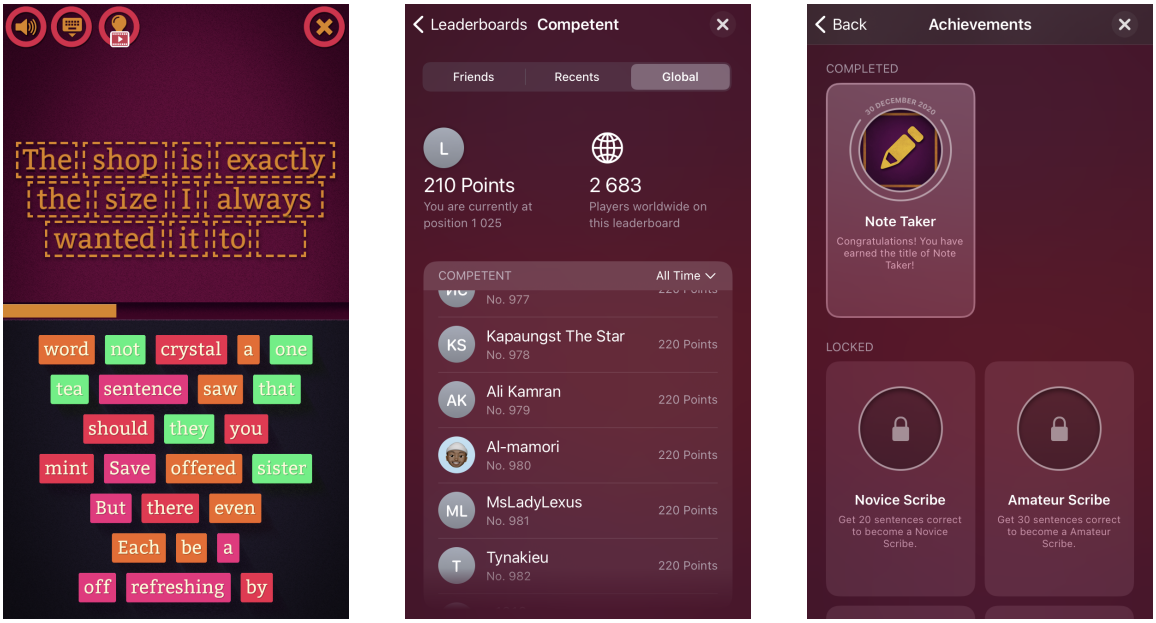
3.1 Listening Master

As the name suggests, Listening Master[2] has to do with understanding spoken English. In this game with rather simple rules, the goal in each level is to transcribe a heard spoken sentence with correct spelling with as little errors as possible and within set time limit. The sentence can be repeatedly played back at any point. The measure of success in each level is determined by the swiftness of completion and number of grammar mistakes made. There are several options for transcribing the given phrase, either by just writing it, or by picking from scrambled letters, or just by picking the entire words in the correct order from a broader selection. This gives the option to more easily identify unfamiliar words if need be. There are four levels or difficulty distinguished by the length and complexity of the spoken phrase.

Positives

For bonus motivation, the game offers achievements unlockable upon progresss. There are also leaderboards based on a sum of scores of all completed levels within a difficulty

[1] https://en.duolingo.com
[2] https://play.google.com/store/apps/details?id=com.masterkeygames.listeningmaster



(a) The challenge

(b) Leaderboards by success in a difficulty level

(c) Achievements for progress

Figure 3.1: Listening Master — a simple game for practicing listening and spelling

group – each difficulty also has its own leaderboard. This may be beneficial since a beginner achieving good results in a beginner category can feel more positively motivated by their success. From a design standpoint, the application is very easy to use, only has a minimum of settings and controls and is free of distracting or purposeless features.

Possible negatives

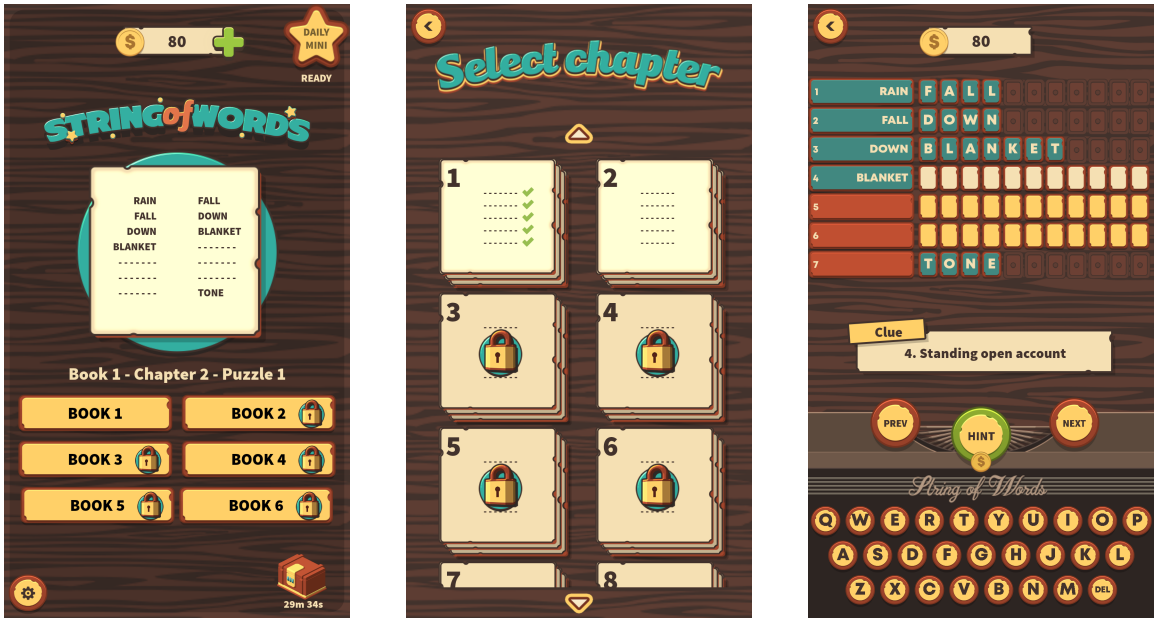
The game comes with a multiplayer mode achieved either by a friend match or by automatic matchmaking, however several dozens of attempts resulted in no matches, suggesting a user base too small for the automatic matchmaking feature to work properly, while matching with a friend also never succeded. Finally, a disputable aspect of this solution is the extreme simplicity and repetitiveness of the game, prompting an idea that it could be and example of a narrowly focused serious game, that heavily favors educational aspects above entertainment value [4].

3.2 String Of Words

The goal in String Of Words[3] is to create a chain of several one or two word expressions, where the first word of each expression is a second word (or a second half of one word) of the previous one. That way, each expression half-overlaps with the previous one and half-overlaps with the following one. At the beginning of each level the player is given the first half of the first expression and the second half of the last expression and has to connect them, which entails progressively figuring out all the expressions in between based on clues

[3] https://play.google.com/store/apps/details?id=com.nickelbuddy.stringofwords

to their meaning and expressions already solved. Figure 3.2c shows a level playthrough in progress.



(a) Main menu

(b) List of chapters (levels)

(c) Game - partially solved string of words

Figure 3.2: String of Words — practicing vocabulary on a set of predefined challenges

Positives

String Of Words balances the entertaining aspect of a variation on a crossword-like puzzle with the obvious need to practice vocabulary and know/learn the correct spelling. It also encourages broader linguistic thinking, since the clues provided are often vague and seldom close to exact definitions. Some expressions also require general outlook and various intercurricular knowledge.

Another positive aspects of the game is a rich, but uncluttered graphical design and a daily mini game feature that may help to keep players engaged over time.

Possible Negatives

Possible downside to String Of Words is the lack of any further sources of motivation other than the in-game progress itself, such as achievements etc. What the game also lacks are social features of any kind, that would add an extra competitive element and could be one of the options to make the game more dynamic.

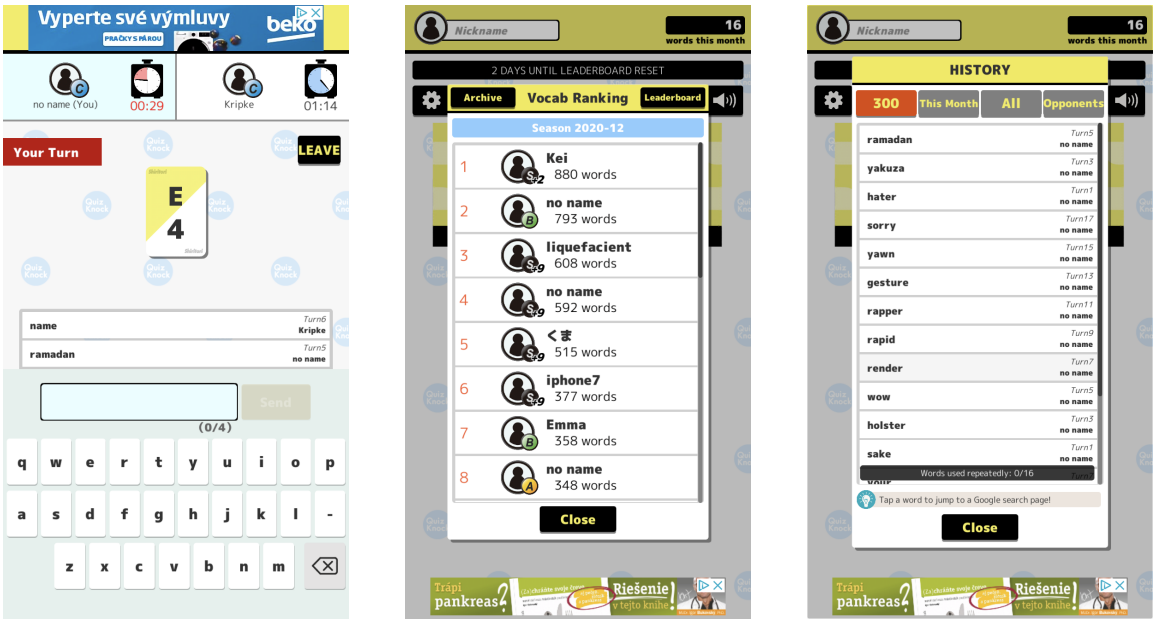
3.3 Shiritori

Shiritori[4] is a game of Japanese origin, similar in essence to English word chain. In this implementation each word needs to begin with the very final letter of the previous one and

[4] https://apps.apple.com/cz/app/shiritori-the-word-chain-game/id1490209869

also needs to be exactly of given length, which is generated randomly between 3 and 11 letters. No word can be used more than once in a single match. The player who creates more words worth more points is declared a winner and his winning score is added to a sum of all-time score. This score then determines the player's position in a global leaderboard, which together with a level system serves as a base for automatic matchmaking.

The overall approach manifested in Shiritori represents a shift from a serious game such as in 3.1, to "edutainment", that combines educational value with a more puproseful aim at entertainment [16], which is closer to the philosophy of this thesis' final product.



(a) One on one multiplayer (b) Global score leaderboard (c) History of used words

Figure 3.3: Shiritori — Originally Japanese game similar to word chain

Positives

The competitive multiplayer gameplay alone makes Shiritori feel more dynamic compared to previously discussed games. This is also aided by the above mentioned leaderboard being periodically reset, since the game is split into month-long competitive seasons.

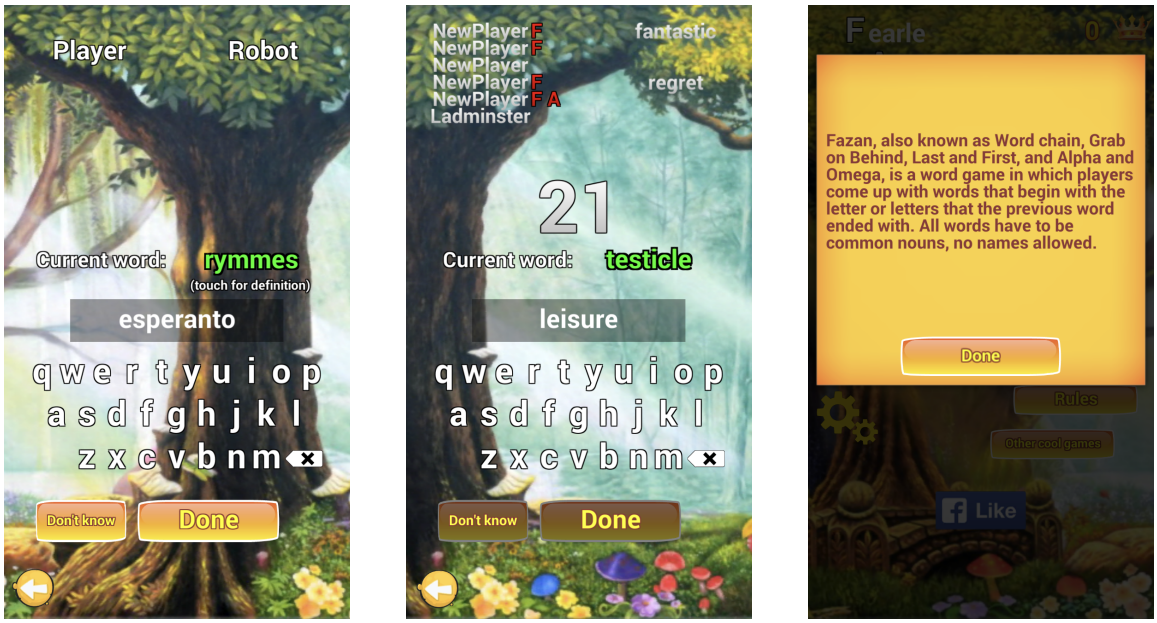
There are several positives from the education standpoint as well: the application features a list of words used by the player in past games, as well as a separate list of words used by their opponents, each of which redirects to Google should a definition or translation be needed. There is a friendly match option, which allows to adjust maximum and minimum length of words as well as time limit, encouraging further exploration.

Possible Negatives

Shiritori only has a few drawbacks, one of which is the user interface. While simple and relatively intuitive, not all of its interactive elements are visibly standing out from the design. What some players may also find discouraging is that there are close to no overall indications of permanent progress, since leaderboards and levels are being reset.

3.4 Fazan

Fazan[5] is a direct implementation of the word chain game as described in 2.3.2. It features a singleplayer mode against a robot and an online multiplayer mode achieved via game rooms players can join and where they have to create the next word within a time limit. If a player does not succeed, they get a strike. The last one with less than five strikes wins. The following word of the chain always begins with two letters from the previous one's end.



(a) Singleplayer mode (b) Multiplayer mode (c) Brief rules explanation

Figure 3.4: Fazan — a direct mobile implementation of the word chain game

Positives

The game has three difficulty settings, that can be unlocked upon a certain amount of online multiplayer victories, which can at first serve as an indicator of progress. Opponent's last played word is clickable for definition, which is provided by redirection to the Oxford English Dictionary. It is also worth noting that the multiplayer, in spite of a handful of debatable quirks, can be an engaging competitive experience.

Possible Negatives

Fazan suffers from a major issue regarding the mode for one player, where the automated opponent sometimes comes up with very obscure words, but upon asking for a definition is unable to provide them, which inevitably leads to the suspicion that some words may be made up. The robot can also come up with a word that ends in a combination of letters that the application itself doesn't have any words for, which sometimes ends the game seemingly at random. A minor downside from the educational standpoint is that the game only accepts nouns, which is a perfectly acceptable rule of gameplay, but greatly limits the range of vocabulary that can be practiced that way.

[5] https://play.google.com/store/apps/details?id=com.digitalapps.fazan

Chapter 4

.NET Ecosystem

The ".NET ecosystem" is a commonly used and understood, though unofficial, term for a set of defining standards, platforms, programming languages and developer tools for creating a broad range of (mainly application) software. The first full implementation of .NET platform, the .NET Framework, was launched in 2002 by Microsoft, along with the first version of C# programming language. Since then .NET has undergone significant development in all areas and in recent years eventually turned from a long-lasting Windows-centric approach to cross-platform support and open-source. Part of this newfound philosophy of .NET is maintaining fast evolution pace needed to keep up with technology and competition. There is also effort to put emphasis on clean and simple code, re-usability and continuous integration. Microsoft offers two native IDEs to be used with .NET: The complex and more monolithic Visual Studio that includes a plethora of development tools, and the lightweight Visual Studio Code, that handles most functionalities via custom-added extensions.

The rest of this chapter overviews .NET implementations and languages as of date.

4.1 .NET Implementations

Every .NET application is developed for one or multiple .NET implementations. Every such implementation consists of several components [31].

- **One or more .NET runtimes** – applications in .NET do not compile directly into hardware-specific binary code, but rather target a Common Intermediate Language (CLI) processed by said Common Language Runtime(s) (CLR).
- **A class library** – is an implementation of the shared library concept on .NET, which gives access to abstractions of essential or commonly used functionalities.
- **Optionally, one or more application frameworks** – that may give the implementation with extra suitability for implementing specific types of applications.
- **Optionally, development tools** – e. g. command line tools for package management or deployment/outside the box automation (NuGet, `dotnet` command).

There are curently four .NET implementations supported by Microsoft, each of which is characterized in this section. differences in class libraries, a formal specification of their exposed APIs called .NET Standard has been created to establish greater uniformity within

the ecosystem [33]. This standard has reached its final version in 2021 and increases compatibility between implementations as long as they're supported. For instance, it is needed to provide code for Xamarin app and .NET 5 app through a shared assembly.

4.1.1 .NET Framework

The original implementation of .NET is closed-source and only supports building and running Windows applications on and for Windows [32]. It had,been recognized as the main .NET development platform from 2002 to 2016, when .NET Core was introduced to compare with and eventually succeed it. The last and final version, .NET Framework 4.8, was released in 2019, with the implementation no longer being developed, but still getting long-term support. It comes with support for several high-level frameworks, such as ASP.NET for web application development or Windows Presentation Foundation (WPF) - a toolset for designing graphical user interfaces [32].

4.1.2 Universal Windows Platform

Universal Windows Platform (UWP) is an implementation released in 2015 and running exclusively on Windows 10 or 11. Both of these operating systems and their variations run on a multitude of hardware platforms, from traditional PCs, to tablets, touchscreen-enabled laptops or the Xbox gaming console. UWP aims at unifying development of client applications and providing maximum support and integration directly from the OS (availability on selected device types through Microsoft Store, Cortana voice assistant, ...) [23].

4.1.3 .NET (Core)

.NET, until version 3.1 known as .NET Core, is open-source and cross-platform. It allows creating applications for all major operating systems (Windows, Linux, MacOS, Android, iOS) and both Intel and ARM processor architectures. Version 1.0 was launched in 2016 to parallel .NET Framework and in 2021, with the release of .NET 5, took over as the main implementation going forward. .NET comes with reimplementations of its predecessor's frameworks (*ASP.NET Core*, *Windows Forms*, *WPF*, *Entity Framework Core*, ...) [31]. .NET lacks some of their functionalities or removes them as deprecated, but recently mainly adds new ones, with emphasis on keeping up with latest technologies and enforcing use of modern design patterns. .NET evolution also benefits from continuous integration of proven, sometimes community or third-party made API extensions, into the core implementation. It also provides additional APIs and frameworks, notably the *Xamarin* platform for mobile development, which, since .NET 6 is succeeded by MAUI (*Multi-platform APP UI*). MAUI allows creating native applications from a single codebase not only for Android and iOS, but also Windows or MacOS. .NET also provides cloud development options, such as for serverless architecture, especially in combination with Azure Cloud Services (see 5.2.2).

4.1.4 Mono

Launched in 2004, Mono is the oldest supported open-source .NET implementation. It is the choice for solutions that require a small, more lightweight runtime [31]. Historically, one of its main uses has been running applications written for .NET Framework on Unix systems, since Mono implements its APIs. Currently, it is the underlying runtime for Android and iOS (macOS, ...) native mobile applications written with Xamarin and MAUI [31].

4.2 .NET Programming Languages

.NET can be used with a wide selection of over 40 programming languages, only a handful of which however are developed or natively supported by Microsoft. This number is uncommonly large, because thanks to compilation through CIL and CLR (see section 4.1) .NET implementations are *language agnostic* [34]. In consequence, they can be used with any programming language as long as it adheres to **Common Type Specification** (CTS). CTS defines semantic rules, usage and properties for types in these fundamental categories: classes, structures, enums, interfaces and delegates. On .NET, different languages conforming to CTS can even be used together, although for maximum interoperability they must also fit into the same **Common Language Specification** (CLS). CLS is a subset of CTS, that defines further requirements on what features and scenarios languages have to support [34].

The following subsection summarizes the most prominent and most commonly used .NET language – C#. Other examples of .NET languages are F#, Visual Basic, C++/CLI, IronPython, IronRuby, Haskell, PHP, ADA, COBOL or FORTRAN.

4.2.1 C#

C# is a multi-paradigm object-oriented, type-safe language inspired by Java, with syntax similar to C/C++ It features all the features commonly expected from a high-level language, such as garbage collection for memory management, exception handling for error detection or built-in support for asynchronous operations to be used when coding distributed systems. On top of that it has Lambda expressions that introduce functional programming to the code or the Lanquage integrated query (LINQ) – a common abstraction for working with data from different types of sources.

C# code is organized into *programs*, that declare *types* organized into *namespaces* and consisting of *members* [26].

C# types are either reference types (holding references to data) or value types (holding the actual data) and derive from a base type `object`, that provides a set of common operations. Apart from simple value types like `int` or `double`, it also supports creation of user-defined `enum`, `struct` or `tuple` types. When it comes to reference types, on top of strings, arrays and classes, common in one way or another to many modern object-oriented languages, C# also offers `interface`, that only allows to specify a contract (declarations of exposed members of a type), to be implemented by a custom `class` or `struct`. While a C# class cannot inherit implemented members from more than one class, it can implement contracts of several interfaces. `interface` allows replacing internal logic of a class without changing any dependant code (loose-coupling), thus making it an example of how C# supports component-oriented programming. Another example is the `delegate` type that is used to reference class methods in a type-safe way (unlike for instance traditional function pointers). C# also allows creating generic implementation templates that concrete types can be based on [26].

There is a regular release schedule of C# versions currently in place (every 1 to 2) years.

New C# versions have historically been released every 1 to 2 years, with every release extending available features and simplifying syntax of existing language constructs in order to keep up with emerging design and programming trends. The most recent version as of date is C# 10, with C# 11 release scheduled for the end of 2022.

Chapter 5

Cloud Computing Services

The proposed mobile game will come with an online multiplayer, which needs to be reflected in its architecture. In such situations, existing solutions often make use of a application backend service deployed on a cloud [28]. This chapter describes general principles of cloud computing and its benefits over different approaches, such as a dedicated server architecture. a brief comparison of some of the most widely used cloud services is given.

5.1 Principles and Assessment

The definition of "cloud computing" has been worded in many different ways. For the purpose of this text a formulation was selected that defines cloud computing as *"a type of Internet-based computing that provides users with multiple scalable on-demand services, data, or products through sharing or accessing various computing resources. Those resources can come from private systems or from third-party data centers, and can be accessed locally or remotely from city-wide to world-wide. Using cloud means users can elastically obtain hardware, software, or other computing resources on demand."* [22] Apart from this definition, cloud computing is also commonly distinguished by five essential characteristics [13]:

- **On-demand self-service** – The system automatically provisions compute options selected by the consumer, without any human intervention.
- **Broad network access** – The services provisioned are available from a broad variety of client platforms, both more traditional and mobile.
- **Resource pooling** – Infrastracture is multi-tenanted and its location has no impact on service delivery or available features.
- **Rapid elasticity** – Infrastracture can change volume on demand without consumer impact.
- **Measured service** – Resource usage is monitored, controlled, and reported in order for complete pricing transparency for both the users and the service provider.

From these characteristics two benefits of cloud computing can already be drawn: First, if the services provided fit the customer's needs, they can focus almost exclusively on the main task they aim to accomplish (more on service models in 5.1.1). For the purposes of this thesis' project that is an important quality. Another meaningful benefit comes from the

characteristics of rapid elasticity, which allows to always only occupy the needed amount of resources, and presence of usage metrics. These together calculate prices to the actual amount of service provided, which often results in lower operation costs [28, 13].

5.1.1 Cloud Computing Service Models

There are many different service models in practice today, each of them offering different resources and options. The three easiest to categorize and most commonly distinguished ones are *Infrastracture-as-a-Service (IaaS)*, *Platform-as-a-Service (PaaS)* and *Software-as-a-Service (SaaS)* [22].

Infrastracture-as-a-Service model enables users to get virtualized computing resources (hard drives, processors, memory cards, ...) from the cloud provider. As such, it is the foundation of cloud computing. With IaaS, customers can focus on operating systems, applications, etc. while the service providers maintain servers, networks, processing and storage [22].

Atop of IaaS environment there can be seated a *Platform-as-a-service*, which offers an extensible high-level programmable platform composing of subsystems that meet requirements for application development and hosting operations (interface or database development, data storage, system testing, ...) [13, 22]. That allows customers to easily build customized applications largely on their own terms, without the hassle and expenses over dedicated infrastracture and platform maintenance.[13]. An important feature of PaaS is providing external developers with an open Web-based development platform. Apart from that it should also promote security and reliability of platforms.[22].

Finally, *Software-as-a-Service*, also reffered to as *Application-as-a-service* is "leveraging cloud-based software solutions to provide users with web-based software, platform and infrastructure services" [22]. Users then usually access these applications via a web browser, without installing any of the software locally. This model is specific in that, unlike the previous ones, it usually connects directly to end consumers.

5.2 Public Cloud Computing Services

Public cloud computing is one that is provided to a theoretically unlimited number of customers by a third party [13].

For the purpose of this thesis we are only looking at clouds offering PaaS services, as the thesis is focused on the design and implementation of application software. What follows are the three leading most widely used public clouds. All of them bind themselves to provide over 99 % availability on all of their major services in their service level agreement, and are honoring it[1]. The below short summary is based on documentation and two comparative articles. [30, 10]

5.2.1 Amazon Web Services

To date, Amazon Web Services (AWS) is the biggest provider of cloud services, especially because of the superiority of its infrastracture developed over the years. It is considered especially well-suited for computing needs of larger organizations, though it's in no way restrictive. It's main compute service, called Elastic Compute Cloud, supports Windows and Linux operating systems and allows for automatic scaling of used resources. It also

[1]Based on independent analytics by CloudHarmony – https://cloudharmony.com/cloudsquare

has many container options for running Web applications, both own and supported ones, such as Docker or Kubernetes. It also has a variety of storage options both for storing and transferring files (Simple Storage Service, Elastic File System, Snowball...) and both relational databases, such as PostgreSQL-compatible Amazon Aurora, or non-relational options like the key-value based DynamoDB. The data can be backed up and later retrieved using the Glacier backup service. On top of these yet more traditional services of more importance to this thesis, AWS, as its competitors, also offers tools for creating serverless architectures, internet of things (IoT) projects and artificial intelligence.

With all its possibilities however, AWS is deemed to be more challenging to be put to advantage fast, especially for small groups or solitary developers also because of the overwhelming variety and complexity of services and pricing options.

5.2.2 Microsoft Azure

This second biggest provider, though not by a narrow margin, offers in essence similar range of services. Despite coming into the field a little later, it had the advantage of having Microsoft's data center infrastructure at its disposal, as well as existing software solutions that could be repurposed for the cloud. Its primary cloud compute service is called Virtual Machines and has support for Linux, Windows Server, Oracle or IBM and more. Similar to Amazon it offers automatic scaling, via Virtual Machine Scale Sets. This is complemented by two container services based on Kubernetes and Docker Hub. Azure also offers an option specifically aimed at applications with microservices architecture, called Service Fabric. It offers both SQL and NoSQL databases, but as of date focuses somewhat more on relational database options. Azure also features options for serverless architectures. Microsoft's take on serverless is spearheaded by a technology called Azure Functions, which allow building serverless solutions from separately instantiated and executed logical units. It further offers solutions for IoT, artificial intelligence and machine learning. Its main platform for web applications is called Azure App Service.

Administration of services can be performed via Azure Portal website, that offers easy access to a plethora of configuration, monitoring and diagnostic tools.

Though Azure's portfolio may be narrower, it can be used with advantage for the purpose of this project: it is closely integrated with a lot of Microsoft's software, including a range of developer tools. That especially manifests itself with applications running .NET code.

5.2.3 Google Cloud Platform

The smallest of the three leading clouds has a smaller selection of specific technologies, but overall offers competitive services, it's primary compute service, the Google Compute Engine, provides the same functions as Elastic Compute Cloud or Virtual Machines. With containers, Google's advantage is that it's the developer behind Kubernetes, which both of the previous clouds make use of. Storage-wise it provides just enough options of storing structured and non-structured data to cover all the basics, it is however shy on data backup. It's main domain of excellence is AI and machine learning. Its Google App Engine for web applications offers similar options to both previous solutions.

The main advantage from the point of view of the project at hand is that while based on the overview it provides sufficient services at some of the lowest prices from the very beginning, including a free subscription tier of unlimitied length.

Chapter 6

Pre-Implementation Phase

Previously presented research of existing language games and development options allowed proposing a concrete vision for the game and its development. After defining a target group and specifying application features and concepts, game rules and solution architecture were drawn out. Finally, a set of development tools and deployment platforms was selected.

6.1 Target Audience

This game is not targeted at any narrow demographic, but it could likely only appeal to those foreign and native English speakers and learners, who already have at least basic vocabulary knowledge already pre-established, at the very least on a A1 level according to the Common European Framework of Reference for Languages (as used by the system of Cambridge English Qualifications) [8]. It is also likely that from educational perspective it will be most useful to learners in range of Independent language users [8] or higher. The average age among native speakers may be lower than among foreign English learners.

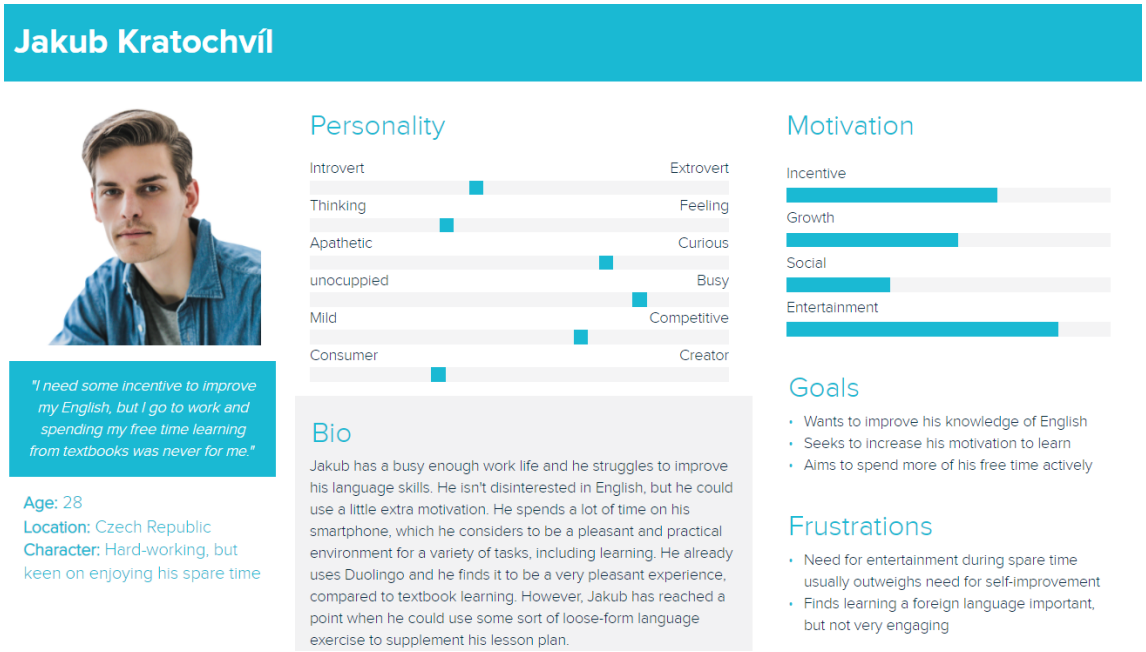


Figure 6.1: Primary user persona

6.2 Application Concepts

This section lays out an overview of the proposed application's features and formal conceptual use-case and data models and ends with user interface conceptualization. Concept of the game itself can be found in section 6.3.

6.2.1 Application Features and Elements

In alignment with the pre-set goals of the thesis, the game application's core feature is going to be an online competitive multiplayer game for two players. To better facilitate the educational aspect, the application is furthermore going to be equipped with a word definition feature and player's history of words encountered in matches. In case players use words that are not found in a selected dictionary, they should be able to suggest such word for addition. Based on these main functionalities it was decided these elements should also be included:

- **Progress** - aggregate game score, personal level system, aggregate personal PvP match statistics
- **Social** - a leaderboard and a simple user profile to enable users compare themselves to others

Application use cases emerging from features mentions are illustrated in figure 6.2.

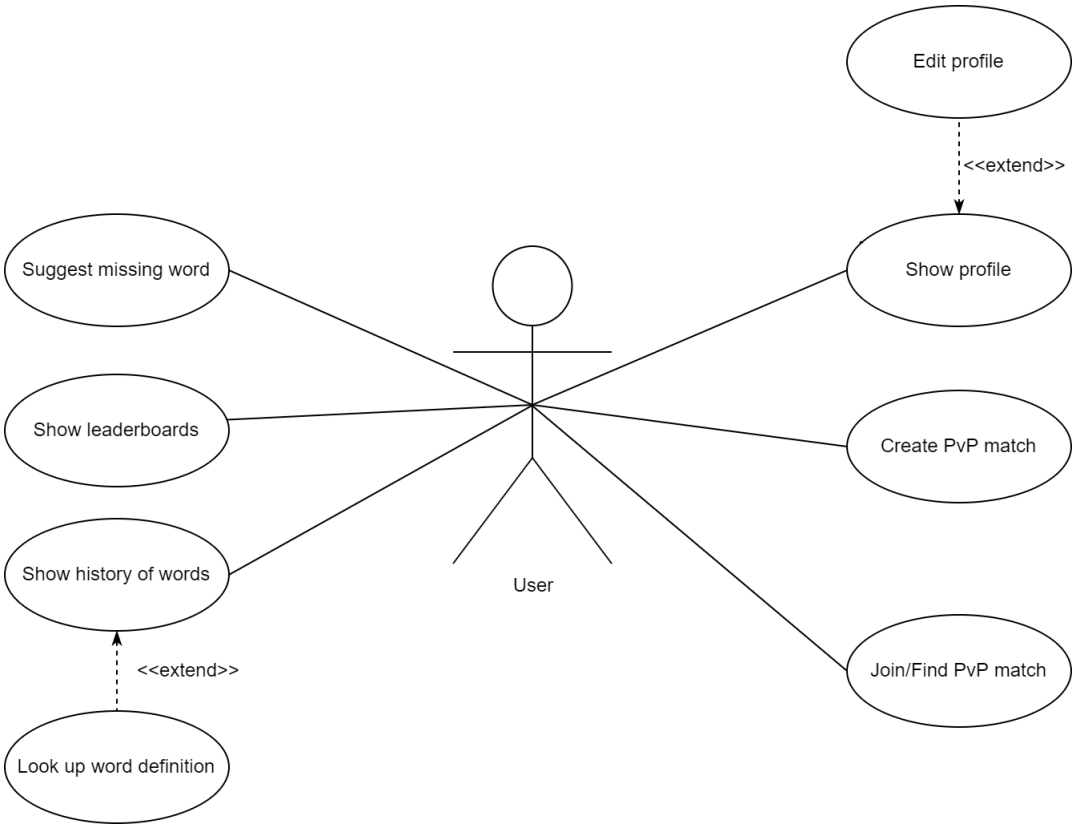


Figure 6.2: Use-case diagram model of the application

6.2.2 Persistent Data Model

The persistent data necessary for supporting functionalities and elements listed in subsection 6.2.1 is structured into a relational data model. An appropriate concept model that translates well into relational data is the entity relationship model.

Diagram in figure 6.3 depicts a representation of this application's data. The central focus here is on the *player* who plays (and sometimes wins) *PvP matches*. During rounds of each match the player plays or encounters *played words*. At the end of each match, they receive word score for words they played. Every time they have enough word score, they reach a new *level*. They also keep track of *words suggested* by them in cases when arguably real words were not found in a dictionary and rejected by the game. Players can also set, if their statistics are going to be visible to other players and they can set their username and profile image.

This model focuses on data specific to business logic and does not model additional data required for authentication and authorization. More on that in chapter 7.

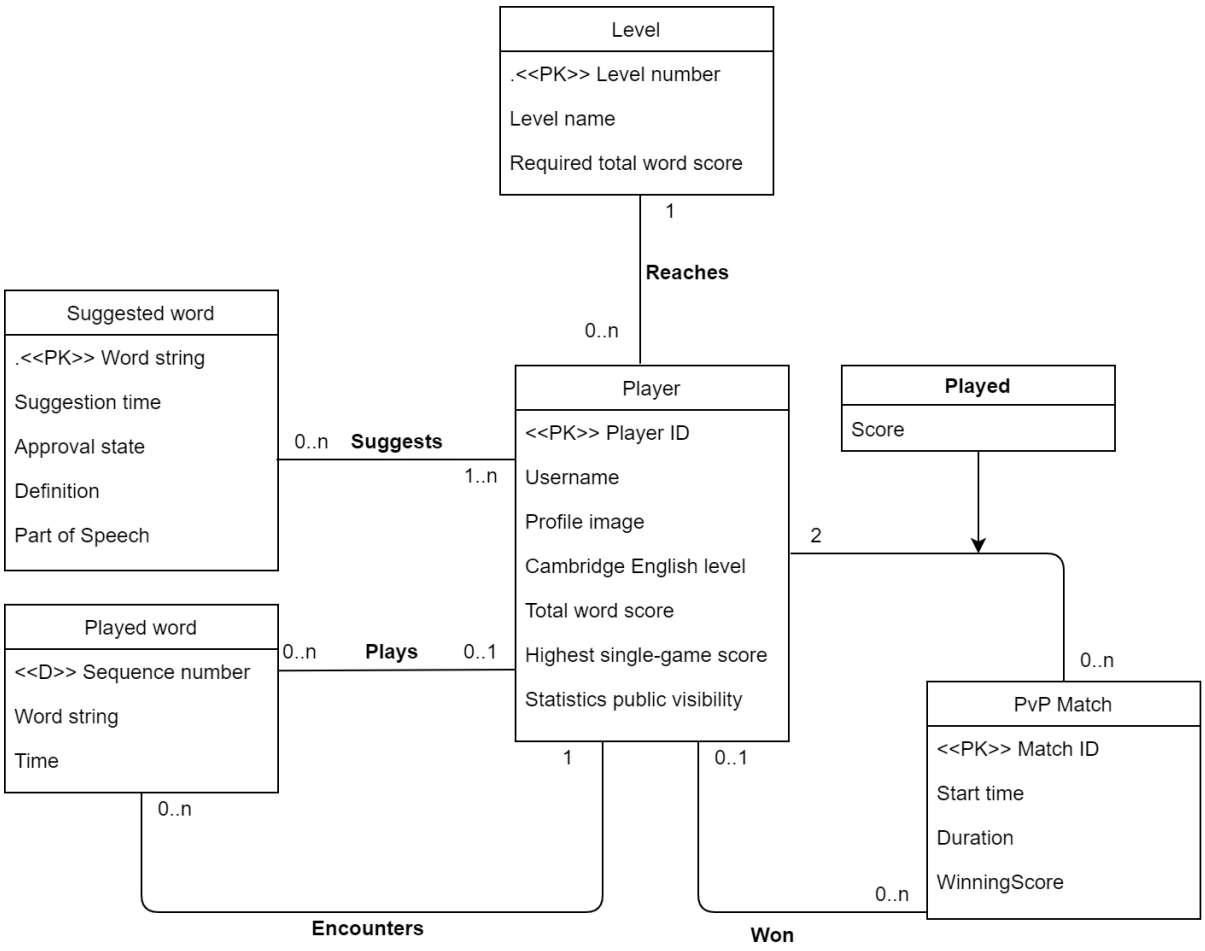


Figure 6.3: Persistent data — Entity relationship diagram

6.3 Game Concepts

Rules of word chain had to be modified to better utilize possibilities provided by the digital platform. Subsequently, flow of the game was visualised as a diagram.

6.3.1 Word Chain Rules Overhaul

Rules of the game to be implemented are going to directly derive from the basic rule set of word chain (as described in subsection 2.3.2), with several variations and additions that aim to make the game more dynamic and make use of gameplay mechanics' automation common in digital games.

Permitted Vocabulary

Vocabulary is restricted to the main parts of speech that bear meaning: nouns, adjectives, verbs, and adverbs. Verbs must be used in infinitive form only. If a singular form of a noun exists, it must not be used in plural form. Personal nouns should be excluded, as long as they don't have another meaning (e. g. "john" – toilet, "roger" – hear and understand, "frank" – honest).

This is a compromise between limiting vocabulary that can be practiced in this way to nouns and allowing for all words, including articles or pronouns.

Additional Rules

- There is a time limit for following up with a word.
- The game starts with a randomly selected player coming up with a word starting with a randomly generated prefix.
- The time limit for the next round (a single exchange of words) becomes shorter after each round where both players successfully landed a word.
- Player gains word score after each successful turn (each word they play).
- When the chain is broken, the game ends, the winner is determined as whoever has higher word score and score for the match is added to the word score total of both players.

Optional Parameters

The player creating a game will be able to change default length of the time limit, and optionally set the number of letters overlapping from previous to next word to two instead of one.

6.3.2 Game Flow Overview

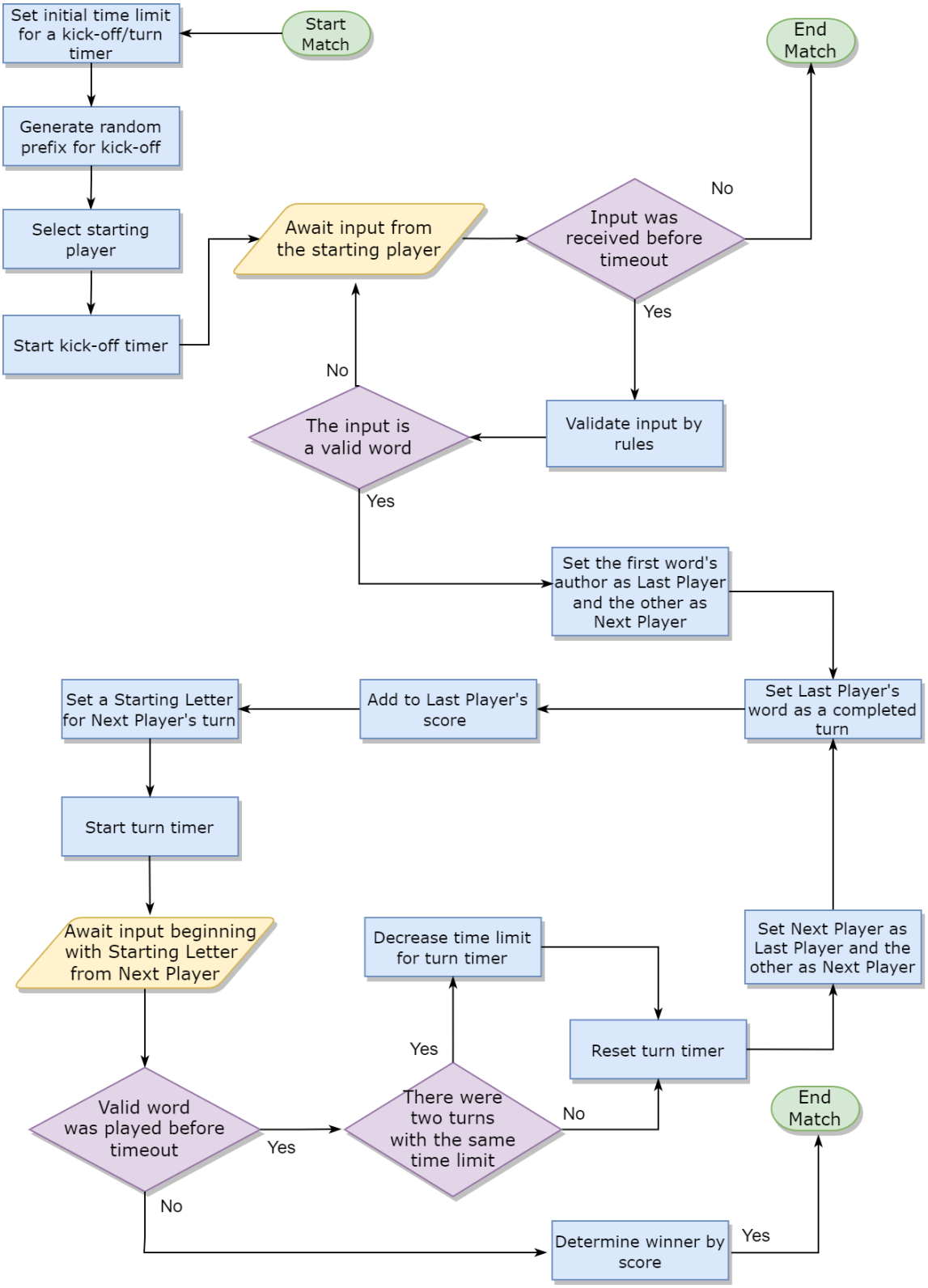


Figure 6.4: Flow diagram of a Word Chain game match.

6.4 Application Architecture Overview

As shown in a diagram in figure 6.5, the application is split into several interconnected web services that are called from a mobile client. Compared to a solution with a single monolithic back-end service, this requires more integration of these respective parts, but allows for more dynamic and modular development. For example the need to replace an implementation of one or more services does not affect the other deployed units as long as application programming interfaces (APIs) remain unaltered.

The following subsections summarize responsibilities and design of each individual unit. Implementation details and further specifics are omitted and can be found in chapter 7.

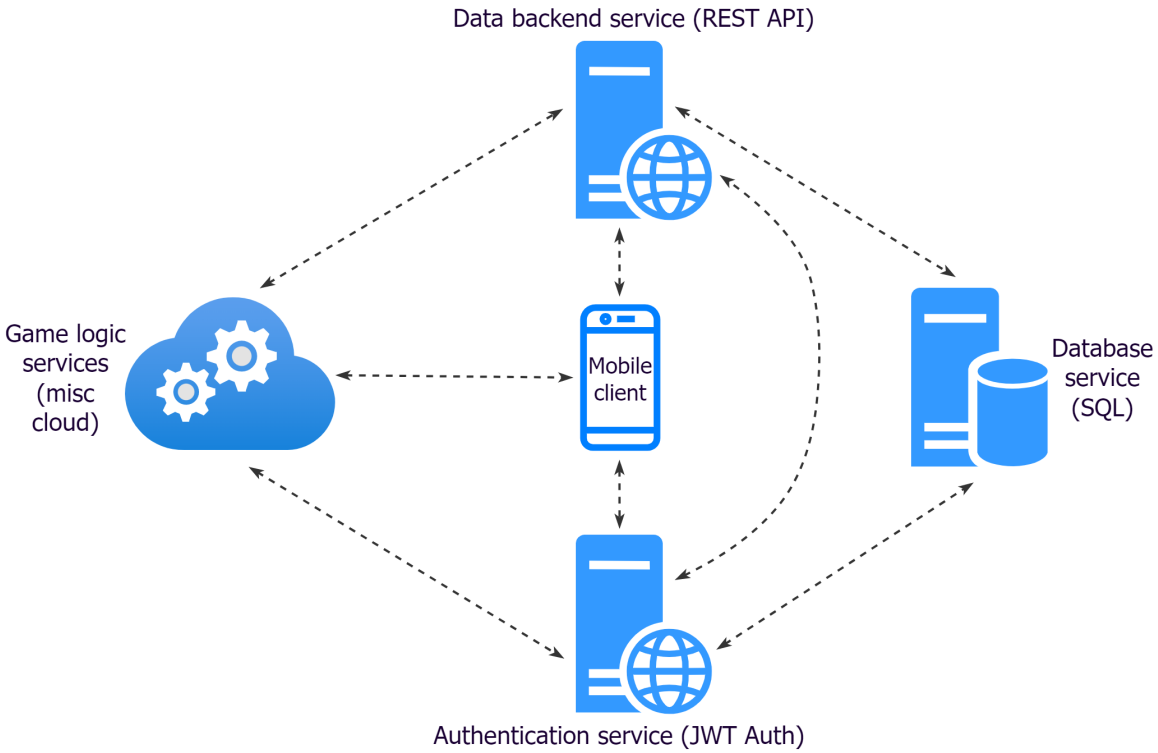


Figure 6.5: Word chain web application architecture overview

6.4.1 Data Backend Service

This part of application backend serves persistent data to all other units. It provides object mapping above the relational database behind it and a layer of abstraction through use of shared data models. The contents of these models are adjusted to mainly contain data required by the mobile client. Data access is enabled through a RESTful API consisting of CRUD (Create, Read, Update, Delete) operations above the models. This service performs all operations above the connected database service, apart from authentication/authorization related queries, and so all functionality of game logic and mobile client are heavily dependant on it.

6.4.2 Authentication and Authorization Service

The authentication and authorization service is a separate unit, since this approach allows for simpler adding and removing of authentication and authorization requiring services to the architecture. It is designed as an identity server that implements elements necessary for JSON Web Token authentication, which is the means of user auth across the entire solution. Following a microservice architecture, it has its own database schema for storing user credentials, valid tokens and other necessary data.

6.4.3 PvP Game Cloud Services

In contrast to the other services, the game itself is conceived less traditionally: not as a single monolithic service, but rather as a collection of smaller cloud services with more specific responsibilities - the so-called "serverless" architecture. This maximizes modularity and individual deployment of units during development. Considering this part of the solution is expected to be the most utilized, PvP Game services should benefit from advantages of cloud outlined in chapter 5.2 in hypothetical moments of high traffic. Game logic roughly consists of service for keeping a smooth (ideally near real-time) connection with the mobile client, a loosely connected collection of endpoints that implement the main automatic gameplay mechanisms and internal part of game flow, and a lightweight storage service for short-term persisting of essential session data during a match. Due to the aforementioned high level of modularity, game services are the most architecturally complex part of the backend and are elaborated on further in chapter 7.

A web-based interface of PvP Game services is exposed for the mobile client and consumes the Data Backend API, which is used for updating player and match data after each completed game.

6.4.4 Mobile Client

The application's front-end is a native mobile client that contains business logic operating above all three web services in the back-end. Its responsibilites include transformations above received and sent data and interactive graphical presentation of business logic results. It only stores minimum of data locally and its functionalities are entirely dependant on internet connection. It implements pieces of PvP game logic that can be executed prior to entering the remote part of game flow. It requires users to authenticate in order to allow access to use-case scenarios.

6.5 Selected Technologies

When selecting technologies for implementation and deployment, it was sensible for a .NET based project to stick with options provided by Microsoft wherever possible, as these options were expected to have more direct support than third-party alternatives.

6.5.1 Development Tools and Languages

The entire application is implemented in the bounds of tools and frameworks that are part of the .NET ecosystem. All projects target open-source .NET implementations .NET 6 and .NET Standard 2.1 (see chapter 4).

The main programming language throughout the entire implementation is C#. The mobile client also uses XAML (eXtensible Application Markup Language) for defining its graphical user interface declaratively [7].

The application solution is mainly written using NET's own development environment Visual Studio 2019/2022. Source files are stored in a git repository, remote copy of which was at time of development hosted on GitLab[1].

6.5.2 Implementation and Deployment Technologies

As already specified is this thesis' assignment, the mobile client is implemented on the Xamarin platform. Xamarin offers progressively unified abstraction framework above platform-specific interface elements on Android, iOS and WUP (Windows Universal Platform) called Xamarin.Forms. In combination with Xamarin.Essentials API, which provide further cross-platform abstractions for accessing common mobile device resources and tools on these operating systems, it is possible to maximize code reusability and build applications for both Android an iOS from a single shared codebase [14, 15].

The Data Backend service and the Auth service are based around the web-focused ASP.NET Core framework, that simplifies writing web APIs with .NET. Above convenience alone, ASP.NET Core services also perform very well in performance benchmarks compared to other options – as of January 13th 2022 – rank 7th overall according to composite scores calculated from tests independently conducted by *TechEmpower*[2].

Serverless PvP game logic is designed and implemented as a collection of interconnected standalone cloud endpoints (Azure Functions), exposed as HTTP or real-time supporting SignalR protocol endpoints, and supporting services. These can be implemented directly on Azure and incrementally integrated, however, local development was chosen instead to allow for more compact project structure. This option uses local Azurite service, necessary to emulate underlying architecture of actual Azure environment, and *Azure SignalR Service* emulator to serve as a front for serverless real-time communication.

Application back-end is partially deployed to Azure in shape of fitting Azure Services. The main incentive for selecting this option were the broad integration options with other Microsoft application software and .NET tools, as mentioned in 5.2.2.

Relational Database will be deployed as *Azure SQL* database service, which is based on Microsoft SQL. a recommended low-maintenance choice for ASP.NET Core deployment is the *Azure App Service*.

PvP game service implementation is by default platform-specific to Azure Cloud and the services it offers, thus is the only deployment option.

6.5.3 Predetermined limitations

The mobile application will be developed on Windows 10 operating system, which brings forth a limitation regarding building and testing the application for iOS, as that requires either an Apple mobile device and a developer account or some form of access to a Mac device to test the solution in an emulator [19]. For the purpose of this thesis, the issue will by default be ignored and the application will be primarily built and tested for Android, using Android Emulator provided by Xamarin.Android SDK and eventually on a low-mid tier Android smartphone.

[1] https://about.gitlab.com
[2] https://www.techempower.com/benchmarks/#hw=cl&test=composite

Chapter 7

Implementation

This chapter goes over architectural, in-code and other implementation details of the final software's individual parts (see section 6.4 for top-level architecture overview). While the development process is described in linear cascade-like fashion, in reality the concept and implementation phase were both iterative, with design requirements changing based on technical experience gained during development process and vice versa. The four main parts were being developed in parallel in a local environment and gradually expanded and integrated.

Development was carried out in C# and XAML programming languages using tools and technologies outlined in section 6.5, with additional frameworks and libraries, the most important of which are highlighted through the rest of the chapter.

7.1 Solution structure

Final implementation is structured as a single .NET Visual Studio Solution consisting of a total of 12 .NET projects: 8 C# .NET class libraries and **4 executable application projects**. The application projects belong to individual parts of the architecture as follows:

- `IdentityServer` – authentication service application,
- `WordChain.Data.Api` – data backend service application,
- `WordChain.Game.Service` – serverless game service application,
- `WordChain.Mobile.Android` – WordChain native Android application.

In the following sections of the chapter, individual projects are matched with each application's architecture overview.

7.2 Authentication Service

This solution's self-managed identity provider is used as an authentication and authorization service in both remaining back-end services as well as in the mobile client. It implements OAuth 2.0 protocol with OpenID Connect authentication and allows clients to authorize with services using JSON Web Token bearer authentication scheme.

7.2.1 IdentityServer4

Given the broadness of the topic of web security and authentication and complexity of the OAuth and OpenId specification, neither of which are the focus of this thesis, and following an advice to use a templated solution if possible, this project's identity server was created using *IdentityServer4*[1] framework for ASP.NET Core and following official documentation [3].

This solution makes use of *IdentityServer4* Quickstart UI[2] template created by Brock Allen. Quickstart UI provides Model-View-Controller web application for IdentityServer, that include a login and logout functionalities.

Tools provided by *IdentityServer4* are integrated with *Asp.NET Core Identity* library that provides useful data models for persisting identity users, claims, scopes, clients, configurations and grants into a relational database.

7.2.2 Authentication Details

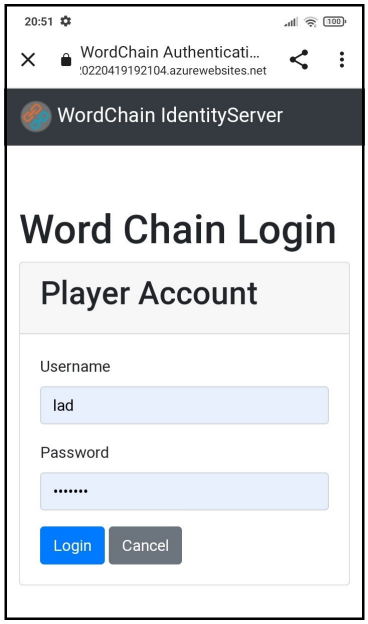


Figure 7.1: WordChain identity server login page

Front-channel authentication (live user login from the WordChain application) retrieves tokens via Authorization Code Flow with PKCE. In consequence, the login operation that authorizes the user to access the native application, takes place directly on the Identity Server login page. User is then considered to be authenticated to use the native application, if this flow succeeds and the native application receives back appropriate access tokens and JWT credentials. Figure 7.1 shows the login page to which the WordChain native application redirects upon Sign In.

Back-channel authentication (machine to machine – PvP Game service to IdentityServer) retrieves tokens via Client Credentials flow, in which the authenticating client and the Identity Server identify each other over a shared secret.

[1] https://github.com/IdentityServer/IdentityServer4
[2] https://github.com/IdentityServer/IdentityServer4.Quickstart.UI

Current Limitation

For the purpose of this thesis, only login and logout flows are implemented. Registration of new users is available neither in the WordChain application nor on the IdentityServer login page. Instead, several test users were seeded into the database beforehand.

Despite the original solution architecture proposal, IdentityServer is not used to secure locally PvP Game services, because no supported option was found for securing locally developed Azure Function project endpoints with a custom openID provider (more on PvP game service limitations in subsection 7.4.4).

7.3 Data Backend Service

Essential persistent data is made available to other parts of the architecture via a RESTful ASP.NET Core API application built on top of an SQL Server database. It is used by both the mobile client and the serverless game service. Access to the API is secured by the Authentication service (see section 7.2).

7.3.1 Architecture overview

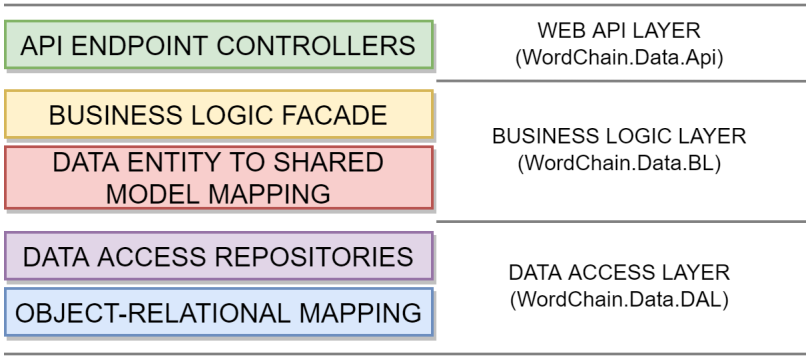


Figure 7.2: Layered architecture of WordChain data web API backend

Data backend service implements common layered architecture as shown in figure 7.2, which splits the application into **data access layer**, **business logic layer** and **web API layer**.

Data Access Layer

The data access layer (DAL) is the foundation of layered architecture and works directly above SQL Server database schema. On the lowest level it performs object-relational mapping from database tables to object entities and vice versa. As a result of this abstraction, the immediately following repository classes do not directly rely on SQL queries for reading and writing operations. Each entity that is directly accessed by the upper business logic layer has its own repository.

Business Logic Layer

In business logic layer (BLL), repositories are covered by facade classes for further abstraction. Facade classes implement pieces of business logic when loading or saving entities

(for example when a player finishes a match, calculate his aggregate score and check if they leveled up). As entities with relationships between them is not an adequate data representation for a CRUD application interface, a facade accepts and returns data to the top layer only in after mapping it to various API models (e. g. `PlayerEntity` maps to `PlayerDetailModel` and `PlayerListModel`).

API Layer

In the uppermost web API layer, there are endpoints grouped into controllers as per standard ASP.NET Core API project. The division of controllers is based on what facade a controller uses (e. g. `PlayerController` uses a `PlayerFacade`). When invoked by an authorized caller, an endpoint calls its facade to perform a CRUD operation. It contains no additional logic apart from calling the facade and returning HTTP responses.

On Create, Read or Update, the data an endpoint can consume or pass on has to be a model that BLL can operate above and map to an entity and has to be a part of API specification.

API models are defined in `WordChain.Data.Models` class library project.

7.3.2 Implementation Details

WordChain.Data.DAL

Object-relational mapping between database and data repositories (see subsection 7.3.1) is done by the *Entity Framework Core* library[3]. *Entity Framework Core* tools were also used to create an SQL Server database schema code-first – by applying migrations generated from a defined set of object entities. Since data entities were already defined as a part of concept design (see subsection 6.2.2) it is the most favourable approach.

WordChain.Data.Api

On startup, `WordChain.Data.Api` project registers *NSwag*[4] middlewares for generating both human-and-machine-readable OpenApi [27] specification and further use it to create a *SwaggerUI* documentation landing page for the API url (see figure 7.3). The documentation is interactive and presents a way of sending mock requests to the API.

The specification was also used in *NSwagStudio*[5] to generate client-side C# code for data consuming applications.

7.4 PvP Game Service

The online portion of the game of WordChain was conceptualized and implemented as an Azure Function application for Azure Cloud. As such, it consists of a collection of separately instantiated and executed, automatically scaled Azure Function endpoints within Azure cloud environment and additional cloud services, these functions operate above or are integrated with.

Development process of a serverless application in local environment has been substantially different and more challenging compared to the rest of this solution and some

[3] https://docs.microsoft.com/en-us/ef/core/
[4] https://github.com/RicoSuter/NSwag
[5] https://github.com/RicoSuter/NSwag/wiki/NSwagStudio

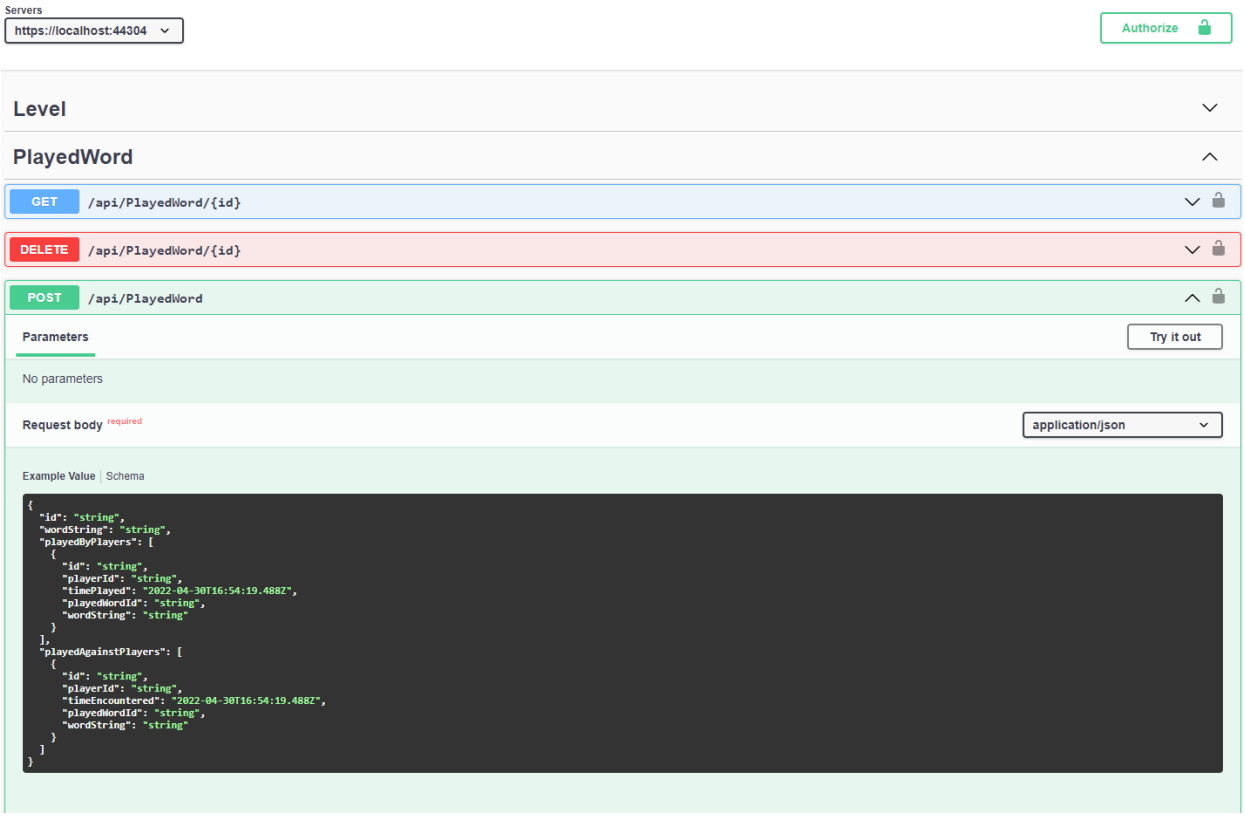


Figure 7.3: WordChain.Data.Api interactive *Swagger UI* with api specification

key issues haven't been fully solved. Current state of implementation is summarized in subsection 7.4.4.

7.4.1 Game API

In order for the game to work properly, there has to be support for real-time bi-directional client-server communication, that does not exist in HTTP. For this purpose, *SignalR* protocol was selected. It supports pushing messages in both upstream and downstream directions and can be easily implemented in cloud as well as in a mobile application using Microsoft-supported libraries.

The PvP game service-client application interface reflects this choice. In the Visual Studio solution, it is defined in a shared `WordChain.Game.Common` class library containing a set of HTTP and *SignalR* service endpoint and client callback identifiers, interfaces to be implemented by both sides, and *SignalR* **argument models**.

Argument Models

Since C# *SignalR* implementation accepts variable number of arguments as contents of a *SignalR* message and transfers it as the basic object type, there is no way to check if a *SignalR* endpoint accepts message contents, before the message is received.

To mitigate this issue, argument models formalize specific message content types, that should in turn only be associated with messages to their designated *SignalR* endpoints and callbacks.

7.4.2 Architecture overview

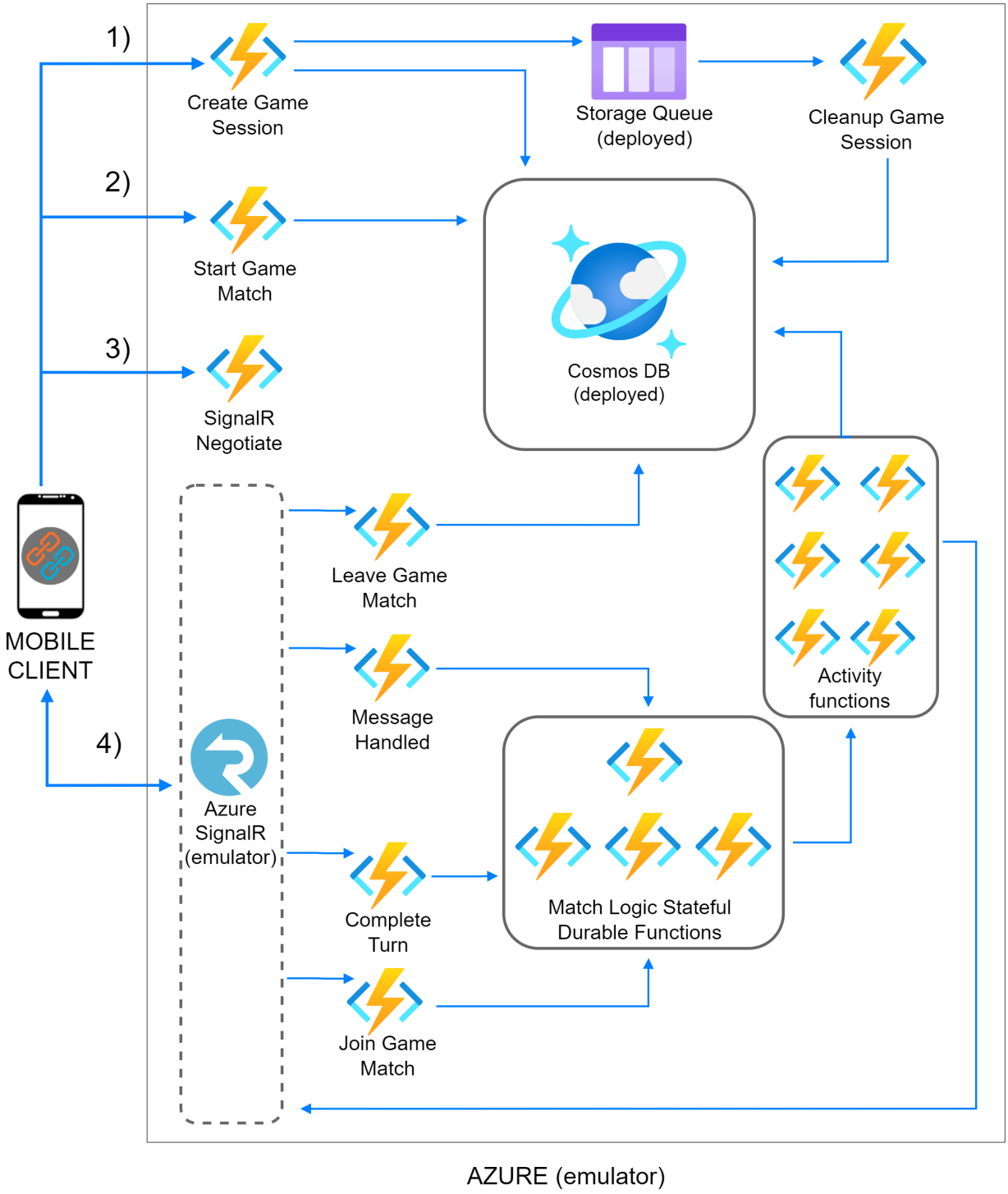


Figure 7.4: Serverless PvP game service overview

Simplified diagram in figure 7.4 depicts what components are used and how they interoperate to form a web service interface for the WordChain mobile client and execute core WordChain game flow logic. Service-side implementation originally contained all game-

related functionalities, but some of them (most importantly turn input validation and persisting played words through data backend) were eventually moved to the mobile application.

The rest of this subsection describes individual architecture components and their purpose.

Integrated Services

As a support for Function logic, the implementation integrates following Azure services:

- *Cosmos DB* – Azure document storage used to persist information about matches into session data (player IDs, unique session code, match instance settings . . .).
- *Storage Queue* – a message queue that any number of Functions can push to and receive from. Diagram 7.4 shows a storage queue used for scheduling newly persisted match session data to be periodically checked and removed if they are unused.
- *Azure SignalR Service* – service fronting a collection of *SignalR* triggered Azure Functions in order to form a real-time API (see subsection 7.4.3).

Service endpoints

Server-side interface available to the client consists of Azure Functions that execute upon being triggered by a HTTP request or a *SignalR* message. Client uses them to:

1. Optionally create a new persistent session to be used within a match instance, or retrieve an existing one by its unique code,
2. Start or join this match instance by joining the session and ensuring an appropriate instance of match flow exists,
3. Establish real-time bi-directional *SignalR* connection to be used for communication during said match instance,
4. Finally join a *SignalR* connection group that contains all players within the instance. The client is now fully connected to the match instance and game flow can begin.

Durable functions

Durable Function (or Durable orchestration) is a specific type of a Function, that allows implementing a stateful workflow. While a typical Function is stateless and it's instance is not persisted after executing, Durable Functions can be instantiated on demand and are tracked by the Function Application even after completing. On top of that, Durable Function instances have an identifier, that can be set to a custom *string* value. They can pause their execution to wait for events. They can call **Activity functions** to perform steps of their logic.

In PvP game service, every match instance is as instance of a Durable `PvPMatchWorkflow` function, identified by an ID of its related persisted session. It executes in three steps, each step being an inner Durable Function (a sub-orchestration): **Match initialization** lasting until players have joined, **Game execution** performing main game logic, and **Finalization** that sends out match results and ends the session.

Activity functions

Activity function is a classic stateless Azure Functions that can be triggered from inside a **Durable Function**. Durable Functions depend on Activity Functions for performing steps of their logic, that execute code with non-deterministic results (any code that can be run twice with different results, such as generating a random number or querying a database).

Activity Functions in PvP game service are mostly used to retrieve or update data from persisted match sessions and push new *SignalR* messages to match-related connections.

7.4.3 Azure SignalR Service

As previously noted, client-server communication during a game match relies on real-time communication over *SignalR*. In standard server applications, *SignalR* can be self-hosted, but with serverless, *SignalR* endpoints must be fronted by *Azure SignalR Service*.

Early into development, it was assumed a remote *Azure SignalR Service* instance could be used for development with a locally running Functions Application. Unfortunately, this option works only for pushing messages downstream from the service to the mobile client. Upstream messaging is not possible, because it relies on registering a webhook with the Function Application. Since the local Functions Application only listens on a local IP address, the remote webhook cannot be registered.

For this thesis' scenario, I decided to use *Azure SignalR Service Emulator* instead, which is run locally, next to WordChain game service. This choice seemed sensible for a large part of development, but resulted in an unexpected issue with testing, which I was not able to solve within the available timeframe (see section 7.4.4).

7.4.4 State of implementation

The implementation state as of date does not fullfill pre-set architecture integration and functionality goals. Because testing the game with two players could not be performed (read 7.4.4), the implementation was altered at the last minute, editing service configuration for only one "player", to at least enable demonstration of implemented principles.

Testing issues

Late into the development process, attempts were made to use a locally running Android emulator and a physical Android phone connected to the local area network. The android phone was able to connect to local instances of IdentityServer and Data backend API, as well as HTTP endpoints from the Function Application, but always failed to connect to the `Azure SignalR emulator` after executing the initial *SignalR* connection negotiation. The issue occured when its service endpoint was set up with the same IP address as all remaining services. Connection from the emulated device showed no problems.

An alternative was attempted to test the application on two emulated devices, but the computer's memory and processing power turned out to be insufficient.

Alternatively, the game still could be tested e. g. from a simple browser application, but this idea was dismissed due to the size of this additional task.

Authentication

It was originally planned to secure service-side game API with custom OpenID authentication. Support for this feature is however available only as a part of Azure integration.

7.5 WordChain Mobile Application

WordChain native mobile application was implemented on Xamarin platform, making use of Xamarin.Forms and Xamarin.Essentials official packages, that provide abstractions of front-end and system services. The application's code is largely reusable on different platforms, but in reality WordChain was only developed and tried on Android, due to no access to devices natively running other platforms (e. g. iOS or Tizen). It combines functionalities provided by all three backend services to deliver previously proposed functionalities to the user.

7.5.1 Architecture Overview

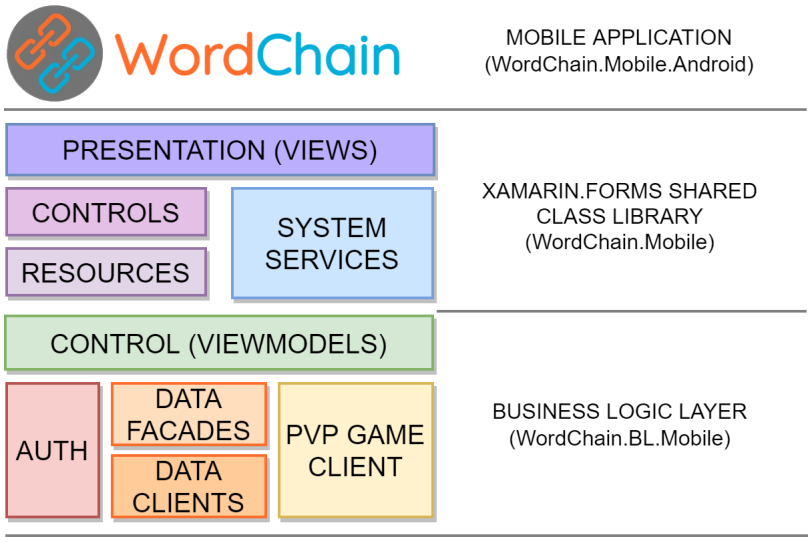


Figure 7.5: WordChain mobile application architecture

Diagram in figure 7.5 shows that the implementation is split into three layers. The upper Android layer generated by Xamarin was only mildly extended for platform-specific configuration and customization and is not going to be further discussed.

The shared Xamarin.Forms codebase layer and the business logic layer were implemented with **ModelViewViewModel** (MVVM) design pattern. It can be seen that in this implementation, the Views reside (as is common) in the Xamarin.Forms-templated layer, but the ViewModels are placed below, in the Business Logic Layer.

Business Logic Layer

Placing ViewModels in a separate layer from Views can be considered clean, but requires additional infrastructure. ViewModels need access to system service abstractions provided by the Xamarin libraries. At the bare minimum, a ViewModel needs to somehow access

navigation between pages, but WordChain also for instance uses the system browser for authentication (see 7.2.2). However direct use of system services would result in circular dependencies between layers, thus loose coupling had to be ensured:

- Business Logic Layer defines interfaces for services it requires but cannot use directly and depends on the upper layer to provide implementation.
- To make service available accross layers, a ServiceCollection pattern with Dependency Injection is used, in a way analogical to common ServiceCollection usage in ASP.NET Core projects.
- Navigation between Views is abstracted as navigation between ViewModels and then mapped in the upper layer to navigation between pages using a naming convention (e. g. `PlayerDetailViewModel` maps to `PlayerDetailView`).

BLL implements clients and additional models for working with the respective backend services. These are used by the ViewModels more or less directly.

Xamarin.Forms Shared Project

Based on its default template, the `WordChain.Mobile` project contains an entry point for the app initialization from the uppermost platform-specific layer and application Pages (MVVM Views) forming the presentation logic. The views bind to ViewModels which they receive upon initialization through constructor dependency injection. They make use of custom UI controls and visual resources (see subsection 7.5.2 for details).

This upper layer also contains implementations of services depending on Xamarin.Forms and Xamarin.Essentials libraries abstraction for access to platform-specific functionalities. Apart from app navigation, this also includes access to system storage, system browser and default e-mail client.

7.5.2 Graphical User Interface

WordChain's graphical user interface was designed and implemented based on feature requirements outlined in section 6.2.1. This subsection describes creation process by showing the design, elaborating on selected implementation details and subsequently delving into the final UI implementation experience.

UI Design

The design aims to be as simple and uncluttered as possible, with high contrast between colors and minimalistic styling. Figure 7.6 shows a part of an inital wireframe design. Most of the navigation between pages is done from a root page with a side flyout menu, since this is a straightforward and commonly used (and thus familiar) type of navigation. Because clickable navigation items in the flyout menu do not look like typical buttons, they are instead accompanied each with it's own material design icon.

Flyout root page schema works most intuitively, when the navigation stack is as flat as possible, it is therefore not advantageous to use extra layers of pages when navigating from a page to display related content. Because of this, it was decided to use popups instead wherever possible.

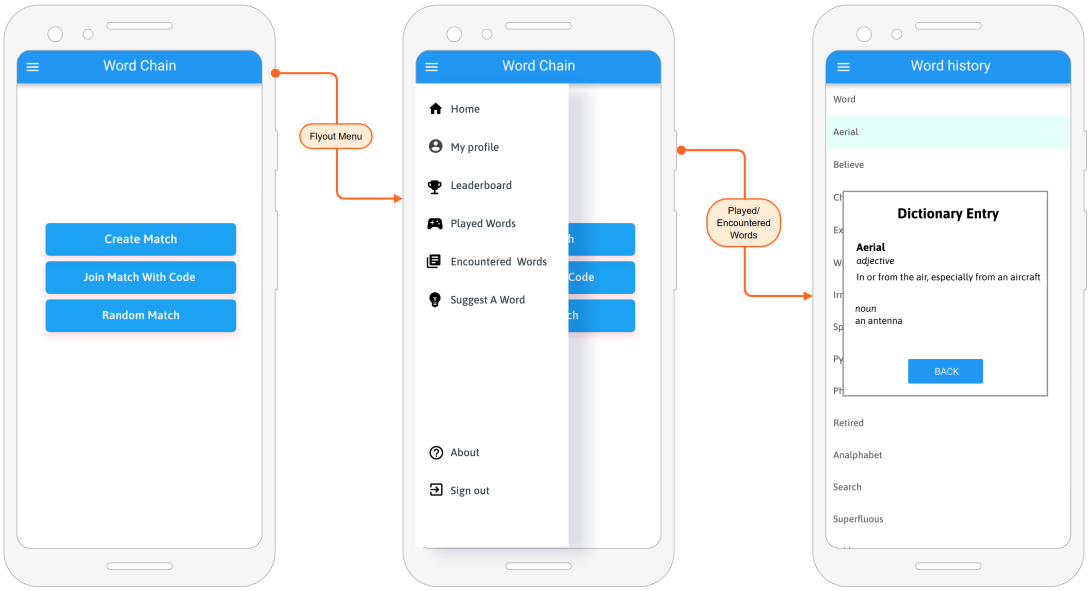


Figure 7.6: Exempt from initial wireframe, created with moqups.com

Xamarin Community Toolkit

Although Xamarin.Forms package comes with a basic set of predefined XAML controls that can be used out-of-box, there are many common use-case scenarios these controls do not cover. Because of this, the *Xamarin Community Toolkit*[6] was used in addition to Xamarin.Forms. *Xamarin Community Toolkit* (XCT) contains more controls as well as other elements, such as type converters and graphic effects). XCT controls are usually quite general and designed in a way that often requires additional customization for meaningful use. While this means XCT is not an out-of-box solution, it makes some common problems simpler and is usable with custom controls and templates.

Custom Controls

UI elements that do not come with Xamarin.Forms, XCT or as a part of a third-party package can be directly created on a XAML or C# Page by combining and nesting controls that are already available. This attitude is still used for some parts of WordChain UI, but some of such elements were instead implemented as reusable custom controls, leading to less repetitive XAML declarations.

Figure 7.7: `WordChainAvatarView` reusable custom control

[6] https://docs.microsoft.com/en-us/xamarin/community-toolkit/

Figure 7.7 shows examples of `WordChainAvatarView` custom control for three different users with different user images. Its implementation consists of two parts:

- a C# class that inherits Xamarin.Forms *ContentView* class. This class defines new bindable properties that allow control customization from within the parent XAML element, where the control is used. In this case the properties are `ProfileImage`, `AvatarSize`, `ProfileImageSize` and `Text`.
- a XAML *ControlTemplate* resource, which declaratively defines appearance of the control. Control Template consists of combinations of existing custom, Xamarin.Forms or third-party controls, that also use XAML resources (even other control templates). For `WordChainAvatarView`, these components are a couple of Xamarin.Forms *Frames* and an XCT *AvatarView*[7].

This attitude toward creating custom controls is more versatile than creating custom controls directly as XAML elements with C# code-behind, because there can be multiple Control Template definitions for one *ContentView* descendant.

On top of that, control templates also support inserting custom content by passing it into their inner *ContentPresenter*. In WordChain, this feature was essential for creating the `DataLoadingStateView` custom control. `DataLoadingStateView` implements a unified way of indicating loading of data from remote services, but thanks to *ContentPresenter* retains the ability to display any custom data that when it is eventually loaded.

Final UI

Figure 7.8 shows the final user interface implementation for in-progress WordChain game session a dictionary entry retrieved for a word from player's personal word history and a player profile edit page.

Development experience

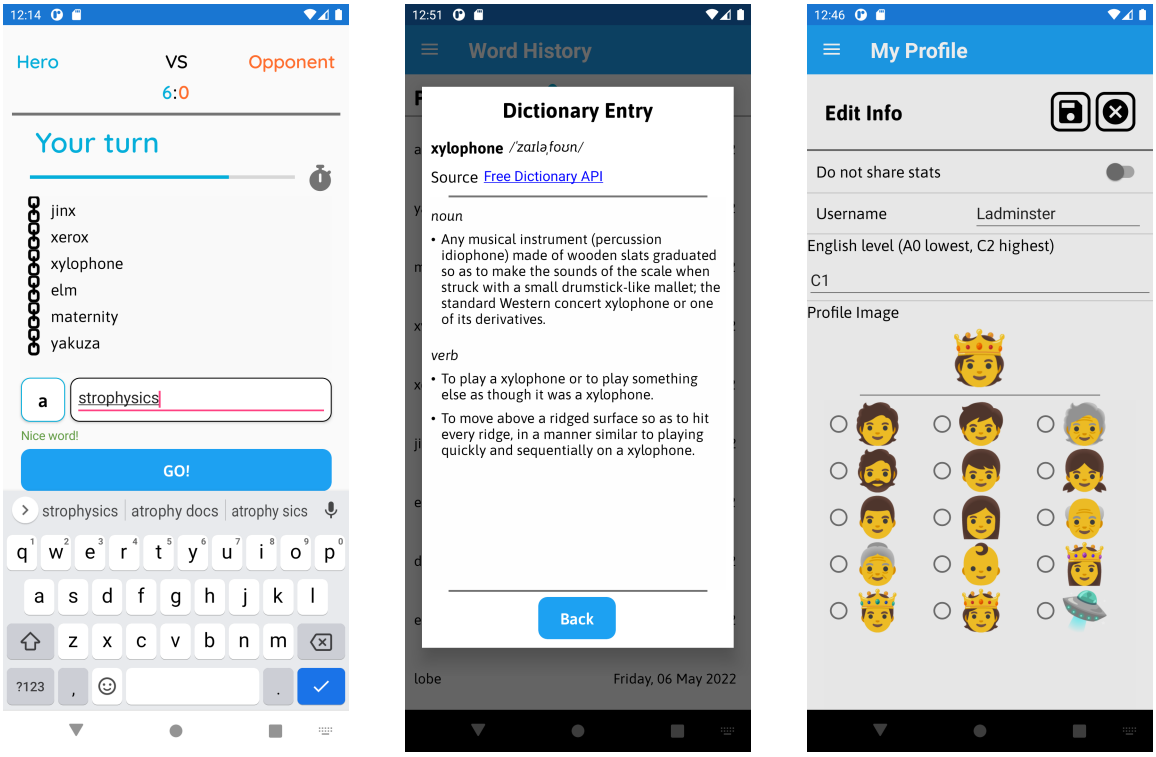
Even though the GUI is minimalistic in styling and utilizes maximum of existing Xamarin.Forms components, implementation was still very time consuming compared to other parts of the solution. This was also contributed to by the fact that although Visual Studio has a XAML Hot Reload feature for showing XAML implementation changes while the application is running, this feature fails in certain scenarios. Some of these 0were encountered in the app, such as modal pages, some custom components, and the XCT *Popup*[8], which was used as a base for custom popup implementation.

Beside this issue, the multiplatform approach with Xamarin.Forms proved to limit styling options for many existing components, some of which could only be styled by implementing platform-specific UI renderers. This option was taken under consideration, but eventually dismissed.

7.5.3 Client-side Game implementation

As noted before in subsection 7.4.2, the native client itself was originally intended to contain little to no game logic, depending for everything on the PvP game service. With that in mind, client-side implementation originally focused on display and communication logic

[7] https://docs.microsoft.com/en-us/xamarin/community-toolkit/views/avatarview
[8] https://docs.microsoft.com/en-us/xamarin/community-toolkit/views/popup



(a) WordChain match (b) Played word entry (c) Profile editation

Figure 7.8: Screenshots from the final UI implementation

only. It was however eventually determined, that some functionalities (e. g. input validation and persisting played and encountered words for individual players) operate more smoothly on the client side.

Overview

Figure 7.9 shows a diagram depicting the client-side implementation. It can be seen right away, that it has added complexity compared to ViewModel patterns with services consiting exclusively from fetch operations (such as consuming data from a REST API).

As explained in section 7.4.3, communication during a PvP match uses *SignalR*. Since in WordChain's scenario *SignalR* is used for both upstream and downstream communication, `PvPGameClient` defines not only methods invoking endpoints on the cloud, but also registers a set of callbacks for handling incoming *SignalR* messages. These handlers further push information to the ViewModel.

Pushing operations are facilitated by a handler and observer class, that together form an Observer design pattern. In this case, `PvPGameClient` exposes a public handler instance and `PvPMatchViewModel` contains an instance of the observer. `PvPMatchViewModel` retrieves the public handler instance and subscribes the observer to receiving updates from it. `PvPMatchViewModel` also holds an instance of a display model locally representing state of the match. This display model instance is shared with the observer, so that the observer can update it. The uppermost `PvPMatchView` is then bound in a standard way to the ViewModel property holding the display model instance, and thus updates user interface with changes, finally completing the push.

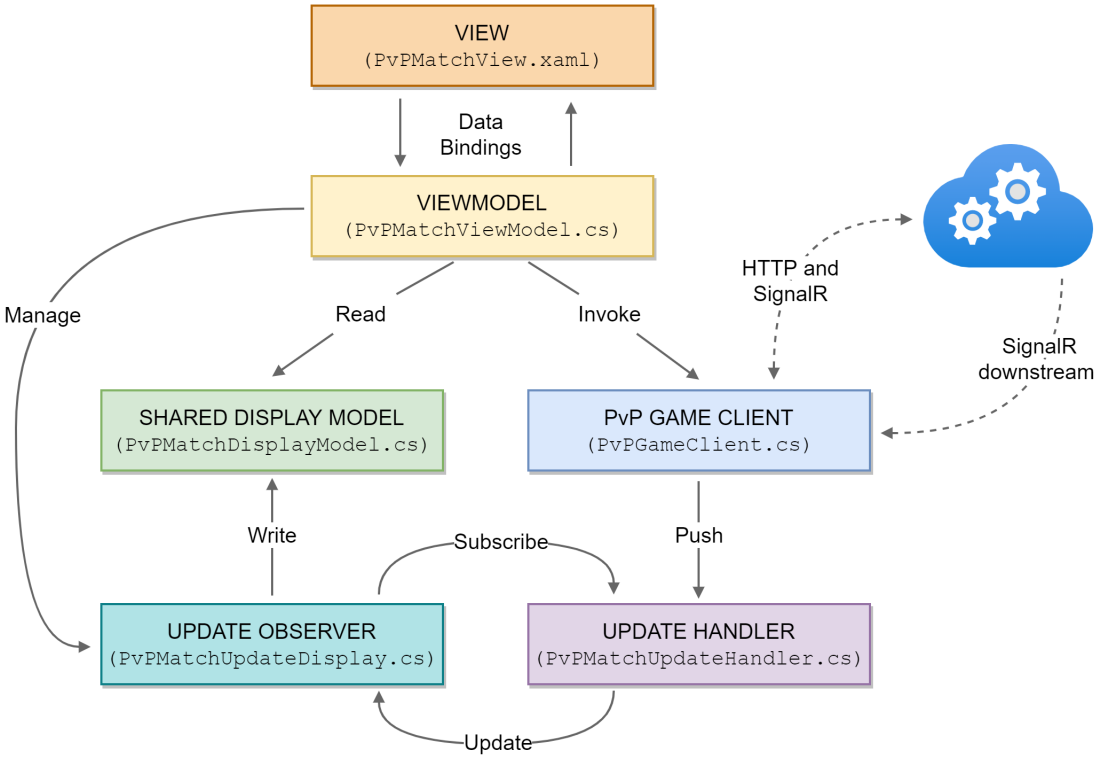


Figure 7.9: Client side game logic with MVVM and Observer pattern

Data mapping between Observer layers is done via *Automapper*[9].

Turn Input Validation

Turn input is validated against a dictionary API[10] to determine whether the input is a permitted existing word (see 6.3). If the input is not found, validation against WordChain's approved user word suggestions is attempted.

Since the validation must be performed as fast as possible, the functionality was moved from cloud services to the application to save a round trip. Because of this, validation queries are sent immediately when player stops typing and results are temporarily kept in a local list cache to reduce the number of outgoing requests with certain typing behaviors.

Limitations

Debugging and testing has shown that even though the synchronous Observer pattern with MVVM works sufficiently in most encountered cases, it is prone to issues stemming from the asynchronous nature of *SignalR* callback invocation, which can lead to race conditions when pushing to the handler. In the future, instead of trying to mitigate this behavior, it might serve better to replace MVVM+Observer with a pattern better-suited for this scenario. The Elm Architecture (or Model-View-Update) pattern appears to be a possible solution [9].

[9] https://docs.automapper.org/en/stable/
[10] Free Dictionary API – https://dictionaryapi.dev

Chapter 8

Deployment and Evaluation

Toward the end of implementation phase described in chapter 7, phase of deployment was started in parallel. After deployment was completed for production-viable parts of the solution, the application was tested in closed environment and feedback from the testing was collected. Suggestions for further development based on this feedback are included in a closing section evaluating development process and its outcomes as of date.

8.1 Deployment

Two parts of the solution reached a level of robustness where deployment was a viable option: the WordChain data backend[1] and IdentityServer[2]. The serverless PvP game service was not suitable for publishing given the state of implementation (see subsection 7.4.4), but it still connects to two remote Azure services.

8.1.1 WordChain Resource Group

Resource	**Service type**
WordChainDataApi	*Azure App Service*
IdentityServer	
WordChainDbServer	*Azure SQL* database server
WordChainDataDb	*Azure SQL* database
IdentityServerDb	
WordChainGameDatabase	*Cosmos DB* NoSQL database
WordChainGameFuncStorage	*Azure Storage* account

Table 8.1: WordChain resource group

Deployed portion of application backend services are resources organized into the WordChain resource group. Resources and their respective service types are listed in table 8.1. The group also contains service billing plans for each service. All billing options for deployed infrastructure were set to minimal consumption plans, which are free with the used *Azure for Students* service subscription.

[1] https://wordchaindataapi20220419212513.azurewebsites.net/swagger/
[2] https://identityserver20220419192104.azurewebsites.net/.well-known/openid-configuration

8.1.2 Monitoring and Diagnostic

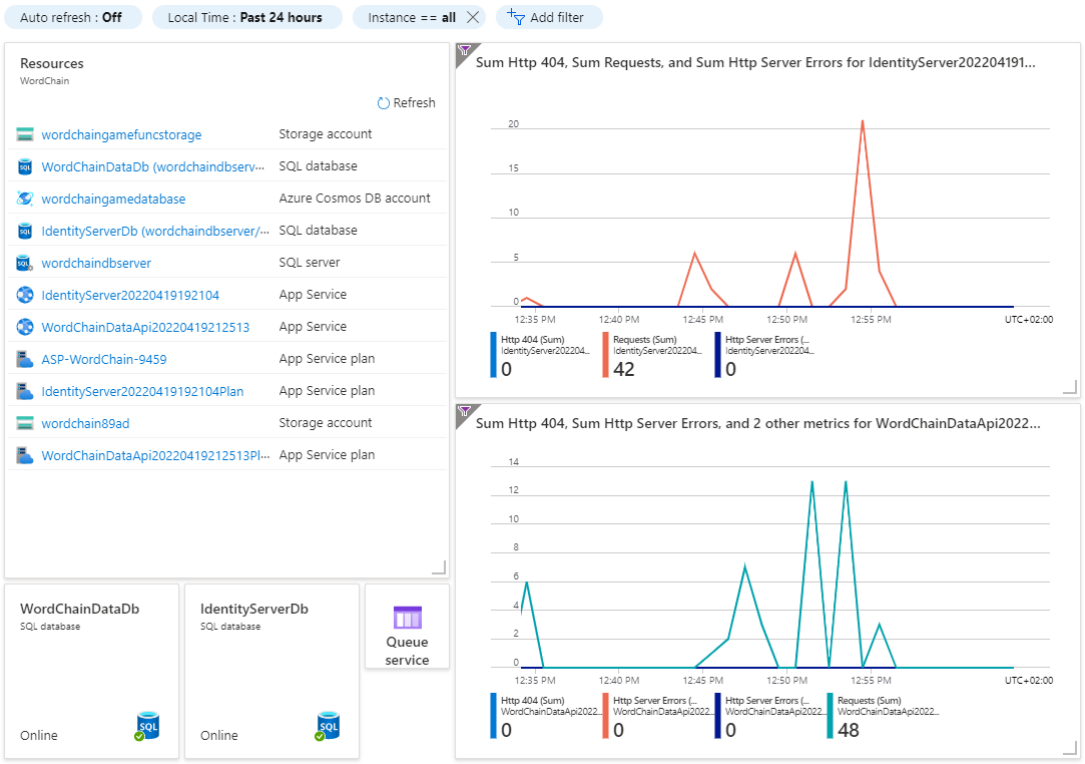


Figure 8.1: WordChain Azure Portal dashboard

Services published to Azure are automatically set up with *Azure Monitoring* service, which keeps track of various sets of metrics depending on service type. For providing compact overview of state and performance of related services, metrics can be customized and, together with other miscellaneous information, placed into an Azure Portal dashboard.

Figure 8.1 shows WordChain monitoring dashboard, that lists services in the WordChain resource group, displays network of *Azure SQL* databases and visualizes selected HTTP communication metrics for IdentityServer and WordChainDataApi.

Logging

Prior to deployment, both web services were configured with *Serilog*[3] logging library for ASP.NET Core. The configuration is very basic as it simply writes all messages into a text file on the host.

When hosting on an *Azure App Service* instance, this minimal setup becomes substantially more useful. Azure is able to automatically retrieve logging configuration from the application and bind it to a real-time log stream (also accessible through Azure Portal).

[3] https://github.com/serilog/serilog

8.2 User Testing and Feedback

The application was tested by a group of 5 users. Before the testing began, users were acquainted with current state of PvP game implementation, so that they could better focus their feedback.

8.2.1 Testing process

Each user was tasked with logging into WordChain, interacting with specific features and afterwards logging out. It was observed whether and how fast they were able to perform these tasks. Afterwards, they were asked a series of open-ended questions about interface friendliness and navigation intuitivity, as well as quality and their perceived utility of tested functionalities.

Finally, users were asked to start a match and play according to rules. After finishing, they were asked about their opinion of the game principle and if they would have use for word suggestion and word history features.

8.2.2 Feedback and suggestions

User feedback indicated that, for the most part, the application was easy to navigate and the UI was well-arranged with decent interaction feedback. However, two users completely missed that lists of played words and players in a leaderboard are clickable for details. They suggested adding a distinct button or a description that an item should be tapped. Beyond that, one remark was made, that should emojis serve as user profile images, selection should be broader, but the user would still prefer using their own images.

From general functionality, all players voiced interest in an **invite-to-play** feature to improve matchmaking.

Regarding the PvP game principle demonstration, users were happy with word validation speed, but initially confused that they were expected to only input suffixes of words to the given starting letter, and typed in entire words instead. Overall, they expressed interest in a fully functional PvP game and confirmed that accompanying it with the word history feature makes the it more meaningful.

8.3 Possible Improvements

The most crucial improvement to be done in the future, is solidifying and properly testing PvP Game implementation to move it beyond minimal demonstration functionality, and deploy it to the cloud. Additional improvement options were identified based on user feedback, technical limitations and own ideas:

- Once fully operational, PvP game should offer an invite-to-play option.
- Some interactive UI elements should be highlighted, because they are easy to miss.
- The game might benefit from adding a "Forfeit" button. That way a user would not have to wait for their turn to time out in order to safely quit to main menu.
- IdentityServer implementation should support user registration and its database should be fronted with a cache, because authentication process in the application is too slow.
- Given the size of this software solution, comprehensive testing should be introduced.

Chapter 9

Conclusion

The main goal of this thesis was to propose, design, implement and test a mobile game for practicing English, based on the rules of word chain, with purpose of helping users with expanding and practicing their vocabulary.

First, a game-based learning and mobile assisted learning evaluation was conducted, which concluded that a well-designed mobile game can combine educational aspects with entertainment and increased engagement.

This knowledge was complemented by assessment of currently existing solutions, number of which turned out to be based on very similar ideas. The new design could then build on their obvious strengths while avoiding possible shortcomings.

Prior to realization, development options of .NET Ecosystem were compared in order to select the best-suited .NET implementation and programming language. Fundamental Cloud computing theory was summarized, followed by comparison of services offered by current cloud computing industry leaders.

In subsequent design phase, game concepts were lined out. Use cases and persistent data were defined. The game was conceptualized as a one-on-one turn-based online match. Top-level architecture of the application solution was proposed as several interconnected web microservices forming a back-end for a mobile client application. Development tools and deployment platforms were selected.

The software solution was developed in C# and XAML. Backend was implemented in three parts: a REST API responsible for operations above application data, an OpenID authentication provider securing the API and the native mobile application, and a serverless Azure Function executing game logic. Mobile client was implemented on Xamarin platform and fully depends on backend services to operate.

Data backend and authentication services were successfully deployed to Azure and are hosted on instances of *Azure App Service*.

Because of unexpected blocking issues while testing a locally running serverless application with bi-directional real-time communication, the game was at no point tested for two players. As a middle ground solution, game implementation was altered to enable demonstration of implemented principles with one player. The service was not deployed.

In the end, user testing was conducted. Based on gathered user feedback and current technical limitations, room for improvement was identified to be addressed in the future.

The final product of this thesis is not a fully operational software solution for the explored problem, but it serves as proof of concept and a basis for further development.

Bibliography

[1] Abbas, M. F. F. Applying Word Chain Game to Improve Students' Vocabulary Mastery. *ELT-Lectura*. 2014, vol. 1, no. 1. Available at: http://journal.unilak.ac.id/index.php/ELT-Lectura/article/view/450/318.

[2] All, A., Nuñez Castellar, E. P. and Van Looy, J. Assessing the effectiveness of digital game-based learning: Best practices. *Computers and education*. Elsevier Ltd. 2016, vol. 92, p. 90–103. ISSN 0360-1315.

[3] Allen, B. and Dominick, B. *Welcome to IdentityServer4* [online]. Brock Allen & Dominick Baier, 2020 [cit. 2022-04-30]. Available at: https://identityserver4.readthedocs.io/en/latest/.

[4] Bachen, C. M., Hernández Ramos, P., Raphael, C. and Waldron, A. How do presence, flow, and character identification affect players' empathy and interest in learning from a serious computer game? *Computers in Human Behavior*. Elsevier. 2016, vol. 64, p. 77–87. ISSN 2451-9588.

[5] *Word Chain* [online]. Boston Children's Museum [cit. 2020-12-26]. Available at: https://www.beyondthechalkboard.org/activity/word-chain/.

[6] Breien, F. S. and Wasson, B. Narrative categorization in digital game-based learning: Engagement, motivation & learning. Wiley. 2021.

[7] Britch, D., Coulter, D., Schonning, N. and Dunn, C. *Xamarin.Forms XAML Basics* [online]. Microsoft, 2017 [cit. 2021-01-14]. Available at: https://docs.microsoft.com/en-gb/xamarin/xamarin-forms/xaml/xaml-basics/.

[8] *How are language levels described?* [online]. Cambridge University Press, 2020 [cit. 2021-01-03]. Available at: https://www.cambridgeenglish.org/learning-english/parents-and-children/information-for-parents/tips-and-advice/011-the-cefr/.

[9] Czaplicki, E. *The Elm Architecture* [online]. Elm, 2020 [cit. 2022-05-08]. Available at: https://guide.elm-lang.org/architecture/.

[10] Harvey, C. and Patrizio, A. *AWS vs. Azure vs. Google: 2021 Cloud Comparison* [online]. Datamation, 2020 [cit. 2021-01-16]. Available at: https://www.datamation.com/cloud-computing/aws-vs-azure-vs-google-cloud-comparison.html.

[11] *Word Chain ESL Vocabulary Game* [online]. ESL Expat, 2019 [cit. 2020-12-26]. Available at: https://eslexpat.com/esl-vocabulary-games/word-chain/.

[12] Govender, T. and Arnedo Moreno, J. An Analysis of Game Design Elements Used in Digital Game-Based Language Learning. *Sustainability (Basel, Switzerland)*. MDPI AG. 2021, vol. 13, no. 12, p. 6679. ISSN 2071-1050.

[13] Guise, P. de. The Cloud. In: *Data Protection: Ensuring Data Availability*. 1st ed. Taylor & Francis Group, 2017, p. 195–224. ISBN 9781482244151.

[14] Hermes, D. *Xamarin mobile application development : cross-platform C# and Xamarin.forms fundamentals*. New York, NY: Apress, 2015. ISBN 978-1-4842-0215-9.

[15] Johnson, J., Britch, D. and Dunn, C. *What is Xamarin.Forms?* [online]. Microsoft, 2020 [cit. 2021-01-14]. Available at: https://docs.microsoft.com/en-gb/xamarin/get-started/what-is-xamarin-forms.

[16] Khaddage, F., Lattemann, C. and Acosta Díaz, R. Mobile gamification in education engage, educate and entertain via gamified mobile apps. In: Association for the Advancement of Computing in Education (AACE). *Society for Information Technology & Teacher Education International Conference*. 2014, p. 1654–1660.

[17] *Gamification* [online]. [cit. 2020-12-28]. Available at: https://www.merriam-webster.com/dictionary/gamification.

[18] Nazarova, M. and Galiullina, E. Game-based learning in teaching English. *Journal of Organizational Culture, Communications and Conflict*. Allied Academies. 2016, vol. 20, p. 8–12. ISSN 15440508.

[19] Ortinau, D., Coulter, D., Dunn, C., Schonning, N., Cholachagudda, P. et al. *Installing Xamarin.iOS on Windows* [online]. Microsoft, 2018 [cit. 2021-01-15]. Available at: https://docs.microsoft.com/en-us/xamarin/ios/get-started/installation/windows/?pivots=windows.

[20] Prensky, M. Digital game-based learning. *Computers in Entertainment (CIE)*. ACM New York, NY, USA. 2003, vol. 1, no. 1, p. 21–21. ISSN 1544-3574.

[21] Qian, M. and Clark, K. R. Game-based Learning and 21st century skills: A review of recent research. *Computers in human behavior*. Elsevier Ltd. 2016, vol. 63, p. 50–58. ISSN 0747-5632.

[22] Qiu, M. and Gai, K. *Mobile Cloud Computing: Models, Implementation, and Security*. 1st ed. CRC Press, 2017. ISBN 9781498796033.

[23] Radich, Q., Kennedy, J., Walker, J. et al. *What's a Universal Windows Platform (UWP) app?* [online]. Microsoft, 2021 [cit. 2022-01-20]. Available at: https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.

[24] Shi, Y.-R. and Shih, J.-L. Game Factors and Game-Based Learning Design Model. *International journal of computer games technology*. Hindawi Publishing Corporation. 2015, vol. 2015, p. 1–11. ISSN 1687-7047.

[25] Singaravelu, G. Video Game Based Learning in English Grammar. *Journal of Educational Technology*. i-manager Publications. 2008, vol. 5, no. 3, p. 49. ISSN 0973-0559.

[26] Skeet, J. *C# in depth*. Fourth editionth ed. Manning, 2019. ISBN 978-1-61729-453-2.

[27] Software, S. *OpenAPI Specification* [online]. SmartBear Software, 2011 [cit. 2022-04-30]. Available at: https://swagger.io/specification/.

[28] Sotsenko, A., Zbick, J., Jansen, M. and Milrad, M. Flexible and Contextualized Cloud Applications for Mobile Learning Scenarios. In:. 2016, p. 167–192. ISBN 978-3-319-26516-2.

[29] *Gamification* [online]. University of Oxford [cit. 2020-12-28]. Available at: https://www.lexico.com/definition/gamification.

[30] *AWS Vs Azure Vs GCP – The Cloud Platform of Your Choice?* [online]. Veritis, 2020 [cit. 2021-01-16]. Available at: https://www.veritis.com/blog/aws-vs-azure-vs-gcp-the-cloud-platform-of-your-choice/.

[31] Warren, G. *.NET implementations* [online]. Microsoft, 2022 [cit. 2022-01-20]. Available at: https://docs.microsoft.com/en-us/dotnet/fundamentals/implementations.

[32] Warren, G., Coulter, D., Lee, D. et al. *Overview of .NET Framework* [online]. Microsoft, 2021 [cit. 2022-01-20]. Available at: https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview.

[33] Warren, G., Dykstra, T., Wagner, B. et al. *.NET Standard* [online]. Microsoft, 2022 [cit. 2022-01-20]. Available at: https://docs.microsoft.com/en-us/dotnet/standard/net-standard.

[34] Warren, G., Wagner, B., Schonning, N. et al. *Common Type System & Common Language Specification* [online]. Microsoft, 2021 [cit. 2022-01-20]. Available at: https://docs.microsoft.com/en-us/dotnet/standard/common-type-system.

[35] Wu, Q. Pulling mobile assisted language learning (MALL) into the mainstream: MALL in broad practice. *PloS one*. Public Library of Science. 2015, vol. 10, no. 5, p. e0128762. ISSN 1932-6203.

[36] Wu, T.-T. and Huang, Y.-M. A Mobile Game-Based English Vocabulary Practice System Based on Portfolio Analysis. *Journal of Educational Technology & Society*. International Forum of Educational Technology & Society. 2017, vol. 20, no. 2, p. 265–277. ISSN 11763647.

[37] Çeker, E. and Özdaml, F. What "Gamification" Is and What It's Not. *European Journal of Contemporary Education*. Academic Publishing House Researcher. 2017, vol. 6, no. 2, p. 221. ISSN 2304-9650.