# New Product Design and Implementation of Aboleth: a Mobile D&D Character Creator for enterprise mobile applications and Metaverse

**Victor Chang[1], Dan Lawrence[2], Le Minh Thao Doan[2], Ariel Qianwen Xu[1], Ben S.C. Liu[3]**

[1] Department of Operations and Information Management, Aston Business School, Aston University, Birmingham, UK

[2] Cybersecurity, Information Systems and AI Research Group, School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, UK

[3] School of Business, Quinnipiac University, Connecticut, USA

Emails: v.chang1@aston.ac.uk/victorchang.research@gmail.com; daniellawrence2468@gmail.com; minhthaodoanle@gmail.com; qianwen.ariel.xu@gmail.com; ben.liu@QU.edu

**ABSTRACT**

As mobile applications have proliferated in recent years, it has become a new emerging contributor to the game market. With the continuous improvement of power and user interface facilities, many mobile applications for gamers have been presented on smartphones. To design and implement such applications, methodological design and development procedures are required. This paper presents our contribution to rapid prototyping for Aboleth, a mobile application intended to serve as a virtual character sheet for playing Dungeons and Dragons 5th edition. We propose to augment an API framework, including details of the general design process and crucial details regarding the specific implementation of the project. The prototype is developed using different latest mobile application development methodologies such as the Computer-Assisted Designs and Xamarin. Software engineering is integrated into game design to improve Human-Computer Interaction and eventually enhance user experiences. We discuss the relevance of using such a tool to achieve rapid prototyping for enterprise mobile applications and Metaverse and explore the opportunities to improve the project design.

**Keywords**: Product design, Software engineering, Human-computer interaction, Xamarin, Agile, enterprise mobile applications and Metaverse.

## 1. INTRODUCTION

Mobile application development has been rapidly changing due to its popularity and ubiquity among consumers. Presently, millions of downloaded mobile applications from app stores (e.g., Apple App Store, Google Play Store, and the Windows Phone Store) are used by users across the globe. They are also one of

the most widely used channels worldwide, with easier and faster communication and changing perspectives, enhancing users' experience and building a brand's awareness (Lin et al., 2021). The rapid growth and the variety of mobile platforms present an unprecedented challenge to develop the same mobile applications separately across each target platform. The industry's typical practice is applying a native development strategy to develop mobile apps but lacks code reuse for another platform. Therefore, the same application must be re-implemented from scratch (Ahmad et al., 2018). Indeed, mobile app development is still cumbersome, and it lacks a methodology geared toward sustaining the development of such mobile applications (Birgitta, 2009).

The new product development process and implementation of enterprises' applications have been emphasized in the literature as the crucial aspect for firms to survive and improve innovation performance in the current fast-changing business environment (Zhu et al., 2019). Hence, there is a need for a mobile application design and implementation prototype, where the development of mobile applications for enterprises is more standardized and transparent. The project detailed in this paper is based primarily on an idea to create a suitable software replacement for the character sheet of Dungeons and Dragons (D&D) Game. This concept aims to develop an application that allows characters to be created and managed efficiently by providing a simple, user-friendly experience. The application would include an optimized step-by-step process for creating a character, followed by an automatically generated character sheet that calculates core character statistics and allows users to select edits to a character where appropriate.

D&D is an immersive fantasy role-playing game (RPG) initially developed by Dave Arneson and Gary Gygax in 1974. Since it first launched, D&D has uplifted players worldwide through numerous hours of imagination and adventure. The worlds created by individuals acting as Dungeon Masters are immersive and complicated, and both creating and playing within them require tremendous amounts of information and imagination. A well-designed character has sufficient character to generate scenes in the game that the player finds engaging and concomitant mechanical support such that the character performs at the desired level during those scenes. These factors could impact players' success and unexpected failure and, subsequently, their engagement with the game. The kind of games in mobile devices grows quickly and opens a new door for business development and researchers to propose a rapid process of game design and implementation with high performance and satisfying consumers. Many studies about the technique of adapting characters to the player's interests have been implemented based on D&D (Faria et al., 2019, Martin et al., 2018). Therefore, we use D&D as the target for our new product design and implementation illustration. The prototype describes the process from identifying the market gap to API implementation.

### 1.1 Research Contributions

The contributions of this paper are three-fold. First, it provides an overview of the problem solution and practical design considerations to other researchers and practitioners interested in developing enterprise mobile applications. The prototype has integrated software engineering and game design to improve Human-Computer Interaction on mobile applications. Remarkably, our paper focuses on user-centered design and HCI to improve user interface, efficiency, and satisfaction and eventually enhance user experiences.

The second contribution demonstrates how Xamarin, a leading cross-platform for mobile applications and Agile framework, could flexibly adapt to the enterprise's new application design and implementation process. We describe the project in detail and explain how it is implemented. Later, we demonstrate the feasibility of our prototype in enterprise practice. This prototype provides a solution for businesses, especially small and medium-sized ones, to flexible and fast design and build their mobile apps. Subsequently, it facilitates the development of enterprise technology and information systems.

Finally, we present a new paradigm of using the C4 model (2021) solution that maps out the relationship between each of the individual feature deliverables and their data sources. Subsequently, the solution can improve the performance of mobile applications and foster the mobile application process. This is reflected and discussed throughout the entire process to improve the design of the enterprise application. Also, security issues are highlighted throughout the design and implementation process.

The paper is organized as follows: first, we present a brief introduction to the prototype, followed by a review of relevant studies. Next, we describe the step-by-step prototype of the new product design and implementation project. Finally, we conclude and discuss our development and implementation process.

## 2. LITERATURE REVIEW

### 2.1 Dungeons & Dragons (D&D) and Human-Computer Interaction (HCI)

The first popular gaming system, Dungeons & Dragons (D&D) (TCHERNAVSKIJ et al., 2022), formalized the basic form of tabletop role-playing games (TRPGs), which was created to perform fantasy stories inspired by fictitious realms like Tolkien's Middle Earth. A number of gaming systems have the basic qualities of D&D, but they are tailored for various fictional genres such as horror, detective stories, and science fiction.

A MUD (Multi-User Dungeon) is a text-based Dungeons and Dragons type of real-time role-playing game available on the Internet. It first appeared in the late 1970s and is the prototype for today's online role-playing games, which are totally text-based, with no visuals (Birmingham, 2021). Online role-playing games today may be thought of as a graphical version of MUDs. MUD creates a virtual world for players and allows a number of players to engage in adventures, wander about and interact with other players at the same time. Moreover, players are able to create their own objects and environments according to their own preferences (Hsu et al., 2009).

Over the past 20 years, Human-Computer Interaction (HCI) has been instrumental in bringing digital technology to TRPGs to produce an enhanced gaming experience. MUDs, for example, provide a more advanced type of conversation at its time. It is rapid conversational communication interspersed with descriptions of feelings, virtual objects, and locations (Olson and Olson, 2003). Players can comment on an action or a feeling, such as " Judy left the room." Objects also play a role in the exchange, with many possessing "magical" characteristics that award a variety of abilities or gifts to the recipient.

Dungeons & Dragons Online (DDO) was the first massively multiplayer online role-playing game (MMORPG) to include voice chat in its system when it was introduced in 2005 (Wadley et al., 2015). When players join a group, the DDO voice channel is set up so that they can communicate with their teammates in a "two-way radio" mode. After the group split, players are not allowed to interact with non-teammates or ex-teammates. By incorporating speech into the game, channel management could be connected with team management.

Many studies have concluded that the introduction of computational elements in TRPGs can improve the quality of the experience, such as immersion and player satisfaction. To address user experience difficulties, they apply strong HCI ideas, including interactive wearables based on the computer system (Buruk & Özcan, 2018), tangible objects (Plijnaer et al., 2020), animated visuals based on Android system (Bengtsson,& Jursenaite, 2019), and mixed reality environments based on Android and IOS systems, respectively (Rizov et al., 2019), in addition to the communication tools discussed above. The previous studies emphasize TRPG play as a user experience rather than a smart collaborative activity.

### 2.2 Mobile Application Development and Interface Designs

There are mainly three types of mobile application development: native, hybrid, and mobile web apps. The native mobile application is designed to run in a specific mobile operating system, i.e., Android, iOS, using a particular programming language and toolkit of the target platform, which leads to a device's full capabilities that can be achieved and utilized (Nawrocki et al., 2021). As a result, it is more likely to optimize the UX to operate more quickly and perform better. However, developing two or more native apps corresponding to a cross-platform app increases development and maintenance costs as additional resources and expertise with the programming languages such as API (Application Programming Interfaces) and SDK (Software Development Kits) of each platform (Alsaid et al., 2021). Next, the second type is the hybrid one, where mobile apps are installed on the devices and run through web browsers. These applications can access the device hardware features but are not as reliable or fast as native mobile applications. The final type is a mobile web application that allows users to access via mobile web browsers. Although this approach is highly portable across multiple mobile platforms, which ultimately reduces development cost and time, it can not use the user's device hardware features such as the camera and application stores.

Some popular cross-platforms for mobile application development are Xamarin, React-Native, and Ionic. Xamarin is a leading free, open-source platform for building cross-platform mobile applications. It is a Microsoft development platform for code native and cross-platform in C# (Hermes, 2015). This platform's key concepts focus on code sharing, enhancing testability, decreasing overall development, and maintaining resources. The primary difference from other frameworks is the Model-View-ViewModel pattern (MVVM). It is also the framework we applied in this paper for our new product design and implementation. According to Vishal and Kushwaha (2018), the Xamarin platform comprises six-layered architecture:

- The Data Layer is where data can be saved on a mobile device. SQLite database is usually implemented, but sometimes any other suitable mechanism with XML files can be used as an alternative approach.

- The Data Access Layer can connect to Xamarin. Forms application to the database to operate creating, reading, updating, and deleting queries. The data access layer can be easily integrated with the user interface (UI) layer to perform the above operations.
- The Business Layer or Business Logic Layer comprises business entity definitions for programmers to implement tasks such as access management.
- The Service Access Layer is used to access services, including data transfer, data retrieval, and web services in the cloud and remote servers.
- The Application Layer is where the code writing for the particular application. The UI, the class definitions, and the impact of multiple screens or device sharing must be maintained using the application layer.
- The UI Layer manages the user screen and can be done through Xamarin.Forms controls. It includes the widgets, screen orientations, sizes, navigations, and controllers, different for various devices.

### 2.3 Human-Computer Interaction

Human-Computer Interaction (HCI) is a field of research focused on interfaces between users and computers. Therefore, it allows interrelating the human with an electronic device capable of solving several contemporary mobile application development problems. According to Chen et al. (2019), HCI could refer to how people use input and output devices to communicate with computers and complete tasks. According to Dix (2009), Observation and Empirical Data, Usability, Design and Methodology, Representation and Analysis and Implementation, and UI Architecture are the fundamentals of the HCI field. With HCI, the transmission of information between people and devices is realized. The key concept of HCI is the interface. The sensors, gestures, and location data play a dominant role in several mobile applications compared to computer applications. The different styles and smaller displays of UI also significantly impact interaction design for mobile applications. A primary issue in designing mobile applications is that users often wish to perform rapid, focused actions instead of long-lasting sessions like applications running on a desktop computer (Kretschmer, Mättig & Fiolka, 2021). Consequently, HCI has been highlighted as the key successful driver of interface mobile app design to achieve comprehension and interaction between users and devices (Mohammed & Karagozlu, 2021).

The HCI development and application have been critical, as it is required to implement with appropriate usability and security matter (Cochran et al., 2017, Harshul et al., 2017) and satisfy the need of users efficiently (Xu et al., 2017). The rapid advancement of technology has facilitated HCI; however, it has faced many challenges in practice, particularly designing an efficient and effective approach from HCI practitioners for mobile applications (Mohammed & Karagozlu, 2021). HCI design concentrated more on usability in the past, but recently, its focal interest became user experience, emphasizing the emotions and interaction of users with the product and improving their satisfaction. Therefore, a new process for rapidly developed design, evaluation, and interaction is required for fast-changing interaction systems and interfaces, facilitating efficient and unique user interactions with computer systems (Tomitsch et al., 2021).

Concurrently, HCI strives to create, develop, and evaluate computer-based interactive systems to improve user efficiency and satisfaction. A clear understanding of user experience is required before adopting any

technical response. Hence, design procedures and principles which promote seamlessness, transparency, and tangibility to enrich user experience have also arisen. The main goal is to trigger the interaction between users and product interfaces for specific tasks. Consequently, software and hardware are developed, compatible with the needs of users~~, functional, and user-friendly~~. HCI researchers and practitioners have proposed many strategies to encounter such problems in software engineering for interactive systems such as user-centered design or UCD (Bosser et al., 1992), GOMS modeling (Gray et al., 1993), and usability engineering lifecycle (Mayhew, 1999).

In HCI, the user intercommunicates with the mobile device. In this communication, the system interacts to mediate between the user and tasks, while the user interacts to execute tasks according to the needs and goals. Such interaction utilizes the UCD technique to use the system more effectively and learn more efficiently. In UCD practices, designers profiled the user, defined the preferences and behaviors for different aspects of a provided application, and utilized the relevant information to make design judgments (Williams, 2009). UCD helps designers understand users more within a context by iteratively using their participation in each iteration. Integrating it through our prototype allows a more descriptive and meaningful interpretation of decisions while conducting this research as an iterative process. According to the ISO 9241-210 standard (2010), six fundamental principles are recommended to ensure an effective UCD project, as below:

- An explicit understanding of users, tasks, and environments is fundamental to the design.
- Users are involved throughout the design and development.
- The design is driven and refined by user-centered evaluation.
- The design addresses the whole user experience.
- The design team includes multidisciplinary skills and perspectives.
- The process is iterative.

By focusing on the UCD in the HCI scenario, user experience (UX) could be improved in the process. According to Norman (2013), UX is one of the primary factors in the success of a given product. Therefore, one of the most critical concerns in mobile application design is user experience (UX) consistency. The sensors, gestures, and location data play a dominant role in several mobile applications compared to traditional computer applications. The different styles and smaller displays of UI also significantly impact interaction design for mobile applications. A primary concern in designing mobile applications is that users usually desire to perform instantaneous and immersed actions instead of long-lasting sessions like those running on a desktop computer.

### 2.4 Agile

Agile development is a process that combines iterative with incremental, which focuses on people and interactions. Since the late 1990s, it has emerged and offered disciplined yet lightweight processes while emphasizing human effort and experience at the core of software development (Cockburn, 2002). The agile approach retains the rigor of engineering processes and best practices while helping stakeholders and software engineers build, deploy, and maintain complex software. According to the latest state of Agile

report (Digital.ai, 2021), there was significant growth in Agile adoption within software development teams, increasing from 37% in 2020 to 86% in 2021. Besides, 97% of survey companies employed agile fully or partially within their organization in 2021, compared to 84% in the first survey in 2007. Nowadays, Agile has become a significant software engineering discipline in research and industrial practice. Researchers and industrial practitioners have implemented several agile approaches, especially in-app design and software development. Some popular agile processes are listed below:

- Dynamic systems development methodology (Stapleton, 1997) focuses on the full project lifecycle, which adjusts time and resources to achieve practicality. This approach considers the need to react quickly throughout product development while incorporating the constraints often imposed by corporate cultures and processes.
- Scrum is a framework to address complex adaptive issues while creatively and productively delivering the highest value products. According to Schwaber and Beedle (2002), the Scrum approach supports flexibility, capacity, and productivity during an unstable setting and is the programmers' implementation technique, particularly in code development.
- Extreme Programming is a software development framework aiming to produce higher quality software, deal with fast delivery issues, and improve the quality of life for the development team. Its features typically comprise small iterations and quick reactions, so this framework is appropriate for the fast-changing business environment.
- Feature-driven development (FDD) highlights the planning and constructing stages, concentrating on the quality perspective. This approach is related to Scrum and focuses on the features or user stories. FDD allows more rapid development by leveraging pre-defined development standards and allows larger development teams to work on the project with correct observations by providing quick and concrete software delivery (Palmer & Felsing, 2001).
- Adaptive software development is the advancement of the rapid application development agile framework that enables teams to quickly and effectively adapt to market needs by evolving the products with lightweight planning and continuous learning (Highsmith, 2013). A progressive and unvaried improvement can solve the crisis of vast and complicated systems.

There are several benefits of applying agile in software development and engineering, such as more flexible methodology, transparency, clear ownership and accountability, and constant feedback to improve the output product. Recently, the integration of agile into the UCD approach has gained more attention from researchers and practitioners to promote user involvement and experience in mobile development (Garcia, da Silva & Silveira, 2019). However, communicating effectively between stakeholders, particularly on a daily basis, to build a shared understanding regarding the project context remains a challenge. This shared understanding among software engineering and HCI individuals becomes essential to the success of many agile projects for mobile development.

### 2.5 Summary and problem statement

In summary, D&D has brought countless hours of imagination and adventure to players around the world since it was first launched. In this fantasy role-playing game, players can create as many characters, items and environments as they like and communicate with other players. For this reason, D&D attracted a large

number of players and became the model for many online role-playing games that followed. Its success also provided a new door for business development and researchers to develop fast-flowing game designs and implementations with high performance and consumer satisfaction. Many technical studies on adapting characters to player interests have been implemented based on D&D (Faria et al., 2019, Martin et al., 2018). Therefore, we used D&D to conduct our study.

Compared to traditional software development on desktop computers, mobile software development is considerably different and faces many challenges, such as various operating platforms and device processor limits. However, the research gap lies in that most studies that have applied human-computer interaction to mobile system-based TRPGs (including D&D) have focused on a single mobile phone system such as Android or IOS. Today's technology has made it more difficult for developers to design an application that can run on multiple computing platforms. In order to deal with this issue, developers need to design the same application for different platforms. In addition, different companies use different development frameworks, leading to a variety of frameworks, which increases development and maintenance costs as well. It is, therefore, crucial to choose a framework that suits the requirements of different operating systems and expense management.

Therefore, the research question in this study is how to design a mobile game application that will increase the satisfaction of users of multi-role-playing games while avoiding the need for game companies to spend a lot of time designing a separate game application for each mobile system.

To address this research problem, we aim to create a suitable software replacement for the character sheet of the Dungeons and Dragons (D&D) Game by employing Xamarin, a leading cross-platform for mobile applications and Agile framework. This application attempts to improve Human-Computer Interaction and eventually enhance user experiences, and provide a solution for businesses, especially small and medium-sized ones, to flexible and fast design and build mobile apps that can be run on multiple platforms.

## 3. PROJECT DESIGN

The embedded applications based on mobile devices are different from PC ones due to the many unique characteristics of mobile devices. We have to consider the shortcoming of mobile phones when designing and implementing mobile apps to suit users' needs. This section describes the prototype of the new product design and implementation in mobile applications.

### 3.1 Description of the Problem

The official character sheet offers a reasonably suitable solution for creating and managing characters, with all the necessary information included and formatted appropriately. However, since it is a physical sheet of paper, the player has to calculate and fill out all of this information manually. For more lengthy data, players may decide to refer regularly to a separate document; typically, this is the player's handbook, though some players may produce a print-out sheet of miscellaneous character data.

This proves to be especially problematic, as this miscellaneous data most often includes the character's racial traits and class features. These describe the overwhelming majority of the gameplay mechanics associated with the character, including a range of individual actions, modifiers, and passive effects granted either at creation or at a given class level. By failing to provide a convenient solution for presenting this information to players, they may find themselves lacking a comprehensive understanding of the character. This leads to negatively impacts the game's overall experience.

### 3.2 Description of the Solution

As a potential solution to the described drawbacks of the physical character sheet, the application mentioned above could be designed and implemented. Our mobile application for the game is developed based on D&D game. This application would include a background source of in-game data, which can be collected to create a dashboard for a user-created D&D character. This data would be filtered and calculated based on the decisions made by the user during the character creation process. Where appropriate, this dashboard would provide a series of tools to allow users to update their characters, developed such that any updates must conform to the game's rules, as written in the official documentation. The name of this application would be Aboleth, named after a high-level monster from the game.

Aboleth would be best written as a mobile application to maximize availability and usability. Owing to D&D's highly social nature, the application would need to be one that could be referred to quickly and efficiently while engaged in a group conversation around a table or an online hangout. A mobile app provided the best possible solution to this problem and was chosen to implement this concept.

The development of this project used a loose variation of the Agile development pattern. In particular, it imitated the process of dividing the development into a series of isolated tasks, prioritizing that each period of development would produce a working piece of software, such that it could be presented as the final version at a moment's notice. For this reason, the design process was heavily deprioritized, focusing primarily on the design of individual features.

However, this does not mean that no top-level system designs were created. To suitably map out the relationship between each of the two deliverables and their data sources, an external system design was conceptualized using the C4 model (2021).

### 3.3 Overall System Design

The Aboleth system is split across three separate environments, as shown in Figure 1.

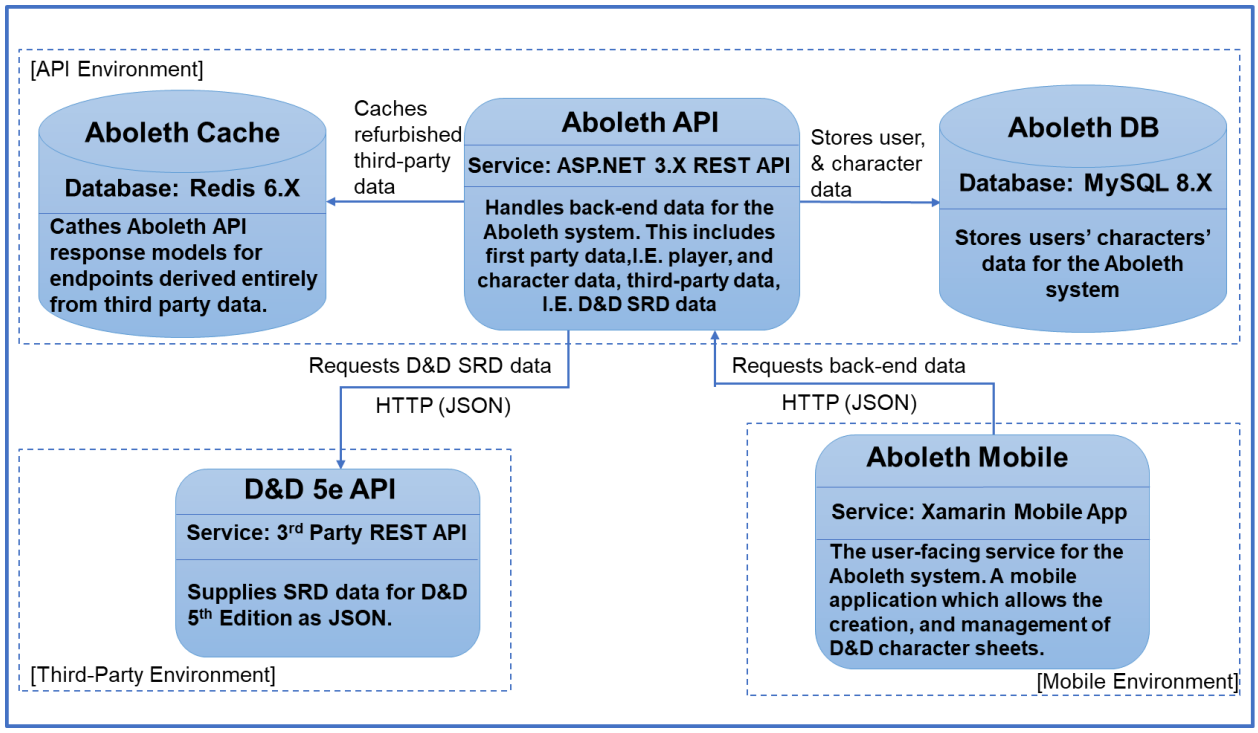


***Figure 1: Service map diagram for the Aboleth system***

Each of these three environments is separated in two different ways, firstly, by where they are hosted or run. The mobile environment refers to a mobile application run at will by individual users. In contrast, the API and third-party environments are hosted online by a suitable hosting service. The third-party environment is hosted and managed by other developers, having no exclusive relation to this project.

Secondly, each of the three environments was explicitly designed to only communicate with one other environment. Specifically, the mobile environment communicates only with the API environment, and the API environment communicates only with the third-party environment. This pattern is easier and more intuitive to implement and ensures that no single service has any more than a single service dependency. With added techniques to reduce this dependence as much as possible, a secure, reliable system could be implemented, with transient problems occurring as infrequently as possible.

This system design was ultimately decided upon to be the architecture of the Aboleth project. The model required very few variations, with the only previous iteration involving the mobile environment making direct connections to both the third-party and the API environments. This iteration was ultimately rejected, as the dependency between the third-party and API environment was far too intricate for this approach to be practical. It would be more time-efficient in the long term to pass the dependency forward rather than to integrate two dependencies separately in different parts of the system.

After the system design was finalized, it could be referred to regularly during the design process of individual features. It could be used as a visual aid in determining how each component would function and

how it would be populated with the necessary data. This information could then be used to flesh out the expected workload for both project deliverables.

### 3.4 Mobile Environment UI Design

Due to Agile's loose implementation for this project, the design process was heavily deprioritized on a lower level. For each feature, the design process would mostly be limited to a series of Computer Assisted Designs (or CADs) for features that would be developed. The thoroughness of these designs would vary greatly, depending on the pre-existing vision for the feature in question. CAD designs typically are used to provide a clearer picture of a component that is otherwise only very vaguely conceptualized.

An excellent example of this is the overall appearance of the character sheet itself. While the existing physical equivalent could be used as a reference to determine the character sheet's general contents, it was a lot more challenging to visualize how these contents would be aggregated together onto a single page of a mobile app. Closely imitating the physical character sheet would be highly unsuitable for a portable format, as it is specifically designed to fit an A-sized sheet of paper. Additionally, individual components of the sheet were redesigned to improve the formatting and the grouping of its contained data.

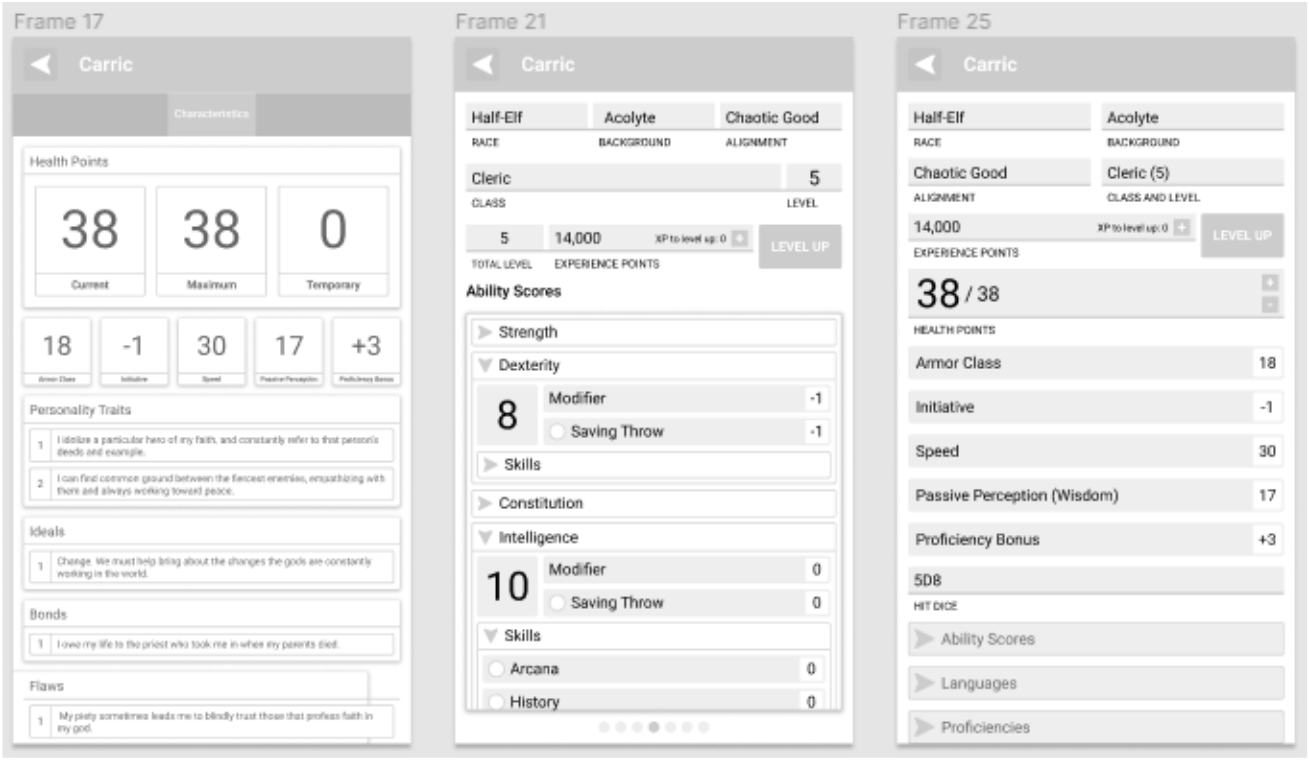


***Figure 2: Iterations of UI Design for the main page of the character sheet***

Several iterations of a design for the character sheet were created to solve this issue and updated regularly as alternative solutions presented themselves. A few of these design iterations can be seen in Figure 2. Figure 3, on the other hand, shows some similar designs for individual sections of the character sheet. These relate to the first proposed solution: for the character sheet to be presented as a series of tabs, splitting data into a series of appropriate categories, EG, "Ability Scores", or "Languages and Proficiencies". This design

was useful early into the development process for implementing early versions of the breakdowns alongside a corresponding step of the character creator.

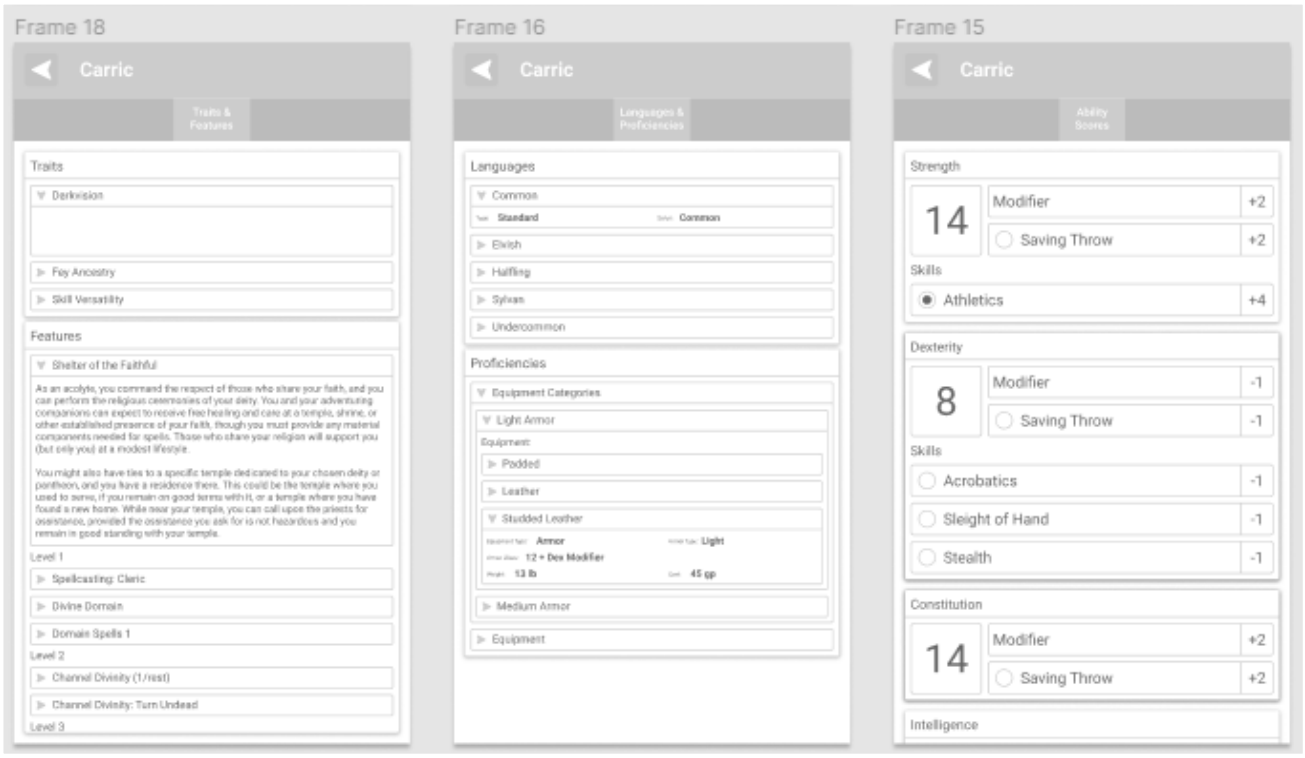


***Figure 3: UI Designs for inner breakdowns within the character sheet***

However, later on, in the development process, new components would be added that would be much better displayed as a part of a master template, which remains visible at all times, such as the health and experience indicators. ~~This oversight was addressed by attempting to implement precisely that~~. For the second iteration, core parts of the character sheet would be displayed at all times at the top of the page, followed by a swipe-able card system that allows users to scroll between different parts of the character sheet. This design would eventually be scrapped due to its implementation and replaced with the current solution, where components are hidden within accordion elements. Users can expand and contract at will to focus on selective aspects of their character as needed. Although this solution was less impressive visually than the previous iterations, it was more practical in its implementation, which was revealed in the design process.

A similar iterative process was used throughout the development process, and each new feature was implemented. It was prefaced with the completion of a suitable UI design, using the principles of Human-Computer Interaction to make informed decisions about the expected inputs and outputs. Appropriate considerations were also made for deciding how the application would communicate with the user. Although the displaying of outputs partially covers this, the app should also articulate how a user can and cannot interact with it at any given time. The implementation phase is made a lot easier by figuring this out during the design phase.

### 3.5 Implementation

Following the loose adaptation of agile development used for this project, the implementation of this project was completed on a task-by-task basis. This process consisted of an extensive number of tasks without necessarily sharing any consistent narrative between them; instead, each feature was developed based on the degree to which it was suitably visualized and designed. As a result, this section will not focus on detailing each of the specific tasks in order. Instead, it will describe the core infrastructure of each of the two deliverables, including descriptions of the primary tools and technologies used by each.

### 3.5.1 Aboleth API

The first of the two deliverables, Aboleth API, was generally simpler to implement, as each of its features could reliably conform to one of two infrastructural design patterns. Specifically, these two design patterns refer to the flow of third-party data from its source and the flow of first-party data from the API's relational database. A standard pipeline was created for each of these two patterns to reduce the overall workload of implementing new endpoints throughout the development process.

#### *A. Third-Party Data Collection*

For official D&D game data, the Aboleth system uses a publicly available REST API called the D&D 5e API. This API is not an official product of Wizards of the Coast, the legal owners of the game; therefore, the data available from this API is limited for copyright reasons. Fortunately, a generous subset of the data written in the player's handbook has been made publicly available online in the form of a Standard Reference Document (or SRD). This SRD is the source of truth for the data presented by this API and contains more than enough data to produce a suitable proof of concept for this project.

Direct connection to this API is handled entirely by Aboleth API. Many of the available endpoints have been proxied to provide Aboleth Mobile direct access to this data, with the response models modified better to suit the strongly typed architecture of C#. Further, the response models have expanded to include complete object definitions of internal properties such as a race or class's starting proficiencies, otherwise only referenced by ID, name, and URL.

#### *B. Connection to Third-Party API*

Aboleth API handles its connection to this third-party environment via a service class called APIClientService. The implementation of this service is straightforward, as the API it connects to is read-only and, therefore, only contains GET requests. As a result of this, only a single method needs to be implemented by this service class; the source code for this class can be seen in Figure 4.

```
2 references
public class APIClientService : IAPIClientService
{
    2 references
    private HttpClient HttpClient { get; }

    0 references
    public APIClientService(IHttpClientFactory httpClientFactory)
    {
        HttpClient = httpClientFactory.CreateClient("DnD5eAPI");
    }

    2 references
    public async Task<T> Get<T>(string url)
    {
        var httpResponseMessage = await HttpClient.GetAsync(url);

        if (httpResponseMessage.IsSuccessStatusCode)
        {
            var response = await httpResponseMessage.Content.ReadAsAsync<T>();

            return response;
        }

        return default;
    }
}
```

**Figure 4: Source code for the APIClientService in Aboleth API**

All communications with the external API are handled via the method Get, which uses a passed-in URL to perform an HTTP request on the third-party API and maps it onto a corresponding C# model declared a generic type. In cases where the API request fails to return a success status code, presumed to be the result of an external problem with the server, the default value of the passed in generic type is returned. This is done to prevent runtime errors from occurring due to the presumed external problems and is handled suitably in the next step of the pipeline.

### *C. Caching of Third-Party Data*

After collecting third-party data directly from its source, the next step is to convert the data into its modified, expanded form and store the modified data persistently in a Redis database instance. This is done using a method called GetCachedData, in a service class called DistributedCacheService; the source code for this method can be seen in Figure 5.

```
33 references
public async Task<TModel> GetCachedData<TDTO, TModel>(string url, Func<TDTO, Task<TModel>> converter)
{
    var cachedData = await DistributedCache.GetStringAsync(url);

    if (cachedData is null)
    {
        var dto = await APIClientService.Get<TDTO>(url);

        if (dto != null)
        {
            var model = await converter(dto);

            var modelJson = JsonConvert.SerializeObject(model);
            await DistributedCache.SetStringAsync(url, modelJson);

            return model;
        }

        return default;
    }

    return JsonConvert.DeserializeObject<TModel>(cachedData);
}
```

***Figure 5: Source code for the method GetCachedData within the class DistributedCacheService***

The logic behind this method is relatively simple; firstly, the cache is checked for existing data. The key under which it is stored in the URL for the API request (excluding the base) is passed in as a method argument for each cached document. If no cached data exists already, the data is collected directly from the third-party API. After this, if the API request is successful, the passed-in function "converter" is called to convert the DTO gathered externally into a domain model class. The form is typically the form that Aboleth API sends the data. This model is then stored in the cache instance and returned.

For performance reasons, the model cached within the Redis instance is the model returned by Aboleth API rather than the DTO collected externally. The primary reason is that the passed-in converter method can contain any number of calls to other service methods, collecting data from the same sources. Even in cases where the data from these other calls are pre-cached themselves, calling a constructor with several arguments, many of which are large lists of data, would result in slower performance than if the entire final model was cached and collected. A negative side effect of this is that the domain model class definitions will be changed, and the whole cache will need to be emptied and refilled to match the new model structure. However, this process is relatively quick and is therefore worth dealing with for its overwhelming benefits.

### *D. Proxying of Third-Party API Endpoints*

Aboleth API uses this infrastructure to create proxies for several endpoints available from the third-party API, albeit in a heavily customized form. A corresponding service class and controller are implemented for each group of proxied endpoints, using the previously outlined pipeline as a data source. Figure 6 shows an example of a service method created to proxy a third-party endpoint.

```
5 references
public async Task<Race> GetRace(string index)
{
    if (index != null)
    {
        var race = await DistributedCacheService.GetCachedData<RaceDTO, Race>($"api/races/{index}", GetRace);

        return race;
    }

    return null;
}
```

***Figure 6: Service method, which proxies the third-party endpoint to collect the definition for a single race***

Due to the robustness of the data pipeline created in the caching service, each service method that uses that pipeline can be conveniently simple, much like the method shown in Figure 6. In this case, most of the unique logic associated with this method has been contained within the converter function passed to the caching service, called GetRace. This function can be seen in Figure 7.

```
1 reference
private async Task<Race> GetRace(RaceDTO raceDTO)
{
    var abilityScores = await AbilityScoresService.GetAbilityScores();
    var languages = await LanguagesService.GetLanguages();
    var proficiencies = await ProficienciesService.GetProficiencies();
    var subraces = await SubracesService.GetSubraces();
    var traits = await TraitsService.GetTraits();

    var race = new Race(raceDTO, abilityScores, languages, proficiencies, subraces, traits);
    return race;
}
```

***Figure 7: Conversion method to convert a* RaceDTO *into a* Race *domain model***

This function is similarly straightforward in its implementation, albeit somewhat more involved. To create the model that this API returns for the proxied endpoint, references to other data sets, such as a race's proficiencies and traits, must be collected from their related services and aggregated together afterward in the output of the model constructor. This method, in particular, collects data from five other services, each of which may also collect data from any number of different services, possibly including one another.

A consequence of this implementation is that for more complex responses, such as this one, the first request made to the corresponding endpoint may have an anomalously slow response time due to the quantity of data it needs to collect from the external source. Exacerbating this problem is that for API calls that return a list of items, such as a list of races, the third-party API does not return complete object definitions for each item. Instead, it only produces a list of reduced references to each item. The consequence of this is that for each list item, the object definition must be collected and cached individually for service methods

such as the ones called in Figure 7. To fix the problem of slow-response time for the first call in the future, we could improve this method by having the third-party API return the full object definitions instead of collecting them one by one, repeating the same steps multiple times. Furthermore, resource utilization can be improved by limiting the number of players online on each server, giving higher-level players higher moderation rights, or requesting moderation rights from community administrators. Players are also encouraged to play in separate sessions.

Fortunately, this issue only persists for the first call; as data is cached persistently, subsequent calls to an endpoint have significantly quicker response times. This extends to endpoints corresponding to datasets used by the first call, as that data is cached separately. The result of this is that calling a small number of endpoints may result in an overwhelming majority of third-party data being cached, resulting in a significant performance increase throughout the API, eliminating the original problem almost entirely.

This overall marks the finishing point for the first of the two primary data flows implemented for Aboleth API. The second concerns the access and manipulation of first-party, user-created data; however, this also uses the third-party data flow to aggregate data based on a set of user-defined filters.

### *E. First-Party Data Management*

The first-party data flow is somewhat more complicated than the third-party one, primarily because it involves making a direct connection to a relational database. However, this process has been simplified as much as possible with suitable tools to streamline each step, from direct database access to integration with third-party data to the generation of appropriate user-facing data.

### *F. Relational Database Schema Access*

For direct access to the MySQL database schema, where user-created data is stored, Aboleth API uses Entity Framework, a popular framework for ASP.NET projects. For the most part, the use of this framework is typical, with each of the standard tools being used in a comparable way to other projects that use the framework. For each table in the database schema, a matching EF model class is defined, with attribute tags used to determine which of the values in the model are required. A static method named BuildEntity is declared for each model class, which performs the necessary setup for the table, including the definitions of primary and foreign keys. Figure 8 shows an example of this method for character, a central table in the database schema.

```
1 reference | Dan Lawrence, 25 days ago | 2 authors, 7 changes
internal static void BuildEntity(EntityTypeBuilder<CharacterEFModel> entity)
{
    entity.ToTable("Character");
    entity.Property(x => x.Id).HasColumnType("VARCHAR(36)");

    entity.HasMany(x => x.CharacterClasses)
        .WithOne(x => x.Character).HasForeignKey(x => x.CharacterId).OnDelete(DeleteBehavior.Cascade);

    entity.HasMany(x => x.TraitChoices)
        .WithOne(x => x.Character).HasForeignKey(x => x.CharacterId).OnDelete(DeleteBehavior.Cascade);

    entity.HasMany(x => x.AbilityScores)
        .WithOne(x => x.Character).HasForeignKey(x => x.CharacterId).OnDelete(DeleteBehavior.Cascade);

    entity.HasMany(x => x.Languages)
        .WithOne(x => x.Character).HasForeignKey(x => x.CharacterId).OnDelete(DeleteBehavior.Cascade);

    entity.HasMany(x => x.Proficiencies)
        .WithOne(x => x.Character).HasForeignKey(x => x.CharacterId).OnDelete(DeleteBehavior.Cascade);

    entity.HasMany(x => x.Equipment)
        .WithOne(x => x.Character).HasForeignKey(x => x.CharacterId).OnDelete(DeleteBehavior.Cascade);
}
```

**Figure 8: Implementation of the static method BuildEntity for the table Character**

Each of the tables declared via a corresponding EF model must be declared in a commonplace before accessing the data. This commonplace is the DB context for entity framework, which establishes a direct connection to a database specified by a passed-in connection string and generates SQL queries internally to map persistent data into the declared EF models. This DB context is added to a project via dependency injection. Figure 9 shows the DB context for Aboleth API.

```
16 references | Dan Lawrence, 25 days ago | 2 authors, 9 changes
public class AbolethDbContext : DbContext
{
    4 references | Dan Lawrence, 91 days ago | 1 author, 1 change
    public DbSet<AccountEFModel> Accounts { get; set; }
    6 references | Dan Lawrence, 68 days ago | 1 author, 1 change
    public DbSet<CharacterEFModel> Characters { get; set; }

    0 references | Dan Lawrence, 68 days ago | 1 author, 4 changes
    public AbolethDbContext(DbContextOptions<AbolethDbContext> options, IHostingEnvironment environment) : base(options)
    {
        if (environment.IsDevelopment())
        {
            Database.Migrate();
        }
    }

    0 references | Dan Lawrence, 25 days ago | 2 authors, 6 changes
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<AccountEFModel>(AccountEFModel.BuildEntity);

        modelBuilder.Entity<AbilityScoreEFModel>(AbilityScoreEFModel.BuildEntity);
        modelBuilder.Entity<CharacterEFModel>(CharacterEFModel.BuildEntity);
        modelBuilder.Entity<CharacterClassEFModel>(CharacterClassEFModel.BuildEntity);
        modelBuilder.Entity<EquipmentEFModel>(EquipmentEFModel.BuildEntity);
        modelBuilder.Entity<LanguageEFModel>(LanguageEFModel.BuildEntity);
        modelBuilder.Entity<ProficiencyEFModel>(ProficiencyEFModel.BuildEntity);
        modelBuilder.Entity<TraitChoiceEFModel>(TraitChoiceEFModel.BuildEntity);
    }
}
```

***Figure 9: Source code for Aboleth's DB context***

This implementation of a DB context is relatively lightweight; each table's implementation of BuildEntity is called whenever the database is migrated. This happens each time the API is launched in the development environment. The properties "Accounts" and "Characters" are how other services access database data; the DbSet type contains a series of methods resembling those of C #'s LINQ package, used to generate appropriate SQL queries for mapping the data into the desired output format. For the sake of simplicity, only these two tables from the schema can be accessed directly in this fashion. The data from other tables are simply included in the output model for the Character table, as there is no current use case for these tables to be accessed independently.

The DB context is used in only one place in Aboleth API. Specifically, it is used in the service of the characters, which integrates this data with its corresponding third-party data. This produces a comprehensive character definition that can be used to build a suitable user-facing view model. Aboleth Mobile then uses this view model to populate the character sheet page.

### *G. Integration of First and Third-Party Data*

For the MySQL database schema, its contained data is as minimal as possible. For each table, game data is referenced via the corresponding index value used by the third-party API. This creates an illusion of an entity relationship between the tables in the first and third-party datasets. The Characters Service contributes to this illusion by integrating the two datasets as business logic. This is done with the support of dependency injection; by collecting scoped instances of third-party data services, a simple algorithm was written to

collect all the necessary data and aggregate it together in a domain model class constructor. This can be seen in Figures 10 and 11:

```
7 references | Dan Lawrence, 25 days ago | 2 authors, 13 changes
public async Task<Character> GetCharacter(Guid id)
{
    var characterEFModel = await DbContext.Characters.CharacterIncludes().FirstOrDefaultAsync(x => x.Id == id);

    if (characterEFModel != null)
    {
        var abilityScores = await AbilityScoresService.GetAbilityScores();
        var alignment = await AlignmentsService.GetAlignment(characterEFModel.Alignment);
        var background = await BackgroundsService.GetBackground(characterEFModel.Background);
        var classes = await ClassesService.GetClasses();
        var equipment = await EquipmentService.GetEquipment();
        var languages = await LanguagesService.GetLanguages();
        var proficiencies = await ProficienciesService.GetProficiencies();
        var race = await RacesService.GetRace(characterEFModel.Race);
        var subrace = await SubracesService.GetSubrace(characterEFModel.Subrace);

        var character = new Character(
            characterEFModel, alignment, background, classes, race, subrace, abilityScores, languages, proficiencies, equipment);

        return character;
    }

    return null;
}
```

***Figure 10: Method to aggregate first and third-party data for a single character***

```
2 references | Dan Lawrence, 25 days ago | 1 author, 1 change
public Character(
    CharacterEFModel characterEFModel,
    Alignment alignment,
    Background background,
    List<Class> classes,
    Race race,
    Subrace subrace,
    List<AbilityScore> abilityScores,
    List<Language> languages,
    List<Proficiency> proficiencies,
    List<Equipment.Equipment> equipment)
{
    Id = characterEFModel.Id;
    Created = characterEFModel.Created;
    Name = characterEFModel.Name;
    CurrentHealth = characterEFModel.CurrentHealth;
    MaxHealth = characterEFModel.MaxHealth;
    Alignment = alignment;
    ExperiencePoints = characterEFModel.ExperiencePoints;
    Race = race;
    Subrace = subrace;
    TraitChoices = characterEFModel.TraitChoices.Select(x => new TraitChoice(x)).ToList();
    Background = background;
    Classes = characterEFModel.CharacterClasses.Select(x => new CharacterClass(x, classes)).ToList();
    AbilityScores = characterEFModel.AbilityScores.Select(x => new CharacterAbilityScore(x, abilityScores)).ToList();
    Languages = characterEFModel.Languages.Select(x => languages.First(y => y.Index == x.Language)).ToList();
    Proficiencies = characterEFModel.Proficiencies.Select(x => proficiencies.First(y => y.Index == x.Proficiency)).ToList();
    Equipment = characterEFModel.Equipment.Select(x => new CharacterEquipment(x, equipment)).ToList();
}
```

***Figure 11: Constructor for the domain class Character***

As these two figures show, first and third-party data are included alongside one another within the domain model and the specific contents of this data are dictated by the former. This domain model is used

throughout the first-party data flow for Aboleth API; for GET operations, this model is collected from or simplified to produce a suitable model for Aboleth Mobile to use. For procedures that manipulate data, such as creating or updating characters, the preferred method is typically to modify the EF model directly. However, after such operations have been executed successfully, the domain model is then collected and simplified to return the newly created or updated character.

The first and third-party data flow create a robust API, which serves as a suitable source of truth for the second deliverable in this project, the mobile application. While the mobile app does not utilize all the available endpoints, its use is crucial to its overall function.

### 3.5.2 Aboleth Mobile

The second of the two deliverables for this project, Aboleth Mobile, is significantly more complex than the first. It contains a series of user interfaces, which need to be individually designed and implemented, and data returned by the API powers each part of the app. This means that a robust method of API communication must be implemented so that its data can be easily accessed and used wherever needed.

***A. Connection to Aboleth API***

Aboleth Mobile handles its connections to Aboleth API using a series of simple service classes, each inserted into Xamarin's implementation of dependency injection. Xamarin implements a simple form of dependency injection, wherein an interface and implementing class can be registered and collected using static methods for a class named "dependency service". This implementation is much more limited than ASP.NET, as registered services cannot specify specific construction rules. Instead, each registered service must define a suitable parameterless constructor. However, this constructor can call the dependency service to collect other service instances; this is the strategy used to allow services to communicate with one another.

To handle direct connections to Aboleth API, a base service named "API client service" was implemented and injected. This service contains a private HTTP client, with methods implemented to perform each of the REST operations available for the API. GET operations to use an in-memory cache to cache static data locally, improving performance when navigating the character creator and level-up page for enterprise mobile applications and Metaverse. The method implements an optional parameter to skip the cache, to be used for data sets that are likely to change frequently, such as the data returned from the service of the characters. Figure 12 shows the Get method for the API client service:

```
14 references | Dan Lawrence, 9 days ago | 1 author, 1 change
public async Task<T> Get<T>(string url, bool skipCache = false) where T : class
{
    if (!skipCache && MemoryCache.TryGetValue<T>(url, out T cached))
    {
        return cached;
    }

    var httpResponseMessage = await HttpClient.GetAsync(url);

    if (httpResponseMessage.IsSuccessStatusCode)
    {
        var response = await httpResponseMessage.Content.ReadAsAsync<T>();

        if (!skipCache)
        {
            MemoryCache.Set(url, response);
        }

        return response;
    }

    throw new Exception("Connection failed.");
}
```

***Figure 12: Source code for the "Get" method within the API Client Service***

Each service implemented in Aboleth Mobile corresponds to a single controller from Aboleth API. For example, the service of the characters corresponds to the characters controller, which handles operations related to user-created characters. Only the characters service implements REST operations other than GET, including basic CRUD operations.

Injected services exist in the background of a Xamarin forms application, independently of the app's current page or state. Because of this, app configuration, including page navigation and standard stylings, can be configured entirely separately.

***B. App Shell and Page Navigation***

To handle hierarchy and navigation between pages, Xamarin Forms offers an app shell feature, where these can be defined either via XAML or by C# code. For Aboleth, page navigation is simple, with only a small number of pages to navigate between. Furthermore, none of the more advanced app shell features, such as a tab-bar or slide-out menu, are used, meaning that its use was primarily limited to defining navigation between pages via C# code. The other primary purpose of the app shell is to configure the base page, which is accomplished with a single XAML tag.

For Aboleth, two separate app shells are defined. The first, the primary shell, defines the page flows for the application's content, while the second establishes the page flow between the login and account creation

page. Login and logout are handled by simply switching the active app shell at runtime. A background check is performed on startup to see if a valid login token is stored; this is used to determine whether a user is logged in already and decide which app shell to use initially.

The app shell does not define the contents or behavior of the pages declared within it, merely their hierarchy. For these, a more fundamental technology of Xamarin is used instead.

### *C. Implementations of MVVM Pattern*

The primary architecture for each of the views and pages contained in a Xamarin application is the Model-View-View-Model pattern (or MVVM). How this works is that for each view class created, a corresponding view-model class is implemented, which holds all the values the view will display. This is done via a process called "binding", wherein a view-model instance is assigned as the "binding context" of the view. Each value from the view model is declared in the view's XAML as the value of a tag property, in the form {Binding [Property Name]}.

This pattern is used throughout Aboleth in various ways. The most straightforward implementation used by the home page involves the binding context class simply being declared in the view's XAML code as a tag. This declaration can be seen in Figure 13.

```
<BindableObject.BindingContext>
    <home:HomePageViewModel />
</BindableObject.BindingContext>
```

***Figure 13: Declaration in XAML for the Binding Context (view-model) of the home page***

This implementation is preferred when no values need to be passed into the view model from another view/view model. The view contains no unique behavior or custom bindable properties that need to be passed into it. If the opposite is the case, then an alternative solution would be to declare the view model as an attribute within the corresponding C# class created alongside the XAML file for a view. This attribute will then be instantiated within the constructor. After that, the attribute can apply any behaviors or assign any custom bindable properties directly to the view model at runtime. A simple example of this can be seen in Figure 14.

```
[XamlCompilation(XamlCompilationOptions.Compile)]
5 references
public partial class CharacterSheetPage : ContentPage
{
    private readonly CharacterSheetPageViewModel viewModel;

    0 references
    public CharacterSheetPage()
    {
        InitializeComponent();

        BindingContext = viewModel = new CharacterSheetPageViewModel();
    }

    1 reference
    protected override void OnAppearing()
    {
        base.OnAppearing();
        viewModel.Load();
        experiencePointsView.Load();
        healthPointsIndicatorView.Load();
    }
}
```

***Figure 14: C# code for the character sheet page.***

***Note how the view model is declared as an attribute and instantiated within the constructor***

In this example, the view model is called whenever the character sheet page appears to force a reload, recollecting the character data from Aboleth API and assigning it to the corresponding values for the view model. This is necessary primarily because the level-up page makes direct changes to a character's data. The view model's "Load" method is otherwise only called when the character id is first passed into it. This method of declaring the view model is by far the most common throughout the app due to the high quantity of unique behaviors and custom binding properties.

In some cases, direct communication between child and parent view models will need to be made. For this scenario, neither of the two above solutions is sufficient, and a third method of declaring a binding context should be used. Fortunately, the binding context for a view is a bindable property itself. For cases where a parent view model directly contains a child view model or a list of child view models as a property, the child view models can simply be declared in the XAML for the parent's corresponding view. Figure 15 shows this being done for the "determine ability scores" step in the character creator:

```
<StackLayout BindableLayout.ItemsSource="{Binding AbilityScoreInputViewModels}" Spacing="0">
    <BindableLayout.ItemTemplate>
        <DataTemplate>
            <determineabilityscores:AbilityScoreInputView BindingContext="{Binding .}" />
        </DataTemplate>
    </BindableLayout.ItemTemplate>
</StackLayout>
```

***Figure 15: Snippet of XAML code from the* AbilityScoreSelectorView*, the main view of the "determine ability scores" step from the character creator***

In this example, the BindableLayout.ItemsSource property is used to add a templated list of views to a parent stack layout element, using a passed-in list of items as a binding context. In this case, the template has been defined elsewhere in a class called AbilityScoreInputView. The property BindingContext={Binding .} denotes that the element's view model is being assigned as the current item in the list AbilityScoreInputViewModels, a list of view models used to hold the input and output values for editing a single ability score.

This method of declaring a view model is used less commonly than the one previously mentioned; however, it still has many use cases throughout Aboleth. For the example of the ability score selector, users are typically given a set of six scores to choose from; these are either rolled by the player or are a simple set of 6 "preset" values. When one of these six choices is picked for a particular score, the parent is called to remove this option from the other five ability score inputs and add back the previously selected score (if applicable). The parent (i.e., the ability score selector view), and the child (i.e., the ability score input views), must have their view models communicate directly. An update to a value in the child must trigger a method in the parent, which will trigger a varying number of methods in each child. Similar logic is found later in the character creator, specifically, the languages and proficiencies selectors in the "Describe your Character" step. This structure is put into place to prevent a user from selecting the same language or proficiency more than once, and this same method of declaring the binding context for child views is used in those cases.

The wide range of applications for the MVVM model offers great flexibility for developing Xamarin applications. This is especially useful for integrating with other features provided by the framework, such as the ability to create new properties to bind with.

***D. Custom Binding Properties for Created Views***

To further expand on its flexibility, Xamarin offers developers the ability to add custom bindable properties to a custom view, fully controlling how this bindable property is used. This feature is used several times throughout Aboleth, particularly for its more generic elements, such as the LabelledNumberView, used in multiple places to display numerical values for a character. Figure 16 shows the C# code for this view's custom binding properties.

```
6 references
public partial class LabelledNumberView : ContentView
{
    public static readonly BindableProperty LabelProperty = BindableProperty.Create(
        propertyName: "Label",
        returnType: typeof(string),
        declaringType: typeof(LabelledNumberView),
        defaultValue: "");

    public static readonly BindableProperty NumberProperty = BindableProperty.Create(
        propertyName: "Number",
        returnType: typeof(string),
        declaringType: typeof(LabelledNumberView),
        defaultValue: "");

    2 references
    public string Label
    {
        get => (string)GetValue(LabelProperty);
        set => SetValue(LabelProperty, value);
    }

    2 references
    public string Number
    {
        get => (string)GetValue(NumberProperty);
        set => SetValue(NumberProperty, value);
    }

    2 references
    public LabelledNumberView()
    {
        InitializeComponent();
        Content.BindingContext = this;
    }
}
```

***Figure 16: Custom binding property declarations for the* LabelledNumberView *class***

Each custom binding property follows a very similar pattern. Firstly, a static instance of the object BindableProperty is declared and instantiated via "BindableProperty.Create". This method requires the developer to enter the property name, type, and class type that declares it. Several optional parameters are also included in this method, such as a default value, default binding mode, and a function to be invoked whenever the property's value is changed. In Figure 15, none of these optional parameters is used meaningfully; however, in other parts of the app, the "PropertyChanged" function is often declared, in most cases updating the corresponding value in a view model. No separate view model is declared in Figure 16; instead, the C# class uses itself as the view model, with a scoped property declared for each BindableProperty instance. These properties are referenced directly when binding custom values in the XAML of a view; the BindableProperty instance is used only to signal to the UI that the value needs to be read.

```
<breakdowns:LabelledNumberView Grid.Row="5" Grid.ColumnSpan="2" Label="Armor Class" Number="{Binding ArmorClass}" />
<breakdowns:LabelledNumberView Grid.Row="6" Grid.ColumnSpan="2" Label="Initiative" Number="{Binding Initiative}" />
<breakdowns:LabelledNumberView Grid.Row="7" Grid.ColumnSpan="2" Label="Speed" Number="{Binding Speed}" />
```

***Figure 17: Some examples of the custom bindable properties for the* LabelledNumberView *being given values on the character sheet page***

This feature is the last of the standard tools offered by Xamarin that was used for this project. These tools are used throughout the app to power each of its features and bar one specific case, and they prove to be sufficient for this task. The one exception to this is a feature seemingly not offered by Xamarin, with a customized solution being used instead.

***E. Step-by-Step Flow for Character Creator and Level Up Page***

The feature in question regards the implementation of both the character creator and the level-up page, as both implement a step-by-step page flow. A series of input views are presented in turn onto a single template page, with shared elements to indicate progress and standard infrastructure to allow inputted data to be passed between steps.

At its most fundamental level, the step-by-step page flow is accomplished using a bound property within the view model, which references an instance of the view class for the current step. This view is bound onto the base page via its XAML code. A dictionary of UI elements is stored within the view model from which the view binding is collected and assigned. For each entry, the key is the title for the corresponding step, displayed on the top of the screen. The current step index is also assigned to decide which view and title to display; this value is also used to calculate the progress on the progress bar for the character creator page for enterprise mobile applications and Metaverse. This index is decremented and incremented by two buttons on the bottom of the page: the "Previous" and "Next" buttons.

When another step is navigated, various operations are performed in the background. Every time a new step is rendered, a background operation is executed to load the data necessary to power that step. Aboleth API is invariably the immediate source of truth for this data. In some cases, the data displayed in a later step may vary based on the input given in an earlier step. This is accommodated for using a model containing minimized representations of the user's inputs stored within the character creator's view model. This model is passed to each implementation of the data loading operation to be used as needed.

Saved user inputs are updated whenever the user navigates to the next step. Each step contains a second operation that appends the results of its user inputs to the overall model; this operation is invoked whenever the "Next" button is pressed. Each implementation of this operation is entrusted to perform its validation. If it fails, a specific type of exception with a suitable message is sent back to the character sheet view model and handled accordingly. This process is also designed to address other unexpected exceptions, although these are made explicitly distinct from those produced via validation failures.

After the final step is completed successfully, each page, both the character creator and the level-up page, make a corresponding call to Aboleth API, requesting the relevant change by passing up the model of user inputs they previously used to power themselves. Upon successful execution of that operation, navigation

is automatically reverted to the page from which the step-by-step page was opened; for the character creator, this is the home page, and the level-up page, the character sheet page.

### 3.5.3 Transferable Skills

From the development of the Aboleth system, many reflections and discussions can be had about the range of transferable skills learned from it. From both this project's design and implementation process, several opportunities to practice several industry skills and technologies were offered. Many of these skills were already known to different degrees; however, new techniques could be discovered and refined by testing them in a brand-new scenario. This results in creating higher quality software, both in Aboleth and in any projects undertaken in the future. A few of these skills have been identified for individual discussion.

#### *A. Mobile Development using Xamarin*

The first of the skills identified in this discussion is developing mobile applications, specifically in the Xamarin framework. No previous experience had been had with Xamarin, meaning that the project outlined in this report was partly a first impression of the framework, implementing an example project to identify its potential. To this end, Xamarin unfortunately compared poorly to other, commonly available mobile development frameworks. The available tools described in the implementation were minimal. A lot of the app's features required complex, bespoke logic, which often clashed with the inbuilt tools and resulted in a lot of untidy code throughout the project.

Another significant limitation was the available range of widgets provided with Xamarin Forms. Many standard widgets, including HTML and almost any other UI development framework, were notably missing from this one; an excellent example of this is the number picker, incrementing and decrementing a numerical value. This omission is especially problematic for Aboleth, as such a widget was explicitly required for several features, such as the ability score selector on the character creator. A custom element was made to solve this, but due to time limitations, this element was of a fixed size, and its custom binding properties needed to be proxied to every view that used it. With a greater amount of development time, these issues could have been corrected; however, it remains the case that the problem did not need to exist in the first place, as Xamarin could and should have included such an element as standard.

Despite the issues experienced while working with Xamarin, the overall experience of mobile development was mostly preserved. Both the design and implementation process required considering how each view could be constrained to a mobile phone screen size while not sacrificing the overall experience. Another transferable skill was invoked to help accomplish this.

#### *B. Human-Computer Interaction*

Of the transferable skills learned during the development of this project, perhaps the most significant is the discipline of human-computer interaction (HCI), a field of study which focuses on the interaction between humans and computers. The study has similarities to the practice of user experience or UX design; the primary difference is that the former has a more academic focus, whereas the latter is more industry-focused.

The relationship between this field of study and the development of Aboleth is fundamental; the primary objectives for this project involved analyzing how a player interacts with a paper character sheet in a game of D&D and designing a computer-based equivalent for these interactions. For each of these, the design

objective was to provide the most intuitive user interaction possible by identifying the necessary inputs and outputs and finding the most appropriate UI elements for each.

A simple example of this is the first step of the character creator, where users are prompted to select a race for their character from a pre-created list. It was decided that the best UI element to make this input would be a select box containing the name of each race. When an option is selected, users should see the details of the chosen race to see the consequences of their choice. It was identified that a race's details are presented as a set of racial traits, with a title and description. As a result, they could be best presented to users as a scrolling list of simple cards, with the title in the larger text at the top and the description in smaller text at the bottom. The implementation of this page can be seen in Figure 18.

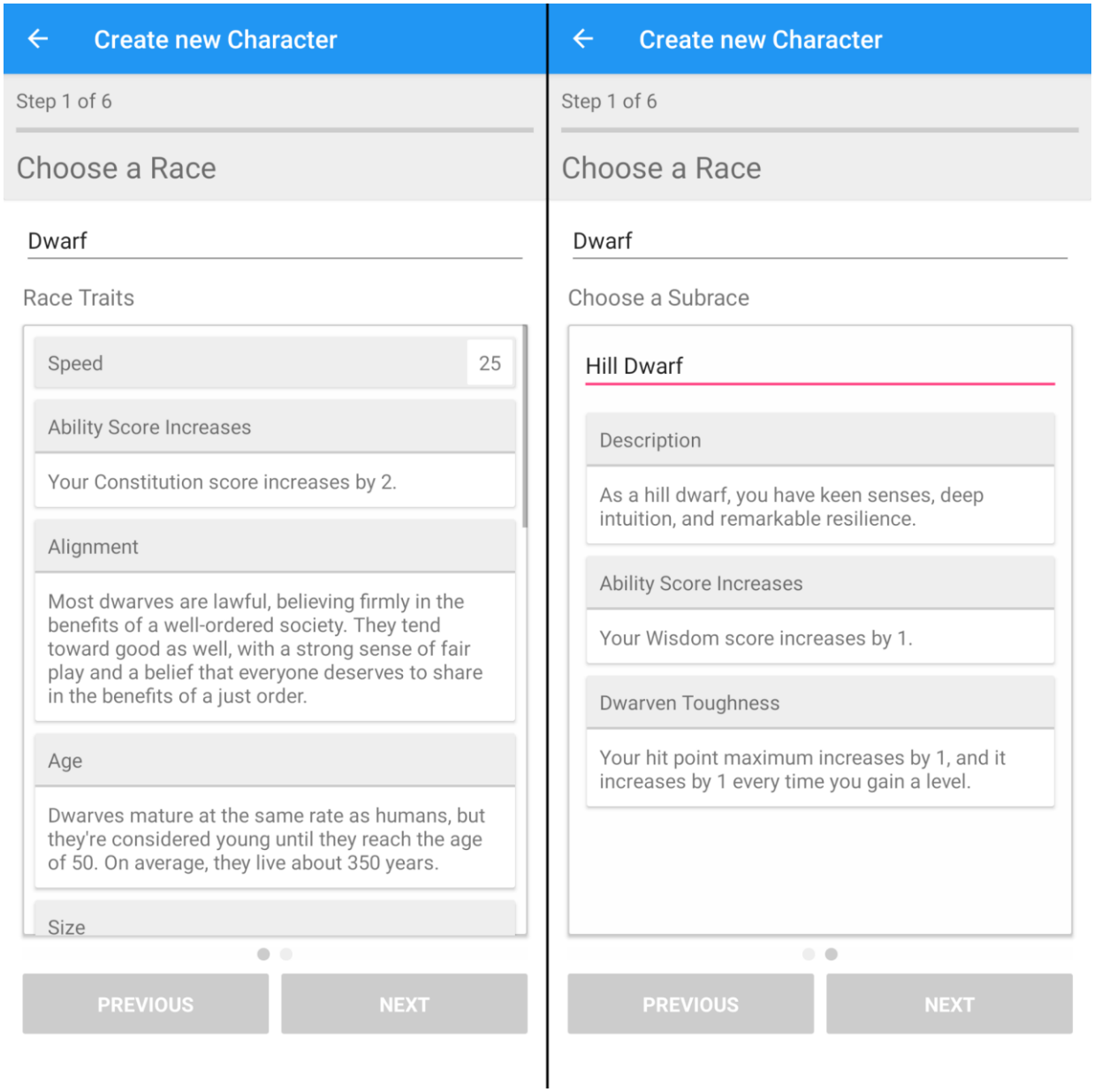


***Figure 18: Implementation of the "Choose a Race" step for the character creator after selecting a race***

HCI principles were used in other application areas to integrate inputs and outputs and create a highly responsive user interface with regular immediate feedback for the consequences of a user's input. An excellent example of this is the fourth step of the character creator, which prompts users to assign their ability scores using one of three methods. A standard user interface is used and modified accordingly for each of these methods, presenting suitable input widgets for each of the six ability scores and a series of boxes displaying the calculated outputs. The result of this creates a highly tactile user interface, allowing users to experiment freely with different combinations of ability scores to find the best possible balance, depending on the kind of character being created.

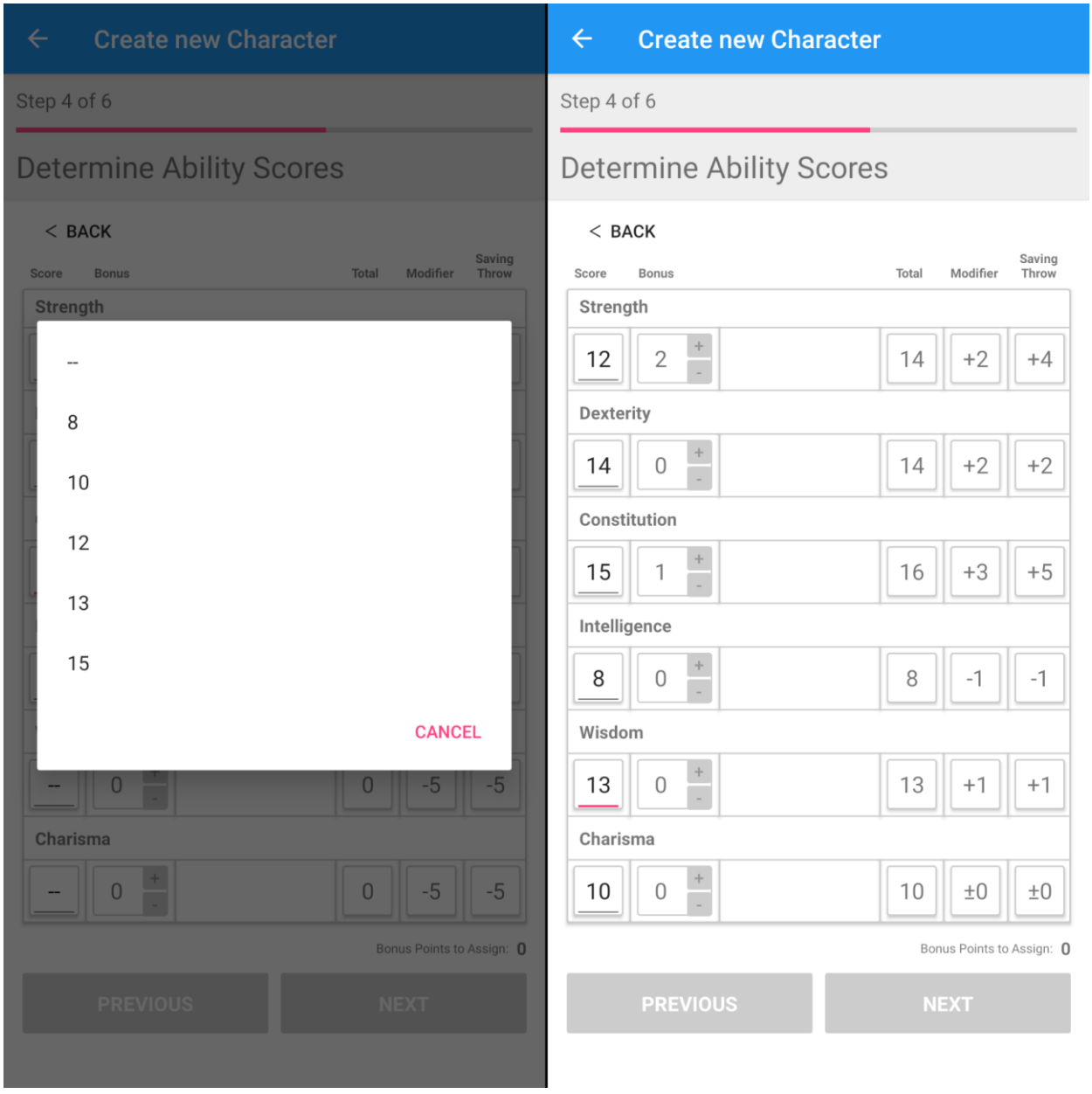


***Figure 19: Main page of the "Determine Ability Scores" step for the character creator. We can use preset scores***

Another area of concern when navigating throughout the application is handling loading states. HCI principles were once again used to create suitable methods of communication to users as to when a user interface is in an intractable state and when it is in a loading state. Fortunately, the Xamarin framework includes an in-built activity indicator, which can be quickly rendered on top of a view and activated on-demand using a bound Boolean variable. This bound variable could be used in other places to prevent user interference by deactivating input widgets. This is especially useful when the input widgets are populated with values collected from the API, preventing users from interacting with an unprepared user widget, such as an empty select box.

In more extreme cases, it might be the case that the user interface is partially or wholly generated using externally collected data. An example of this is the character sheet itself. In such cases, the loading state will hide the view entirely, instead only displaying the activity indicator until the loading process is complete. This is a common technique across various UI-based applications. Users should be fully aware of the circumstances and wait patiently for the loading process to complete, rather than assume that the application has malfunctioned and reacted drastically. In the character creator and level-up process, the loading process for navigating between steps will not display the next page until it has finished loading. This is especially useful for the level-up process, as it contains a step that only needs to be used once every four levels. Therefore, by not rendering the next step until it is completed, the logic for skipping that step can be executed without presenting anything unusual to users.

Unfortunately, the use of the HCI discipline for this project was underwhelming, with only a basic consideration made for how to make suitable UI designs rather than using a more academic, research-based approach. The UI design suffers in several areas: the final product, while well suited to more experienced players, is often unclear to more unfamiliar users as to its general purpose. The character creator's demonstration to audiences with no prior history with D&D shows that the app can easily be misinterpreted as a video game, particularly when the screen to choose a class is shown. This could have potentially been mitigated, if not prevented entirely, with a more robust and more research-based approach toward HCI, focusing on better communication with the user regarding the character creation process and the consequences of each user input.

### *C. Integrated Systems Design and Implementation*

As outlined in the system design, the development of Aboleth involves multiple software systems communicating with one another asynchronously. As a result of this, the development of Aboleth required an effective use of integrated systems development to function effectively and reliably.

The approach taken for this project was very effective; communications with other services are standardized using service-based data pipelines, with appropriate technologies used to cache incoming data. This loosens the dependence between services, as required data is more likely to still be available in the case of any failures or downtime suffered by the providing service. Both project deliverables were further written to be suitably transparent. If a failure occurs while connecting to another service, the client handles it appropriately while also communicating that the connection failed.

The development of this skill was further expanded upon by the inclusion of the third-party environment. By including a service not explicitly written for the project, secure and effective service integration methods became a significant priority to ensure that the third-party service was used reasonably. The use of this service originally inspired the decision to use caching. Another critical difference is the lack of control over the function of a third-party service; when developing Aboleth API, total power is given over the endpoints to include and the format in which they return their data. This luxury is not provided when using a publicly available API, as those decisions have been made beforehand, leaving users to adapt to their given tools.

However, the API used for the Aboleth system was a special case since it is an open-source project, with an open invitation to suggest contributions. This opened a dynamic opportunity to learn a new transferable skill, which was taken advantage of, to supply necessary data for Aboleth, which would otherwise not be available.

## 4. DISCUSSION

### 4.1 Review of the Product

Overall, the final quality of the product developed for this project is, unfortunately, very mixed. While there are many well-implemented features and back-end code to support them, there also exists an equal number of bad practices, roughly written features, and completely missing features. The reasons are partly due to time constraints and project management.

Perhaps the strongest aspect of the project overall is the implementation of the API. This is partly due to the simpler of the two deliverables by a considerable margin. However, even as simple as the API turned out to be, its overall quality was greatly affected by excellent design practices and good use of the tools offered by the frameworks it uses. Dependency injection, for example, is used extensively and intuitively. The result is a modular project that can be expanded upon quickly and easily, as existing services and clients can be collected and accessed with little effort. Additionally, the excellent use of a third-party service, including the caching system, and the standardized pipeline for converting a third-party model into a first-party response, creates a reliable and intuitive API capable of operating even without its primary dependency, at least temporarily.

On the other hand, the mobile application is much more problematic. This project suffered most from the bad practices and missing features for a variety of reasons. This application does have some redeeming qualities, however. For example, while missing several options available on the paper character sheet, particularly non-gameplay affecting character descriptions such as their personality and appearance, the character creator for Metaverse closely matches the original vision of a step-by-step page scrolling form. Each feature implemented is as accurate to the official documentation rules as possible. An exceptionally well-implemented part of the creator is the ability of the score selector, which presents users with three different methods. Each option leads to a form that provides all the necessary feedback about the consequences of the user's inputs, producing a pleasingly tactile experience as a result. Other parts of the application offer a similar experience to a lesser extent.

The primary weaknesses of Aboleth Mobile relate to its implementation. The limited toolset provided by Xamarin meant that many of the features developed for the project required a customized approach, leading to an undisciplined mix of framework code and standard C# code, which is challenging to navigate and understand. This further affected the amount of time each feature took to implement and made many applications look very basic visually. Moreover, there is little available time to provide suitable polish to new features after the initial implementation.

A secondary consequence of this limitation is the abundance of missing features initially planned for the process. The most egregious of these is the lack of an equipment breakdown on the character sheet. The character creator for Metaverse offers a step to choose equipment based on the selected class and background, so not providing a breakdown of these choices defeats the purpose of making a choice in the first place. Furthermore, the equipment breakdown is a crucial part of a character sheet containing the character's weapons and armor, two necessary pieces of information for players to have when engaging in combat. The absence of this feature makes the application significantly less usable as a direct result.

The inevitable consequence of these problems is that the application in its current form is not in an appropriate state to be distributed publicly. Instead, it serves only as a proof of concept, particularly for the character creator for Metaverse, which shows the most promise of any part of the app. The application could be worked on further in the future, with the eventual hope that it could be brought to a releasable standard. A suitable plan will need to be devised to accomplish this, to figure out the next steps of development for the Aboleth project.

### 4.2 Potential Next Steps

Considering the current status of Aboleth as a system, the first possible step to make for its continued development would be to tie up any loose ends remaining on the API. The relatively high quality of its implementation means that this should not be too difficult of a task. The focus will be given to ensuring that the character definitions are feature complete, including the options and filters available for D&D, but not implemented anywhere for this project. An excellent example of this is spelling, a crucial mechanic for several classes in the game. Once this is done to a suitable degree, the API can be feature complete, barring any expansions that may be made in the future.

As for the mobile application, it is more likely than not that the product produced for this project will be officially declared an experimental version and not developed any further. Instead, another more flexible framework will be found, and the experimental version will be used as an inspiration to help decrease the development time. While the overall quality of the implementation for Aboleth Mobile is mixed, many of the ideas created have been worth expanding upon and may be realized better with a different choice of framework. An attractive option could be the React framework for JavaScript. Its direct use of HTML and CSS offers a great deal of flexibility over a proprietary XML system such as the one Xamarin uses.

A personal objective for this project was to create an application worthy of being worked on beyond the scope of this project. If not necessarily high quality, it would demonstrate that the product showed enough potential to be said to be a successful project. The brief plan conceptualized here shows that this personal objective was at least partially completed, leading to the conclusion that despite its abundance of problems,

it can be said with confidence that Aboleth was, at the very least, a partially successful project. It creates a suitable proof-of-concept with plenty of opportunities for refactoring and expansion in the future.

### 4.2 Evaluation

Following the methods used to assess game design in the works of Dhillon et al. (2011), Helms et al. (2015), Morschheuser et al.(2018), and Schöbel et al.(2020), open-ended interviews were also conducted to evaluate the mobile D&D character creator proposed in this study to develop a holistic perspective on the subject matter. The aim of the evaluation is to investigate whether the proposed mobile D&D character creator meets its pre-defined objectives. The interview questions consist of 1) What do you think our proposed work can be relevant to Metaverse, and if the user interface is a focus, how can we improve on this? 2) In your opinion, how can other similar games, or similar strategy games, improve on its user interface and gamer involvement for Metaverse? 3) Please provide any other constructive feedback (not mentioned above) to improve our work and how they can be related to Metaverse more. These three open-ended questions help us to identify the strengths of Metaverse's mobile D&D character creator, and to determine areas for improvement and the potential ways to improve them. A qualitative content analysis was used to evaluate the responses.

Open-ended interviews were also conducted to evaluate the mobile D&D character creator proposed in this study. The interviewers included User Interface experts, Gaming researchers/experts, and professional/highly experienced gamers. They were introduced to the character creator and asked for their opinions. Based on these interviews, our work has the following strengths. First, in terms of the user interface concerning the Metaverse, the electronic user interface can greatly reduce the difficulty of accessing information compared to a physical player, increase the fluidity and speed of the game for the player, and save the player from having to flip through hundreds of pages of rulebooks or record all sorts of character data. Secondly, the UI design follows the design language of traditional paper character sheets, which makes it easier for experienced players to become familiar with the game quickly. Thirdly. This paper creates a suitable software alternative to the character sheets for Dungeons & Dragons games, thus confirming ownership of the digital product (produced by the tool) and allowing the digital product to be freely traded between users.

However, there are several improvements that could be made to this character creator. The first and main drawback is that our UI design is text-based. There should be less textual information and a more intuitive visual design to improve the player experience. There should be more emphasis on graphic design and 3D environments or even the inclusion of a time dimension. The UI design should also be interesting, for example, by increasing the visual presentation and more contrasting colors to better appeal to the youth group rather than using boring information sheets. Another area for improvement is the placement of personality and background information on the same page, as in Figure 1. Displaying the same category of information on the same screen should be avoided, which may confuse players playing the game for the first time. Moreover, the adjustment values and proficiency details should be calculated automatically and these should be hidden by default so that the user only needs to fill in the appropriate attribute values to move to the next step to reduce the difficulty of using and understanding new users. In the future, we will improve on these aspects of this character creator discussed above.

In addition to the suggestions for the D&D character creator presented in this paper, experts also expressed their opinions on the user interface and gamer involvement of other applications for Metaverse. Firstly, in the logical design of the game, it is important not only to model the objects within the game realistically in 3D, but also to include the concept of time and give objects and users a 4D coordinate to enhance interactivity. In terms of the gameplay design, to enrich the metaverse game should be enriched with entertainment and gameplay. The concept of blockchain games can be introduced to combine Metaverse and virtual tokens. In addition, it can be associated with real-world objects, such as using physical cards to match the method to increase the entertainment and participation of the game. Moreover, the existing game resources should be integrated to facilitate more comprehensive access to information for players. In addition, the design of the game's UI should be user-centered with a more elegant design, and more advanced hints can be retrieved using gestures or the keyboard. We include the summary of eight interviews in the excel file.

## 5. CONCLUSION

This paper presents a prototype of a new product design and implementation process by illustrating the step-by-step development process for Aboleth, a virtual character sheet for playing Dungeons and Dragons 5th edition mobile application. We describe in much detail and explain how the primary code is executed, which later demonstrates the feasibility of our prototype in this study. We also discuss and reflect during the design and implementation process to explore the opportunities to improve the design of the application in the future.

The process highlighted the contribution of the Agile framework, HCI, and Xamarin apps in the new product design and implementation for enterprise mobile applications and Metaverse. The prototype made it possible to identify and understand how a loose variant of the agile framework in software engineering could integrate into HCI to facilitate user-centered design in a game design context and eventually improve user experiences. By employing the C4 model, the team can share understanding without defining details that might cause the architecture changes. Finally, the C4 model solution matched the relationship between each of the individual feature deliverables and their data sources is implemented to improve the performance of mobile applications. It could be used as an architecture and recommendation for enterprise mobile applications and Metaverse.

Our study makes several contributions. Firstly, the academic implication is that it demonstrates to other researchers in the field of games research a user-centered prototype that integrates software engineering and game design to improve the human-computer interaction of mobile applications. Secondly, the practical contribution is that this prototype provides a solution for enterprises, especially small and medium-sized enterprises, to quickly design and build their mobile applications for multiple platforms in a flexible and rapid manner. Finally, the C4 model solution can improve the performance of mobile applications, and foster the mobile application process.

**Acknowledgment**
This work is partly supported by VC Research (VCR 0000173).

**References**

Ahmad, A., Li, K., Feng, C., Asim, S. M., Yousif, A., & Ge, S. (2018). An empirical study of investigating mobile applications development challenges. *IEEE Access*, 6, 17711-17728.

Alsaid, M. A. M. M., Ahmed, T. M., Jan, S., Khan, F. Q., & Khattak, A. U. (2021). A Comparative Analysis of Mobile Application Development Approaches: Mobile Application Development Approaches. *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences, 58*(1), 35-45.

Bengtsson, D., & Jursenaite, G. (2019). A user study to analyze the experience of augmented reality board games.

Birmingham, C. C. (2021). A comparative analysis of nonverbal communication in online multi-user virtual environments (Doctoral dissertation)

Buruk, O. T., & Özcan, O. (2018, April). Extracting design guidelines for wearables and movement in tabletop role-playing games via a research through design process. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (pp. 1-13).

Chen, R., Rao, Y., Cai, R., Shi, X., Wang, Y., & Zou, Y. (2019, August). Design and Implementation of Human-Computer Interaction Based on User Experience for Dynamic Mathematics Software. In *2019 14th International Conference on Computer Science & Education (ICCSE)* (pp. 428-433). IEEE.

C4 model (2021), The C4 model for visualizing software architecture, https://c4model.com/, accessed on 30 December 2021.

Cochran, D. S., Arinez, J. F., Collins, M. T., & Bi, Z. (2017). Modelling of human–machine interaction in equipment design of manufacturing cells. *Enterprise Information Systems*, 11(7), 969-987.

Cockburn, A. (2002). Agile Software Development (Vol. 177). Boston, MA: Addison – Wesley

Rizov, T., Đokić, J., & Tasevski, M. (2019). Design of a board game with augmented reality. FME transactions, 47(2), 253-257.

Dhillon, J. S., Ramos, C., Wünsche, B. C., & Lutteroth, C. (2011, June). Designing a web-based telehealth system for elderly people: An interview study in New Zealand. In 2011 24th International Symposium on Computer-Based Medical Systems (CBMS) (pp. 1-6). IEEE.

Dix A. (2009) Human-Computer Interaction. In: LIU L., ÖZSU MT (eds) Encyclopedia of Database Systems. Springer, Boston, MA.

Faria, F. G., Pereira, L. T., & Toledo, C. F. M. (2019, October). Adaptive generation of characters for tabletop role playing games. In *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* (pp. 39-46). IEEE.

Garcia, A., da Silva, T. S., & Silveira, M. S. (2019, May). Artifact-facilitated communication in agile user-centered design. In *International Conference on Agile Software Development* (pp. 102-118). Springer, Cham.

Garg, H., Choudhury, T., Kumar, P., & Sabitha, S. (2017, February). Comparison between significance of usability and security in HCI. In *2017 3rd International conference on computational intelligence & communication technology (CICT)* (pp. 1-4). IEEE.

Hermes, D. (*2015). Xamarin mobile application development: Cross-platform c# and xamarin. forms fundamentals.* Apress.

Highsmith, J. (2013). *Adaptive software development: a collaborative approach to managing complex systems.* Addison-Wesley.

Hsu, C. C., & Chen, E. C. H. (2009, July). Exploring the elements and design criteria of massively-multiplayer online role-playing game (MMORPG) interfaces. In International Conference on Human-Computer Interaction (pp. 325-334). Springer, Berlin, Heidelberg.

Gary M. Olson and Judith S. Olson. (2003). Human-Computer Interaction: Psychological Aspects of the Human Use of Computing. Annual Review of Psychology 54(1), 491-516.

Helms, R. W., Barneveld, R., & Dalpiaz, F. (2015, January). A Method for the Design of Gamified Trainings. In PACIS (p. 59).

ISO 9241-210. (2010). Ergonomics of human-system interaction, human-centered design for interactive systems. Geneva, USA: Int'l Standards Organization.

K. Vishal and A. S. Kushwaha, "Mobile Application Development Research Based on Xamarin Platform," 2018 4th International Conference on Computing Sciences (ICCS), 2018, pp. 115-118, doi: 10.1109/ICCS.2018.00027.

König-Ries, Birgitta. "Challenges in Mobile Application Development" , vol. 51, no. 2, 2009, pp. 69-71. https://doi.org/10.1524/itit.2009.9055

Kretschmer, V., Mättig, B., & Fiolka, M. (2021, June). Dynamic Break Management in Logistics on the Basis of Individual Vital Data: Designing the User Interface of an AI-Based Mobile App for Employees in Order Picking. In *Congress of the International Ergonomics Association* (pp. 483-490). Springer, Cham.

Lin, W., Xu, M., He, J., & Zhang, W. (2021). Privacy, security and resilience in mobile healthcare applications. *Enterprise Information Systems*, 1-15.

Martin, L.J., Sood, S. and Riedl, M., 2018. Dungeons and DQNs: Toward Reinforcement Learning Agents that Play Tabletop Roleplaying Games. In INT/WICED@ AIIDE.

Mohammed, Y. B., & Karagozlu, D . (2021). A Review of Human-Computer Interaction Design Approaches towards Information Systems Development . BRAIN. Broad Research in Artificial Intelligence and Neuroscience, 12(1), 229-250.

Morschheuser, B., Hassan, L., Werder, K., & Hamari, J. (2018). How to design gamification? A method for engineering gamified software. Information and Software Technology, 95, 219-237.

Nawrocki, P., Wrona, K., Marczak, M., & Sniezynski, B. (2021). A Comparison of Native and Cross-Platform Frameworks for Mobile Applications. *Computer*, 54(3), 18-27.

Norman, D. (2013). *The design of everyday things: Revised and expanded edition*. Basic books.

Palmer, S. R., & Felsing, M. (2001). *A practical guide to feature-driven development*. Pearson Education.

Plijnaer, B. A., O'Neill, D., Kompier, E., Wallner, G., & Bernhaupt, R. (2020, November). Truesight Battle Grid-Enhancing the Game Experience of Tabletop Role-Playing through Tangible Data Visualization. In Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play (pp. 103-107).

Schöbel, S. M., Janson, A., & Söllner, M. (2020). Capturing the complexity of gamification elements: a holistic approach for analysing existing and deriving novel gamification designs. European Journal of Information Systems, 29(6), 641-668.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum* (Vol. 1). Upper Saddle River: Prentice Hall.

Stapleton, J. (1997). *DSDM, dynamic systems development method: the method in practice*. Cambridge University Press.

TCHERNAVSKIJ, P., WEBB, A. M., GEMEINHARDT, H., & MACKAY, W. E. (2022). Readymades & Repertoires: Artifact-Mediated Improvisation in Tabletop Role-Playing Games.

Tomitsch, M., Janssen, A., Curwood, J. S., & Thomson, K. (2021). Insights into the interrelationship between the design process and user practice: a case study on user engagement in a platform for professional learning. *Enterprise Information Systems*, 15(10), 1546-1561.

Wadley, G., Carter, M., & Gibbs, M. (2015). Voice in virtual worlds: The design, use, and influence of voice chat in online play. Human–Computer Interaction, 30(3-4), 336-365.

Williams, A. (2009). User-centered design, activity-centered design, and goal-directed design: A review of three methods for designing web applications. Proceedings of the 27th ACM international conference on design of communication, 1–8.

Xu, Z., Qiu, X., & He, J. (2016, August). A novel multimedia human-computer interaction (HCI) system based on kinect and depth image understanding. In 2016 International Conference on Inventive Computation Technologies (ICICT) (Vol. 3, pp. 1-6). IEEE.

Zhu, X., Xiao, Z., Dong, M. C., & Gu, J. (2019). The fit between firms' open innovation and business model for new product development speed: A contingent perspective. Technovation, 86, 75-85.

**Additional Resources**

1. D&D 5th Edition API - https://www.dnd5eapi.co/, accessed on 30 Dece,ber 2021

2. D&D Standard Reference Document - https://media.wizards.com/2016/downloads/DND/SRD-OGL_V5.1.pdf, accessed on 30 Dece,ber 2021.