## **Table of Contents**

1. [**Introduction…………………………………………………………………………………………………………………………………..1**](#_myc4ihp34vk1)

[1.1 Purpose……………………………………………………………………………………………………………………….……………….1](#_ltmm3gol9eun)

[1.2 Scope…………………………………………………………………………………………………………………………..……………….1](#_7f1f9ixa906x)

[1.3 Definitions, Acronyms, and Abbreviations……………………………………………………..………………..1](#_pxr9mu8hxmgy)

[1.4 References……………………………………………………………………………………………………………….………………..2](#_z65lefgab4zg)

[1.4.1 Comparison of Platforms………………………………………………………………………………….………………..2](#_r2vuowvzhe6d)

[1.4.2 Relational Databases (SQL) vs Non-Relational Databases (NoSQL)……….………….2](#_dbiar7rrjdfo)

[1.4.3 Relational Databases: MySQL vs PostgreSQL……………………………………………………………3](#_smynufghi8w6)

[1.4.4 Non-Relational Databases: MongoDB Atlas vs Google Firebase…………………………3](#_xh195f1u1bqy)

[1.5 Overview………………………………………………………………………………………………………………….………………….3](#_k8osu1domr15)

1. [**Overall Description………………………………………………………………………………………………….………………….4**](#_eo4rnuwdvm5o)

[2.1 Product Perspectives………………………………………………………………………………………..……….…………..4](#_bu1hu07v5za)

[2.1.1 Software interfaces…………………………………………………………………………..……………………..4](#_4fn7sflj1qod)

[2.1.2 Communication interfaces………………………………………………………….……………………….4](#_a2ds5vo5o9eh)

[2.1.3 Memory…………………………………………………………………………………………….…………………………4](#_6bvotprusfvf)

[2.1.4 Operation………………………………………………………………………………………..…………………………5](#_8qfsgoo77m4m)

[2.2 Constraints……………………………………………………………………………………………………………………………….5](#_aryus766k3as)

[2.3 Assumptions and Dependencies…………………………………………………………..………………………….5](#_5rawn3fg2n8f)

1. [**Specific Requirements…………………………………………………………………………………..……………………………6**](#_pi41emkqp9ge)

[3.1 External interface requirements………………………………………………………..………………………………6](#_9hst55lm614t)

[3.1.1 User interfaces………………………………………………………………………………………………………….6](#_41l08gyos4yj)

[3.1.2 Hardware interfaces……………………………………………………………………..………………………..6](#_x53x0oecrgqj)

[3.1.3 Software interfaces……………………………………………………………………..………………………….6](#_1m75r8kx2o20)

[3.1.4 Communication interfaces……………………………………………………..……………………………6](#_6farzxu766bw)

[3.2 Performance requirements…………………………………………………………………..……………….…………….6](#_3d5e42lwnsd6)

[3.3 Design constraints………………………………………………………………………………..……………………………….6](#_tbjzx539sctq)

[3.4 Software system attributes……………………………………………………………………………….………………..7](#_ax0vputc8yqn)

[3.4.1 Reliability………………………………………………………………………………………………….………………...7](#_smdrd0st39t0)

[3.4.2 Error Handling…………………………………………………………………………………….…………………..7](#_90qv2gr3eqkm)

[3.4.3 Ease of Use………………………………………………………………………………………….……………………7](#_ny4vm77udk09)

[3.5 Other Requirements……………………………………………………………………………………….……………………..7](#_k2ejl946wflb)

1. [**Software Architecture………………………………………………………………………………………….……………………..8**](#_vgk53dpf0r0v)

[4.1 High Level Architecture………………………………………………………………………………..……………………….8](#_smner2zhh6al)

[4.1.1 Architectural Diagrams……………………………………………………………….…………………………8](#_j802rlx74vlw)

[4.1.2 System Component & Interaction………………………………………….…………………………..8](#_x9iea8ud9q5d)

[4.1.3 Data Flow & Communication………………………………………………….…………………………….9](#_dx7kxyjt6wsz)

[4.1.4 Tech Stack…………………………………………………………………………………….……………………………9](#_n1pz1strlh94)

[4.1.5 Scalability Considerations…………………………………………………….……………………………10](#_2zylnweoencm)

[4.1.6 Security & Data Privacy………………………………………………………….…………………………….10](#_x28ogi238kqm)

[4.2 Game Level Architecture…………………………………………………………………….……………………………….11](#_6be53jahbzc3)

[4.2.1 Client Side Components…………………………………………………….………………………………...11](#_obzyc670k58n)

[4.2.2 Server Side Components………………………………………………….………………………………...12](#_tj5ch0sq5hk3)

[4.2.3 Networking…………………………………………………………………………….………………………………..12](#_tz8rao67en1i)

[4.2.4 AI Components…………………………………………………………………..…………………………………..12](#_1rbgyj4tpoz2)

[4.2.5 Persistence & Data Storage………………………………………….…………………………………...13](#_jcvi0b2b1ce)

[4.2.6 External Services……………………………………………………………..……………………………………13](#_eqgwf8yqvr06)

[4.2.7 Monetization……………………………………………………………………………………………………………………….13](#_to4ms2rca25c)

1. [**Comparison of Platforms…………………………………………………………………………………………………………15**](#_8ncut1v6k5as)

[5.1 Unity…………………………………………………………………………………………………………………………………………..15](#_cbkdbutgyr2t)

[5.2 Unreal Engine……………………………………………………………………………………………………………………..…16](#_i0hdoqpzzqk4)

[5.3 Web-application……………………………………………………………………………………………………………….…...17](#_98r5jir3xx1t)

[5.4 Final Recommendation…………………………………………………………………………………………………....…18](#_1iejpqke1474)

1. [**Potential Databases…………………………………………………………………………………………………………….…….18**](#_8fy60axjlrbx)

[6.1 MongoDB…………………………………………………………………………………………………………………………….……18](#_68lq9b72koqm)

[6.2 Firebase………………………………………………………………………………………………………………………………..….18](#_w6qrzxplf6ix)

[6.3 Google Cloud……………………………………………………………………………………………………………………….…18](#_ossygasxg2zw)

[6.4 Relational Databases (SQL) vs Non-Relational Databases (NoSQL).….….….….….…19](#_576gcsj7gzx2)

[6.5 Relational Databases: MySQL vs PostgreSQL………………………………………………………….…21](#_dqtdwtlf9996)

[6.6 Non-relational Databases: MongoDB Atlas vs Google Firebase……………………….…24](#_5xdx00i1ubz5)

[6.7 Final Recommendation………..……………………………………………………………………………………….……27](#_7gpc4d1fg3dq)

1. [**Final Blueprint / Specifications……………………………………………………………………………………………2**](#_j353n33hdqh0)**7**

# **1. Introduction**

### 1.1 Purpose

The purpose of this Software Requirement Specification is to detail the requirements and goals of the VitalBlinks business simulation game.

### 1.2 Scope

The VitalBlinks business simulation game is a platform for players to compete against one another in game theory strategies using simulated currency and product lines. Players have the option to borrow currency, sell product lines, acquire assets, and engage in business litigations.

### 1.3 Definitions, Acronyms, and Abbreviations

**SRS**: This document, the Software Requirements Specification (SRS), that illustrates the functional and non-functional requirements of the VitalBlinks platform

**UI:** An acronym for “User Interface”, which describes the way that a piece of software looks to users.

**UX:** An acronym for “User Experience”, which describes the way that a user experiences a piece of software, and ideally they’d have a good experience.

**Player:** A non-administrative user that has access to join game sessions within VitalBlinks, or start their own offline game against bots.

**Admin:** An administrative user that has access to create and spectate game sessions within VitalBlinks.

**User:** Either a player or admin who would be playing VitalBlinks.

**Team:** One or more *players* that work together within a game session of VitalBlinks to a maximum of 5 players.

**Bot:** A non-playable character (NPC) or computer-controlled player within VitalBlinks who a player can play against or with.

### 1.4 References

### 1.4.1 Comparison of Platforms

1. [Unity vs Unreal](https://kevurugames.com/blog/unity-vs-unreal-engine-pros-and-cons/)
2. [Unity pros/cons](https://citrusbits.com/a-unity-review-pros-and-cons/)
3. [Unreal reviews (gives pros/cons)](https://www.getapp.com/it-management-software/a/unreal-engine/reviews/)
4. [Unreal pros/cons](https://www.newgenapps.com/en/blogs/unreal-engine-review-pros-cons-and-suitability)
5. [Unity review + Unity vs Unreal](https://ventionteams.com/blog/unity-pros-and-cons)
6. [Web Development Info](https://developer.mozilla.org/en-US/docs/Games/Introduction)

### 1.4.2 Relational Databases (SQL) vs Non-Relational Databases (NoSQL)

Database Analysis and Management course

[Databases\_Overview.ipynb](https://colab.research.google.com/drive/1JbXAi2D1o1nRLmqC7RCHfDFeHv0N-20L?usp=sharing#scrollTo=H02K5kflxq4N)

[MongoDB Intro.ipynb](https://colab.research.google.com/drive/1fANwBEUHNJYVVCOYF4eK8L2yMgxy87DS?usp=sharing)

1. [IBM Source](https://www.ibm.com/blog/sql-vs-nosql/)
2. [Coursera Source](https://www.coursera.org/articles/nosql-vs-sql)
3. [Reddit Source 1](https://www.reddit.com/r/webdev/comments/xb4bbi/sql_vs_nosql_which_one_to_choose/)
4. [Reddit Source 2](https://www.reddit.com/r/learnprogramming/comments/lo5kpt/can_someone_explain_with_example_when_to_choose/)
5. [Reddit Source 3](https://www.reddit.com/r/Database/comments/rtnnyq/what_are_the_advantages_of_using_nosql_databases/)

### 1.4.3 Relational Databases: MySQL vs PostgreSQL

1. [Hackr.io Source](https://hackr.io/blog/postgresql-vs-mysql)
2. [Digital Ocean Source](https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems)
3. [Geeks For Geeks Source](https://www.geeksforgeeks.org/difference-between-mysql-and-postgresql/)
4. [Arctype Source](https://arctype.com/blog/postgres-ordbms-explainer/)
5. [Stack Overflow Source](https://stackoverflow.com/questions/1921437/is-it-legal-to-using-mysql-in-commercial-environment)
6. [Prisma.io Source](https://www.prisma.io/dataguide/postgresql/5-ways-to-host-postgresql#:~:text=Installing%20PostgreSQL%20on%20a%20separate,running%20on%20a%20dedicated%20computer)

### 1.4.4 Non-Relational Databases: MongoDB Atlas vs Google Firebase

1. [Appmaster.io Source](https://appmaster.io/blog/what-is-firebase)
2. [MongoDB Source](https://www.mongodb.com/firebase-vs-mongodb)
3. [Techmagic Source](https://www.techmagic.co/blog/firebase-vs-mongodb/)
4. [Medium.com Source](https://medium.com/mqos-technologies/mongodb-vs-firebase-which-is-the-best-database-in-2022-aff873566586)
5. [Reddit Source 1](https://www.reddit.com/r/Firebase/comments/yuxwec/firebase_vs_mongodb_atlas/)
6. [EzFire.io Source](https://ezfire.io/blog/cloud-firestore-vs-mongodb/)
7. [Fullstack Labs Source](https://www.fullstacklabs.co/blog/difference-between-mongodb-firestore-when-you-should-use-each)
8. [Reddit Source 2](https://www.reddit.com/r/mongodb/comments/12jwq2s/mongo_v_firebase/)
9. [Ably Source](https://ably.com/topic/scaling-firebase-realtime-database)

### 1.5 Overview

This document aims to provide information on the overall details on the development of VitalBlinks, detailing what future developers should aim to use and include. Details such as ease of use, security, data storage, exact softwares, information flow, and much more are included here.

**Section 1** introduces the document and VitalBlinks itself to readers. It also provides the references used to develop the information in the document.

**Section 2** covers several general topics of VitalBlinks: some non-user interfaces, memory, operation, product functions, constraints, and assumptions.

**Section 3** covers specific requirements of VitalBlinks: requirements of its interfaces and performance, design constraints, and attributes of its system software.

**Section 4** covers the architecture of the game’s software which includes high level architecture and game level architecture. This section mostly describes the backend part of VitalBlinks.

**Section 5** covers the comparison of several game platforms that can be leveraged to deploy VitalBlinks with. It also provides a final game platform recommendation.

**Section 6** covers the comparison of several databases that can be leveraged to use with VitalBlinks, and discusses the final game database recommendation. It also covers types of databases and their comparisons.

**Please refer to the VitalBlinks UI/UX Design Summary document for more information, visuals, and descriptions of in-game screens, icons, and more.**

# **2. Overall Description**

### 2.1 Product Perspectives

### 2.1.1 Software interfaces

There are two software interfaces used to create this project:

1. **Miro** is used to create the flowcharts used to then create the UI prototype for VitalBlinks.
2. **Figma** is used to create the final prototype with working buttons for users to navigate through the game menus within VitalBlinks.

### 2.1.2 Communication interfaces

A communication interface exists between the user’s local copy of VitalBlinks and the Firebase server. This will be through the Firebase Realtime Database.

### 2.1.3 Memory

When using Firebase’s free version, there are certain memory limits that they allow. Only one gigabyte of storage is allowed, and per-day only 50,000 document reads, 20,000 document writes, and 20,000 document deletes. Only 10 gigabytes of downloads per month, and only 100 simultaneous connections at a time. Given this, consider using the paid version of Firebase. Source: [link](https://firebase.google.com/docs/firestore/quotas).

### 2.1.4 Operation

The Firebase server must be on at all times in order to ensure that multiplayer is always available, but VitalBlinks can still operate in offline gameplay against bots if the server ever goes down. Firebase is provided by Google Inc., and it runs active servers at all times with adequate data protection.

### 2.2 Constraints

1. Getting comfortable with using Figma and Miro in order to develop the game’s User Interface.
2. Performing efficient research and development on all aspects of the game.
3. The physical game wasn’t able to be accessed to either play or experiment with.

### 2.3 Assumptions and Dependencies

1. Organizations will have computers available for admins and players.
2. These computers have internet access, a web browser, and access to our product.
3. Admins and players have emails which will be used to receive their special access code.
4. Admins and players are able to read and understand English.
5. Firebase and its Realtime cloud Database will be functional and running at all times.

# **3. Specific Requirements**

### 3.1 External interface requirements

### 3.1.1 User interfaces

The user interfaces for this game will mostly consist of buttons, basic shapes, and text. Details about these user interfaces are available in section 3.2.

### 3.1.2 Hardware interfaces

Both players and admins will need access to a computer, monitor, keyboard, mouse, and internet service in order to interact with VitalBlinks.

### 3.1.3 Software interfaces

VitalBlinks will communicate with the Firebase cloud database server in order to retrieve, save, and delete information from users and game sessions.

### 3.1.4 Communication interfaces

The Firebase server must communicate with the local copy of VitalBlinks in order to send and receive user and session information, so that players are able to have functioning multiplayer.

### 3.2 Performance requirements

The Firebase server must be able to reliably store and send information to and from game sessions in order to provide a quality multiplayer game experience. It must be able to happen in real-time with quick search algorithms so that retrieving information is fast for users wanting to receive certain data at any point.

### 3.3 Design constraints

A user’s computer must have an internet connection, a valid access code (also purchase codes for admins), as well as a local copy of VitalBlinks in order to access the game. Internet access is required to send and retrieve information from the Firebase Realtime Database. Otherwise, users won’t be able to send or retrieve information which can either prevent them from accessing the game at all, or from playing multiplayer.

### 3.4 Software system attributes

### 3.4.1 Reliability

Measures will be taken to ensure that both VitalBlinks, and its FIrebase server are reliable. Ensuring that these are reliable will allow for a smooth gameplay experience with fast responsive server communication and minimal game crashes.

### 3.4.2 Error Handling

Error messages will be displayed at any instance where the user enters invalid input or when the Firebase server has an error. These messages will be clear, understandable, and will be pop-ups in the user interface.

### 3.4.3 Ease of Use

VitalBlinks will be designed in an intuitive way that will make it easy for users to navigate. Moving from one screen to the next should be easy, and every screen should be absolutely necessary. The interface should have many quality of life features to have the best user-experience possible. Users should also not be able to be stuck on any screen or pop-up.

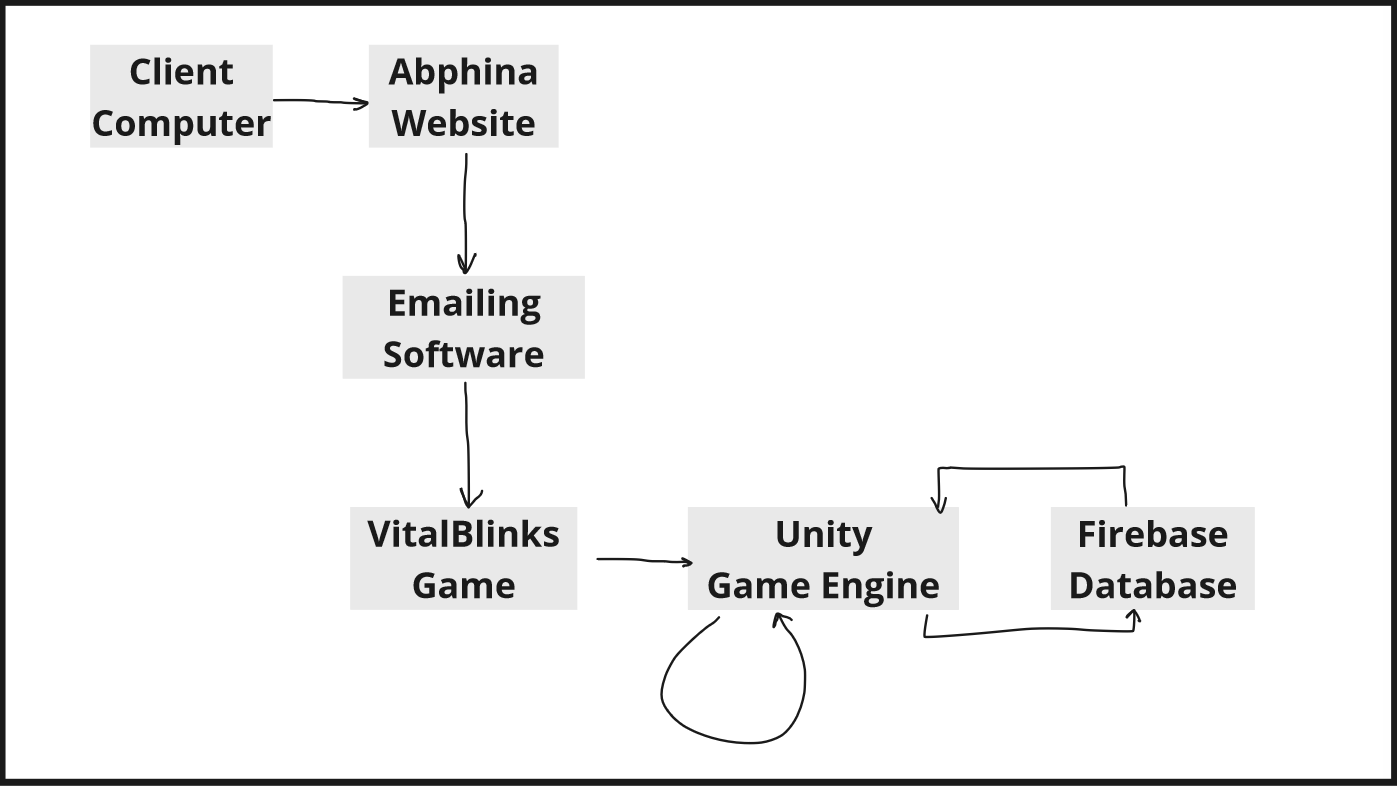
### 3.5 Other requirements

It was discussed that there may potentially be two versions of the game: one that can be completed within an hour, and one that can be completed over a semester as a capstone project. It is up to administration and management to determine which version they would like to focus on using the gathered information.

# **4. Software Architecture**

### 4.1 High Level Architecture

### 4.1.1 Architectural Diagrams



### 4.1.2 System Component & Interaction

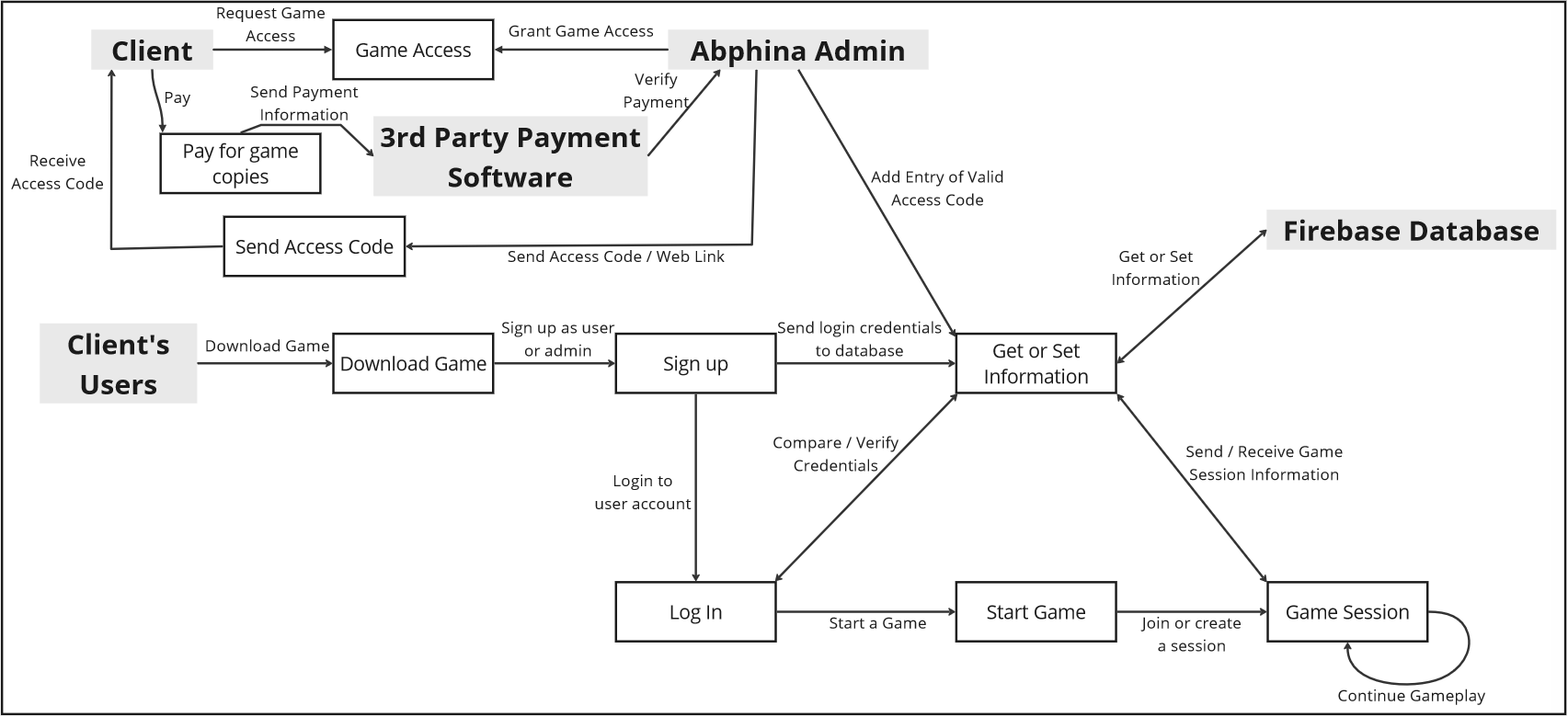
*Leaderboard*

A server uses the database as well as player score data from a game session in order to update players’ leaderboards.

*Player and Team Information*

A server uses the database in order to display information on players and teams, which will be used for admins.

### 4.1.3 Data Flow & Communication



### 4.1.4 Tech Stack

*Unity*

Unity is a game engine that allows for the creation of various game types (2D, 3D, Virtual Reality, Extended Reality, etc) which can be deployed on multiple platforms, or even be cross-platform.

*Firebase*

Firebase is an app development form that allows for the creation and growing of apps or games, created by Google.

*Firebase Realtime Database*

Firebase Realtime Database is a NoSQL (Non-relational) cloud database. It offers comprehensive app analytics in various categories, allowing administrators to gain insights into user behavior, app engagement, and app performance.

*Firebase Analytics*

Firebase comes with the feature of Google Analytics. This allows administrators to monitor and measure various statistics and events within their app. Administrators can also define custom events to measure specific things.

*Firebase Cloud Messaging (FCM)*

Firebase Cloud Messaging allows our server to have a reliable, free, battery-efficient connection with devices in order to deliver and receive messages between them. This applies to devices on iOS, Android, and even on the web.

### 4.1.5 Scalability Considerations

Due to the light amount of database access this game needs overall, scalability is not a serious concern. Scalability is well-automated with Firebase.

### 4.1.6 Security & Data Privacy

There’s no real reason to have additional security or privacy measures. Firebase already encrypts login information, and user accounts won’t have any personal information attached to them.

### 4.2 Game Level Architecture

### 4.2.1 Client Side Components

*VitalBlinks**Base Game*

Each player will download a copy of the game which includes all of the user interfaces (UI), bots, and overall game functionality.

*Online and Offline Bots*

Bots for both online and offline gameplay will be available with the base game. This includes the other computer players that players can battle against offline, as well as the bots for online games such as the Judge and Reader.

### 4.2.2 Server Side Components

*Sending / Updating Session Information*

The server should send all information of teams’ and players’ scores within a session to admins and other players of the same session so they can all have a correct leaderboard.

*Hosting Games*

The server should be in charge of hosting game sessions, allowing players to connect to these sessions

*Multiplayer*

The server should be in charge of transferring information from team to team to allow for multiplayer gameplay.

*Chat Capability*

If an in-game chat is opted for where you can type to other teammates, then the server will be in charge of sending and receiving messages to and from other players or teams. Would have to also consider different messaging modes: messaging all players, messaging only your team, messaging certain teams, or messaging certain players.

### 4.2.3 Networking

This game should use a client-server model for online multiplayer, where players connect to a game session hosted by the server. Through this session, the server can transfer information with the players but the server doesn’t handle game logic. The server should just send information, and then the local copy of the game takes this information and performs the appropriate action.

### 4.2.4 AI Components

All bots will be included with the base game, and thus exist on the local machine. Bots for online gameplay need information from the server in order to take particular actions, but bots for offline gameplay do not need any information from the server.

### 4.2.5 Persistence & Data Storage

Session data will be destroyed after the session is complete, but session-specific data persists within that session for as long as it’s running. For each session there is a block of storage allocated for it within the database for users to access information.

### 4.2.6 External Services

Firebase notification services, analytics, authentication, Google analytics, in-app messaging, cloud functions.

### 4.2.7 Monetization

Any institution but *only* institutions should be able to sign up and pay for the game. If payment is successful, then have someone send copies of the game out to whoever ordered them, as well as creating a new unique purchase code for them. Possibly have something else in line in order to avoid people stealing / giving out copies for free. Maybe each purchase code can be used however many times the buyer requests (up to a certain limit with possible exceptions) and no more.

Currently, the model of using purchase codes and access codes to verify purchases is a simplistic model for earlier iterations of the game. Later versions may use more complex purchase and verification protocols. Based on our current model, the following should be observed:

1. Purchase codes allow for a limited number of administrators.
2. Access codes allow for a limited number of administrators and players.

These restrictions prevent people from giving out copies of the game for free.

However, the possibility of VitalBlinks account sharing using throwaway email addresses is still possible. One possible way to reduce (not eliminate) this possibility is to limit the purchase of the game in earlier iterations to institutions, disallowing individuals. Institutions cover two of the largest groups of customers, namely academia and corporate. Permitting only institutions has two benefits: institutions are more likely to adhere to legal conditions disallowing improper distribution of the game, and the members of each institution have email addresses that are both unique and identifiable by their common suffix. The game also requires an institution name/number when registering, so this restriction is natural.

Additional monetization possibilities can be achieved by controlling the following features:

1. How many games each administrator can create in total;
2. How many games each administrator can run simultaneously;
3. The additional option of team selection either by manual player selection or algorithmically;
4. The option for solo play against bots;
5. The ability to modify parameters of the game, e.g. the minimum point balance, the end-of-level multipliers, number of levels before a card must be handled, etc.
6. The level of detail that administrators can see when viewing game, team, and player statistics;
7. The ability to download game statistics as a spreadsheet (useful especially for large classrooms and corporate trainings, where it would be tedious to manually go through the game screens);
8. Bonus cards (example scenarios of the biopharma and management world);
9. Voice chat

# **5. Comparison of platforms**

### 5.1 Unity

*Pros:*

1. Allows deployment of more than 25 platforms
2. Intuitive interface
3. Easy to use for beginners
4. When it comes to VR/AR/XR, it’s faster, easier, and supports more platforms
5. Doesn’t require developers to have computers with high software specifications
6. Royalty-free

*Cons:*

1. If you want read-only code, need Unity Enterprise subscription
2. Lower performance compared to other highly specialized engines → lags during world editing if too many assets, textures, and such
3. Advanced multiplayer features only available by downloading frameworks
4. Less quality graphics than Unreal → including VR/AR/XR quality
5. Apparently licensing is an annoying process
6. “Not suitable for simple games”? Unsure if this is true or if VitalBlinks is even a ‘simple game’. (from source: [link](https://www.newgenapps.com/en/blogs/unreal-engine-review-pros-cons-and-suitability))

*Notes:*

* Free for projects generating less than $100,000 a year
  + “Unity Pro or Unity Enterprise plans are required for businesses with revenue or funding greater than $200K in the last 12 months, and for those who do work with them. Pro and Enterprise plans have no financial eligibility limits – everyone is eligible. Please note that the Enterprise plan is for larger teams and requires a minimum purchase of 20 seats.” [link](https://unity.com/pricing)
* Written in C#

### 5.2 Unreal Engine

*Pros:*

1. High quality visuals → including for VR/AR/XR
2. Great performance even with many / high quality assets and environments
3. Advanced toolkit for multiplayer, AI and VR
4. Blueprint coding system
5. More stable than Unity → frequent updates, accurate bug fixes are prioritized

*Cons:*

1. Fairly complex interface → harder for beginners to learn
2. Need to pay 5% royalties if product earns more than $1,000,000
   * 1. [link](https://www.unrealengine.com/en-US/faq)
3. High system requirements for fully developed games (Storage space, graphics card, memory, etc)
4. Hard to run on computers that have low system specifications
5. Can often crash in editor
6. Some say that many aspects of editor get in the way since they’re usually unneeded in games

*Notes:*

* Main feature is the support for high definition graphics / AAA-quality games
* Written in C++

### 5.3 Web-application

*Pros:*

1. Have absolute control of basically all aspects of the game
2. Updates and bug fixes would be faster to implement
3. No royalties or license fees

*Cons:*

1. It’ll be harder to implement since HTML5 and Javascript will have to be used, which are much more tedious languages.
2. Based on opinion, since full control is granted then that means every single aspect will also have to be implemented which makes it tougher to create.

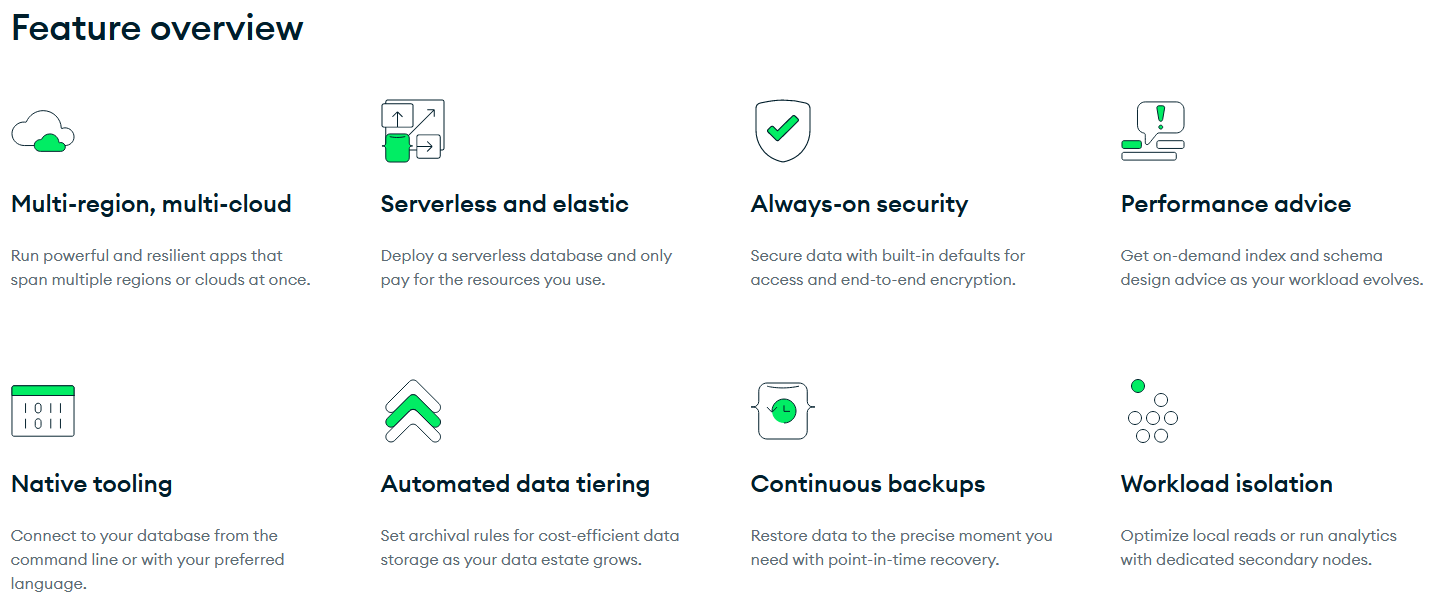
### 5.4 Final Recommendation

Using Unity will be the best option. It just makes the most sense: VitalBlinks doesn’t need state-of-the-art visual graphics (as given by Unreal) and is a simple 2D card game. Web-development will make the process more difficult and tedious. However, be sure to consider conflicts with the cost of using each platform based on what kind of revenue the game is expected to generate.

# **6. Potential Databases**

### 6.1 MongoDB

[Features](https://www.mongodb.com/atlas/database):

**

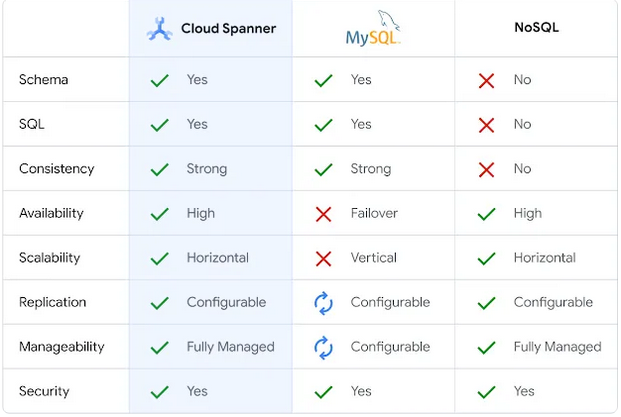
MongoDB for gaming: [link](https://www.mongodb.com/use-cases/gaming)

### 6.2 Firebase

More info on which of their DB’s to choose: [link](https://firebase.google.com/docs/firestore/rtdb-vs-firestore)

### 6.3 Google Cloud

Features ([link](https://cloud.google.com/solutions/databases/games)):



### 6.4 Relational Databases (SQL) vs Non-Relational Databases (NoSQL)

Structure:

SQL:

Data is stored as tables, with columns as attributes and rows as records.

Pros: Data is easily parsed and manipulated. Therefore, complex data operations can be handled well, and the transfer of data from one database to another is relatively easy, whether the destination database is SQL or NoSQL.

Cons: The structure of the database must be set up properly, to ensure correct relations between attributes within a table, and correct relations between tables to one another.

NoSQL:

Data is stored as key-value pairs, documents, or graphs.

Pros: Accommodates unstructured data or changing data sets well. Easy to set up, since the dynamic structure of NoSQL databases does not require much planning or organization of data.

Cons: complex data operations are not handled well, since the relationships between data are not clearly defined. Data is not easy to port between databases.

Scalability:

SQL:

Scales vertically - typically data is stored on a single server, and if more capacity is needed, either the server needs to increase its hardware capabilities, or the database needs to shift onto a larger server.

NoSQL:

Scales horizontally - if more capacity is needed, additional servers can be added to handle the load.

Support:

SQL:

Has existed for a long time, and is thus well-tested and has extensive documentation and large support communities.

NoSQL:

Relatively new, and shows promise. However, its shorter history and non-standardized formats result in less testing and expertise.

Additional considerations:

SQL:

SQL databases must be ACID compliant, which is a set of properties (atomicity, consistency, isolation, durability) that ensure data validity for all database transactions. Therefore, SQL databases are highly reliable.

NoSQL:

Many are integrated into backend-as-a-service (BaaS) platforms. These platforms provide a host of services supporting the database that reduce the need for manual setup.

### 6.5 Relational Databases: MySQL vs PostgreSQL

Usability:

MySQL:

Relatively simple, and thus easier to users less familiar with databases. Since it is the world’s most popular database management system (DBMS), there is extensive expertise and documentation for assistance. Many third-party tools exist to help manage MySQL databases.

PostgreSQL:

Newer and more advanced DBMS, and thus more complex with more features. Harder for new users. While support is not as extensive as for MySQL, due to its lower popularity, there is still an active and growing user community. Fewer third-party tools exist to help manage PostgreSQL databases.

Features:

MySQL:

Not fully compliant with SQL standards. Basic SQL commands are covered, but some more complex and less used features like full outer joins (complete union of tables) are not supported. MySQL is a purely relational DBMS, and therefore cannot support objects or table inheritance.

PostgreSQL:

Nearly fully compliant with SQL standards, and has many optional features, such as soft deletion (marking data as unusable without erasing it). PostgreSQL is an object-relational DBMS, and therefore supports features such as objects, table inheritance, and extension of data types.

Speed:

MySQL:

Designed for speed, by reducing supported SQL features. This is particularly true for read operations. However, because its handling of concurrency (multiple users at the same time) is via locks, write operations may be slower.

PostgreSQL:

Designed for complex tasks and data integrity. Read operations may not be as fast as MySQL. However, because it uses Multi-Version Concurrency Control (MVCC), it can handle a larger number of simultaneous users and higher concurrent write speeds.

Scalability:

MySQL:

Supports relatively easy replication (copying the database to more servers and updating the copies), which is useful for backups or horizontal scaling. Also relatively easy to upgrade to a commercial MySQL product for automatic sharding (splitting the database into smaller parts to be hosted on multiple servers), which is important for horizontal scaling.

PostgreSQL:

Replication is supported, but it is a more complex and a newer feature on PostgreSQL. Better support for vertical scaling.

Price:

MySQL:

Originally open-source, currently owned by Microsoft. Has a dual-licensing model, where the open-source community edition is free, and the commercial edition is paid. The community edition can be used for commercial purposes, but the commercial edition provides more plugins and features.

PostgreSQL:

Completely free and open-source, and can be used for commercial purposes.

### 6.6 Non-Relational Databases: MongoDB Atlas vs Google Firebase

Features:

MongoDB Atlas:

MongoDB database, merged into a BaaS product. MongoDB Atlas primarily centers around its database. Data in MongoDB databases is stored as documents. As the most popular NoSQL DBMS, MongoDB is well-developed with powerful querying tools for its database. MongoDB Atlas provides servers around the world for cloud storage, metrics, alerts, user authentication, and security measures.

Google Firebase:

A BaaS product, including the Firestore DBMS. Google Firebase is built primarily as a suite of services to handle most backend needs, especially for mobile apps. Data in Firestore databases is stored as documents. Google Firebase provides servers around the world for cloud storage, metrics, alerts, user authentication, security measures, in-app messaging, and test labs.

Usability:

MongoDB Atlas:

Relatively easy to learn and set up, but perhaps more complex than Firebase. May require more third-party applications for support than Firebase, since its ecosystem is less complete. The MongoDB database itself allows for more powerful queries, and as it is more widely used than Firestore, it has more complete documentation. MongoDB Atlas is a newer product, and thus has less support.

Google Firebase:

Easy to learn and set up. Well integrated with other Google services, such as Google Cloud and Google Analytics, making it simple to connect those services to support the application. Has good documentation from Google and other developers. Lacks in certain querying features such as aggregations.

Performance:

MongoDB Atlas:

MongoDB is faster than Firestore, with both read and write speeds exceeding Firestore’s. Supports more complex queries. Designed to function well for a range of data loads, ranging from light to medium to heavy.

Google Firebase:

Firestore is slower than MongoDB, and allowed queries are simpler. However, the suite of services it comes with make it much easier to jumpstart application development. Designed for mobile application data, which tends to be lighter.

Scalability:

MongoDB Atlas:

The MongoDB DBMS is designed for distributed databases, and accordingly, MongoDB Atlas makes it easy for new servers to be set up in different regions, for data to be replicated, and for data to be sharded.

Google Firebase:

Like MongoDB Atlas, horizontal scaling is simple with Google Firebase. Firebase provides the ability to set up new servers in regions across the world, auto-sharding, and auto-scaling of database size depending on real-time needs.

Price:

MongoDB Atlas:

Free up to a limit. Then options are 1) serverless, where you are charged based on reads and writes to the database, data traffic, etc. 2) dedicated cluster, where you pay per hour for a dedicated server.

Details can be found here:

<https://www.mongodb.com/pricing>

Google Firebase:

Free up to a limit. Then there is a serverless paid plan, where you are charged based on reads and writes to the database, data traffic, cloud services used, etc.

Details can be found here:

<https://firebase.google.com/pricing>

### 6.7 Final Recommendation

The four databases compared are some of the most popular and common databases for commercial use, especially for game development. Among them, Google Firebase is the best option for VitalBlinks. As a database on its own, Google Firestore is likely the newest and most primitive of the databases. However, the types of data being transmitted for VitalBlinks are simple (primarily numbers), the database queries are likely not complicated, and the volume of data transmitted is not especially high. Firestore should be sufficient to handle these workloads. Given its ease of setup (being a NoSQL database), along with the well-integrated suite of services that support the database (which many other databases do not have, requiring expertise in third-party software), the compromise on database functionality is worth it, and should not have a heavy impact on the quality of the game experience.

# **7. Final Blueprint / Specifications**

* Unity
  + Build so that it can run on MacOS, Windows, Linux
  + Using most recent version of Unity is recommended
  + Consider using paid option
* Firebase (see [Tech Stack](#_n1pz1strlh94))
  + Is in charge of hosting game sessions created by admins
  + Will temporarily store info of game sessions and delete them after the game ends
  + Allows for multiplayer gameplay
  + Consider using paid option
* Some game details:
  + Max team size 5
  + Minimum team size 1
  + 10 levels
  + No limit to how many teams there are (Feel free to make a limit though)
  + Make game have maximum time of 1.0 to 1.5 hours
  + Team leaders are a single player of a team who should have ability to edit some team info (Slogan, description, region of operation, etc) as well as manage players: kick, mute
  + Types of users:
    - ***Players***, who can
      * Join / Resume games, can only be active in 1 game at a time
      * Play against bots in offline game sessions hosted locally
    - ***Admins***, who can:
      * Host games, also multiple at a time
      * View overall game statistics & statistics per team
      * Spectate games
  + Player’s can’t host games, admins can’t join games
  + Joining a game:
    - Players can only join games via a valid game session ID, no game lobbies or anything like that
  + Start of game:
    - 10 product lines are split amongst members of a team (make sure to have method for odd cases such as 3 members) for free
      * Players are able to customize each of their respective product lines
    - However, players must pay for business executives
    - Players can borrow from bank even in beginning of game, before level 1 starts
    - Forming teams: can either let players choose / remove teammates, or randomly assign equally-balanced teams (all teams are close in member count)
      * For player-choice method, player who creates team is the team leader
      * For random-assignment method, randomly assign team leader
    - Game starts when ***admin*** who created game session presses a button to start the game
      * Can implement system where each team checks a box indicating they’re ready, and admin can only start game when all teams are ready (to avoid early accidental starts)
  + Gameplay:
    - Judge, Reader & Banker are now computer-controlled ‘players’ who players can interact with, no player will take these roles
    - Have each player of a team switch roles each level (Banker, Legal, Merger & Acquisition, team leader, etc)
    - When receiving cards for product lines, each pair is ***randomly chosen*** for the player, who then chooses to flip each one or put it aside to settle them later
    - Using 10 legal aids does NOT negate the impact of a legal action (comes from fact that each decreases impact by 10%), will simply reduce by 10% 10 times: 100 → 90 → 81 → … → 34.867…
    - Difficulty affects how many points each card plays for, increases risk / reward factor
    - Each level begins by briefly explaining its game theory, what to keep in mind, and what players are aiming to learn
    - Implement a system so that all players are ***forced*** into the next level after an interval of time if they have not advanced yet → keep in mind players are able to advance to other levels quicker than the rest if they so choose, but this is not a beneficial decision for them
    - Cards are either positive and give points (think of a card like monopoly where it’ll say “You did this and so you get <amount>!”) or a ***challenge*** card where you have to make a decision yourself, and if you choose wrongly then points will be deducted
      * If a player doesn’t make a decision for their card by the second consecutive level after receiving the card, then the game will actively make the ***worst*** choice for you
    - Teams who lose the game early become spectators, being able to explore the entire map and the dashboard, view other teams’ details and see their progress, and they’re no longer able to chat with everyone else, they’re only able to chat amongst themselves (isolated spectator chat)
    - Phantom rule is not known about and not visible until the ***end of the game***, also note that spectators who lost earlier in the game are able to vote as well
  + Level 10:
    - ***Level 10 is not like levels 1 - 9***, players simply need to prioritize winning
    - 3 win options:
      * 1) Follow games original path
      * 2) Wager up to 1000% of face value of picked cards
        + Cards unseen prior to wager
      * 3) Pay 250 points per picked card
        + Cards unseen prior to payment
      * → Judge appraises all assets and liabilities of each team, highest points wins
* See other PDFs (Rule Print) for more constraints
  + 10 levels, each have set themes and game theories
  + Rules
    - Bank
    - Legal
    - General
    - Merger & Acquisition
* Needed choices for VitalBlinks
  + Need some kind of banking system to use for processing customers’ payment for VitalBlinks
  + Choose email software for Abphina admins to get in contact with customers (gmail, yahoo mail, etc)
* See general game notes that Avinash, David, and Joshua took here, might help with clearing up any details not caught in this blueprint section: [link](https://docs.google.com/document/d/1MuyseeB239OpPJb2K2Ux0CdRSbRNbsqKFTkTPZa5qaE/edit#heading=h.wexcmitgsjgm)
  + Other information can be found on the UI/UX document, and help can be found from the visuals provided from the interns’ creations on [Miro](https://miro.com/app/board/uXjVM6huKZs=/?share_link_id=501627345659) and [Figma](https://www.figma.com/file/P6HYyqYKCPBQZWzKShRxNp/Vital-Blinks-UI-Design?type=design&node-id=128%3A1855&mode=design&t=hTbq2r8Y1JnHwnIb-1)