



Technical Report

[VIRTUAL GAME ITEM TRADING PLATFORM]
THOMAS BOYLE – X12410922@STUDENT.NCIRL.IE
BSHC4 – CLOUD COMPUTING (2015/2016)

CONTENTS

Executive Summary	4
Introduction.....	5
Background	5
Competitor Platforms	5
Aims 5	
Technologies	6
SQS – Simple Queue Service	6
Dynamo DB – NoSQL Cloud Database.....	6
Platform Bots – C# SteamBot.....	6
PlayFramework	6
CloudFlare	7
System	7
Requirements	7
Functional requirements.....	7
Data requirements	8
User requirements	8
Environmental requirements.....	9
Usability requirements.....	9
Design and Architecture	10
High-Level Architecture	10
Implementation	11
Java Manager Classes.....	11
Data Access Object Classes	12
Testing 13	
Software Testing	13
Load Testing	14
Customer Testing	15
Graphical User Interface (GUI) Layout.....	16
GUI Screenshots.....	16
Evaluation	19
User/Customer Feedback	19
Conclusions.....	21

Further development or research	21
References	21
Appendix.....	22
Project Proposal.....	22
Objectives	22
Background	23
Wireframe UI	23
Technical Approach.....	24
Technical Details	25
Evaluation	26
Project Plan.....	26
High Level Analysis & Design	27
Introduction	27
Architecture Design	28
Application Program Interfaces	30
User Interface Design.....	32
Homepage Mockup.....	32
Profile Mockup.....	32
Virtual Inventory Mockup	33
Appendix B: Key Terms	34
Monthly Journals	35
Reflective Journal September	35
Reflective Journal October	36
Reflective Journal November	37
Reflective Journal December	37
Reflective Journal January	39
Reflective Journal.....	40
Reflective Journal.....	41
Glossary Of Terms.....	43

National College of Ireland

Project Submission Sheet – 2015/2016

School of Computing

Student Name:

Student ID:

Programme: **Year:**

Module:

Lecturer:

Submission Due Date:

Project Title:

Word Count:

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:

Date:

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

EXECUTIVE SUMMARY

The project will allow users of the “STEAM” platform to trade & bet their virtual game items inside the project platform. These virtual game items are offered to players as cosmetic enhancements for the weapons or characters in a game. Players can choose to buy these items from Steam directly.

The project gives users the ability of storing their own virtual items on the platform for use in trading and betting. These stored items are accessed via web hosted GUI, this allows users to import items from their Steam account or export their items from the platform back to their Steam account. “Trade Offer”, is an offer of trade between two users of the Steam platform, a trade offer contains the items you want from the receiver and items you’re willing to give. Trade-Offers are used extensively in the project as means of transferring items between the users and the project platform.

Cloud computing services & infrastructure provide the basis of the project’s technical footing. A cloud hosted No-SQL database holds user information as well as information about the user’s game items. Cloud queue messaging services act as buses of information between the cloud components. Cloud hosted “Bots” interact with the Steam platform and are instructed to carry out tasks for the project via a cloud messaging queue. These bots hold the platform users virtual game items. When a user wants to export an item from the project platform a bot interfaces with the Steam API and sends that user a trade offer with the item they requested. The bots can be scaled horizontally, with each bot capable of holding 1,000 virtual game items.

INTRODUCTION

BACKGROUND

Virtual game items are entities with no physical presence. Gaming micro-transactions are becoming a huge source of revenue for game companies. Game companies are selling virtual goods to players for real currency. Occasionally certain gaming platforms allow their users to trade these items or even sell these items for credit on that platform.

Steam is a PC gaming platform with an active virtual ecommerce marketplace. Users can sell digital game items on this market for steam store credit, which they can use to buy games or additional content.

Various sites exist that implement some form of gamification of Steam items. I desired to create a similar platform that utilized cloud technologies and infrastructure. The use of cloud services will remedy common issues plaguing current competitors such as denial of service, high-latency response times and downtime. SteamBots would be used like nodes in a distributed system, these nodes would be given tasks to perform against the Steam network (new trade offer, accepting trade). This nodes view of the bots allows for increased horizontal scaling, with each bot being able to hold 1000 virtual items maximum.

COMPETITOR PLATFORMS

www.csgolounge.com (Specific Game-Item Trading/Betting)

www.tf2wh.com (Specific Game-Item Trading)

www.tf2outpost.com (Specific Game-Item Trading)

Each competitor platform offers its own unique version of betting and/or trading of virtual game items. Tf2wh.com specializes in "Team Fortress" (Game) items which doesn't directly compete with the project. However Csgolounge.com specializes in "Counterstrike" virtual game items and as such does compete with the project market share for that game.

AIMS

The project aims to create a platform that will be capable of coping with a large user demand, in excess of 1000+ concurrent active users. The platform should allow users the ability to add their game items into the platform and to remove their game items. The platform will provide a "virtual" inventory for its users, with the items residing in the user's virtual inventory distributed across multiple "bot" steam accounts.

The platform plans to support 1 game at launch, this game is called Counterstrike. This game contains various cosmetic virtual items suitable for use in the project platform. These virtual items are also known as "skins" for the game's community.

The platform should be built on top of cloud technologies and services, it should follow best practices when implementing these services. Messaging Queue services will provide backlog capabilities and will allow for information to travel quickly between each component. Cloud hosted database will provide fantastic scaling and fault tolerance. DynamoDB has a three time replication policy in place.

TECHNOLOGIES

Below is a list of the current project technologies, each with a brief description outlining its role within the project. These technologies are critical to the project as they each serve an important role in the project's architecture. The architecture diagram further illustrates this point.

SQS – SIMPLE QUEUE SERVICE



An Amazon web service cloud hosted message queue system that allows for communication between project components. When a bot is instructed to carry out a task for the system it will be sent the job via SQS as a JSON message.

The result of that job will also be returned via SQS JSON message to the SteamBotManager SQS queue.

DYNAMO DB – NOSQL CLOUD DATABASE



An Amazon web service which provides a cloud hosted NoSQL database with 3 times replication globally. This database is used extensively in the project for holding data on the platform users and their items within the platform. The database also contains various tables for holding metadata on different project components.

DynamoDB provides SDK support for Java and also has a Java object mapper, which makes integrating it into the project very easy. As you can save modelled Java objects directly to a database.

PLATFORM BOTS – C# STEAMBOT



SteamBot is a C# project created by GitHub user "JesseCar96".

The project is open source and available at the following GitHub repository:
<https://github.com/JesseCar96/SteamBot>

This "SteamBot" is the most important component of the project as it will allow the project to create Steam bots that will be given tasks to carry out against the Steam network such as sending trade offers to users of the platform or simply storing items for the project platform users. The library is completely unit tested and regularly updated.

Each bot is configured in a JSON configuration file, this file contains various details such as account names and passwords for the bots. It also includes some timeout settings and various configuration parameters.

PLAYFRAMEWORK



"Play Framework makes it easy to build web applications with Java & Scala. Play is based on a lightweight, stateless, web-friendly architecture."
<https://www.playframework.com/>

The framework is used to host the front end of the project, the framework makes use of model view controller (MVC) pattern. Users of the project will all communicate with this framework, it will render the web-pages for each user/device accessing the project.

The front-end for the project is written in Scala running on the PlayFramework. This framework provides a multitude of benefits and makes development very agile. HTML & CSS are also used with the bootstrap framework for the HTML rendering.

CLOUDFLARE



“CloudFlare is a free global CDN and DNS provider that can speed up and protect any site online.”

<https://www.cloudflare.com/>

The project uses CloudFlare as protective measure from denial of service attacks. The domain name servers for the project point to the CloudFlare DNS servers that handle any requests and can filter out malicious requests. It can also make effective use of CDN thereby reducing load on the main server and improving latency times for the project users.

SYSTEM

REQUIREMENTS

Use cases for all functional requirements can be found in the attached requirements spec. Below are the functional requirements for the project as well as additional various general non-functional requirements.

FUNCTIONAL REQUIREMENTS

This section outlines the functional requirements for the project. These functional requirements state what the platform will do.

- User Registration
 - This will allow users to create an account for the platform. This account will enable users to access the platform and interact with the platform. An account is required for trading and betting.
- User Login
 - This will allow users with an account to authenticate and sign-in to their account. You must be logged in to trade & bet on the platform.
- Administration Controls
 - This will allow users with administrative rights the ability to modify platform properties and settings on the fly without having to modify any of the code.
- Betting
 - Bet items
 - Users can bet virtual items on competitive game matches, the odds will self-balance depending on the value of items placed on both teams.
 - Users can enter raffles, where their chance of winning is based on the total value of their items put into the raffle.
 - Bet against platform users
 - Users can bet virtual items against each other in virtual games hosted on the platform.
 - Collect rewards

- If user wins a bet he receives the original items he bet plus the additional items he won. If a user loses a bet his items are forfeited and are given to the users who bet on the correct team.
- Cloud Hosted
 - The project will be hosted on Amazon Web Services, it will use several AWS services. Including DynamoDB, SQS and perhaps EC2 for the servers, depending on cost.
- Web Based Interface/GUI
 - The web based GUI will be developed in bootstrap and rendered by a tomcat java web server. The web server will use a model view controller pattern to render the web pages dynamically.
- Manage Virtual Item Inventory
 - Add Item
 - A user can add an item to their virtual platform inventory.
 - Remove Item
 - A user can remove an item from their virtual platform inventory.
 - Retrieve item value
 - The relative value of each item a user owns is displayed to them.
- User Profile Editing
 - Email Change
 - A user can update their email address for their account.
 - Username Change
 - A user can update their username for their account.

DATA REQUIREMENTS

1. Passwords will not be stored, only the hashed equivalent.
2. SSL connections will be forced for all data between users.

Hashed Passwords

Should use some form of SHA encryption for passwords. No passwords will be stored in clear text in any form, a salt will also be used to further increase the complexity of the hashed passwords.

SSL Connection

All data will be secured and encrypted end-to-end to prevent man in the middle attacks. The domain for the project will be issued a SSL certificate.

USER REQUIREMENTS

1. Login/Registration
2. Steam OpenID Connect
3. Virtual Inventory Functionality

Login/Registration

Login & Registration will be handled by the Steam OpenID connection requirement. This requirement will require all users to have a steam account before being able to login and access the website.

Steam OpenID Connect

Competitor platforms and other Steam based web applications, allow their users to login via Steam's OpenID policy which allows any user to use their steam account for a different website. This is called an OpenID connection and is a requirement both for the user and system.

Virtual Inventory Functionality

Users should be able to add inventory items to their virtual inventory within the project platform. They can then perform actions on those items:

- Remove item(s)
- Add item(s)
- Bet item(s)
- Trade item(s)

ENVIRONMENTAL REQUIREMENTS

1. Linux OS (Ubuntu Server LTS) for production infrastructure
2. [SteamBotManager Host] Java 8 minimum
3. [SteamBot Hosts] .net 4.5 required

Linux OS Ubuntu Server LTS

Ubuntu Server with long term support will be used in all production machines for consistency and for my own familiarity.

Java 8

Java 8 is a requirement on SteamBotManager hosts as the project makes use of Java 8 features such as:

- Lambda Expressions
- Java Streams

.Net 4.5

SteamBot requires .Net 4.5, as such each bot environment must have it installed.

USABILITY REQUIREMENTS

1. Responsive Web Design
2. Simple UI, all screen sizes supported
3. Modern browser support

Responsive Web Design

The website will use a responsive framework (bootstrap) to ensure that the page dynamically supports different screen sizes.

Simple UI

The UI should not be confusing, it should be clean and elegant and will allow easy usability of the project for its users.

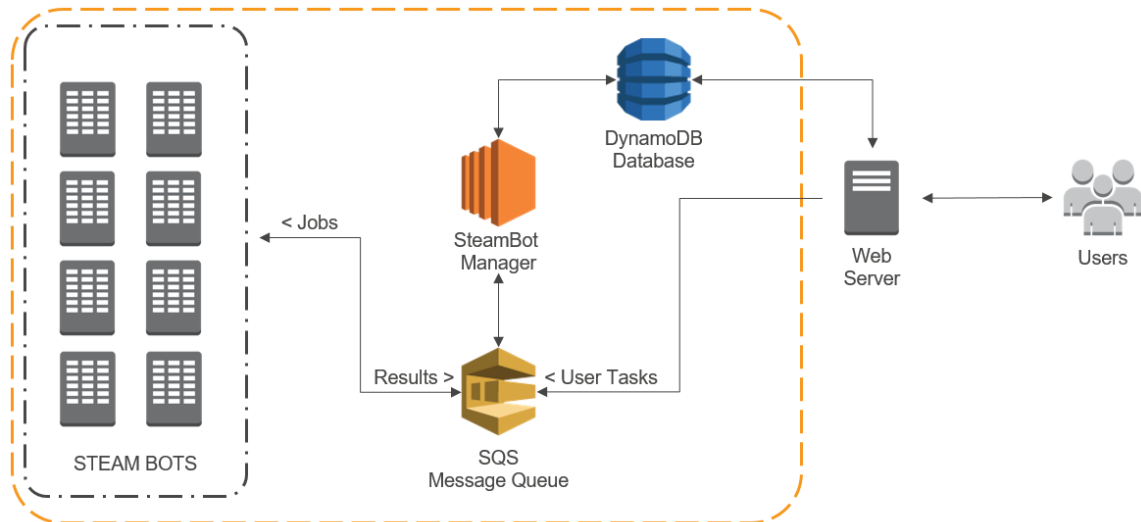
Modern Browser Support

- Usable on all devices, mobile browser friendly
- Chrome, Edge, Firefox support

DESIGN AND ARCHITECTURE

Describe the design, system architecture and components used. Describe the main algorithms used in the project. (Note use standard mathematical notations if applicable).

HIGH-LEVEL ARCHITECTURE



WEB SERVER

The web-server uses the Play-Framework, which is a web framework built using “Scala” a functional programming language which runs on the Java Virtual Machine. This framework is highly scalable and provides concurrency and asynchronous actions support.

DYNAMODB

DynamoDB is a NoSQL cloud hosted database used thoroughly in the project for user information holding and project component meta-data storage.

SQS MESSAGE QUEUE

SQS is a cloud hosted messaging queue, used in the project as a communication bus between several components. Bots are each given their own queue and all bots return job results to the SteamBot Manager SQS Queue.

STEAMBOT MANAGER

The SteamBot manager is a project component coded in Java. This component manages all of the SteamBots the platform currently has. The manager receives jobs from users via SQS message queue. These jobs are processed and dispatched to the appropriate Bot to carry out the task.

STEAM BOTS

The bots are essential for the project as they hold all of the platform items, a bot can contain a total of 1000 virtual game items. A bot can receive and send items to project users who have a steam account. A bot has an account name and password that is validated against the Steam platform when it starts.

IMPLEMENTATION

The application is developed using Java, Scala, and C#. It consists of a number of key classes performing the SteamBotManager (Java), MVC Framework (Scala), and SteamBot(C#) roles.

SteamBot Manager (Java)

JAVA MANAGER CLASSES

Singleton classes handling various aspects of the SteamBotManager.

- BotJobManager.class
 - Sends and handles bot jobs between SQS queues.
- InventoryMannger.class
 - Pulls inventory data from the Steam API
 - Returned data is JSON formatted.
- SteamBotManager.class
 - Connects to all the SteamBots, issues health checks and can perform soft-restarts on the bots if they experience errors.
- TradeOfferManager.class
 - Can send trade offers to steam users.
- VirtualInventoryManager.class
 - When a trade offer is accepted on a Bot, the changes must be reflected for the project user's virtual inventory.
 - If an item is removed from the platform or added to the platform this must be reflected on the user's virtual inventory.

BotJobManager Job handle Code

```
public void handle(ManagerJob job){
    if (validJob(job.getJobId(), job.getJobTimestamp())) {
        switch (job.getManagerJobType()) {
            case EXPORT_INVENTORY: exportSteamInventoryJob(job.getBotName(), job.getJsonJobStr());
                break;
            case IMPORT_INVENTORY: importSteamInventoryJob(job.getBotName(), job.getJsonJobStr());
                break;
            case NEWTRADE_OFFER: createNewTradeOfferJob(job.getBotName(), job.getJsonJobStr());
                break;
            case CANCELTRADE_OFFERS: cancelTradeOffersJob(job.getBotName(), job.getJsonJobStr());
                break;
            case EXPORT_TRADE_OFFERS: exportTradeOfferStatusJob(job.getBotName());
                break;
            case IMPORT_TRADE_OFFERS: importOutgoingTradeOffersJob(job.getBotName(), job.getJsonJobStr());
                break;
            case STATUSCHECK_RESPONSE: botManager.handleStatusCheckResponse(job.getBotName(), job.getJsonJobStr(), job.getJobTimestamp());
                break;
            case TRADEOFFER_CREATED: tradeOfferManager.registerOutgoingTradeOffer(job.getBotName(), job.getJsonJobStr());
                break;
            default:
                Logger.warn("Found Job with invalid job type, job:{}", job);
        }
    }
}
```

DATA ACCESS OBJECT CLASSES

“Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.”
(TutorialsPoint Design Patterns, 2015)

This pattern is used to provide object mapping and functionality between the DynamoDB database tables. It makes use of interfaces to decouple the database operations from the code.

BOT DAO

Handles object transfer between database, stores and pulls bot metadata from the database.

- BotDaoImpl (Class)
- BotDao (Interface)

USER INVENTORY DAO

Handles Steam inventory updates, grabs inventory data for a Steam user and stores it.

- InventoryDaoImpl (Class)
- InventoryDao (Interface)

Save Inventory Code

```

public void save(UserInventory userInventory) {
    try{
        dynamoDBMapper.save(userInventory);
        logger.debug("Saved inventory data successfully, steamId: {}, size: {}",
            userInventory.getSteamId(),
            userInventory.getItemList().size());
    }catch (Exception ex){
        logger.warn("An error occurred while saving inventory data",ex);
    }
}
}

```

VIRTUAL INVENTORYDAO

Handles platform virtual inventory objects between database, stores and updates virtual inventory changes of the database.

- VirtualInventoryDaoImp (Class)
- VirtualInventoryDao (Interface)

SQS Job Poller

Listens to the SteamBotManager SQS queue and regularly pulls new messages off the queue for processing within the manager.

POLLING CODE:

```

private void pollQueue() {
    Logger.debug("Started Polling SQS Queue:{}", SQSQueueURL);
    ReceiveMessageRequest messageRequest = new ReceiveMessageRequest()
        .withQueueUrl(SQSQueueURL)
        .withWaitTimeSeconds(1);

    while (isActive) {
        ReceiveMessageResult result = sqsClient.receiveMessage(messageRequest);
        result.getMessages().parallelStream().forEach(this::handleSQSMessage);
    }
}
}

```

TESTING

SOFTWARE TESTING

Software testing is used thoroughly throughout the project to ensure that each component and service of the project functions as should be. This is done via a combination of unit and integration testing frameworks.

UNIT & INTEGRATION TESTING TOOLS

- Junit (Java Library)
 - For unit & integration testing

- Junit is used heavily for unit testing, it enables the majority of code tests.
- Mockito (Java Library)
 - Powerful library for mocking classes for further testing
 - Mocks allow for classes to be created simply and values to be injected if need be.
- Powermock (Java Library)
 - Another powerful mocking library with reflection capabilities

UNIT TEST FOR DYNAMODB FUNCTIONALITY

```
public class DynamoDBUserDaoTest
{
    @Autowired
    private DynamoDBUserDAO userDao;

    @Test
    public void testFindById()
    {
        Private String expectedSteamId = "testId";
        Private String expectedIconUrl = "testUrl";
        Private String expectedUsername = "testUserName";

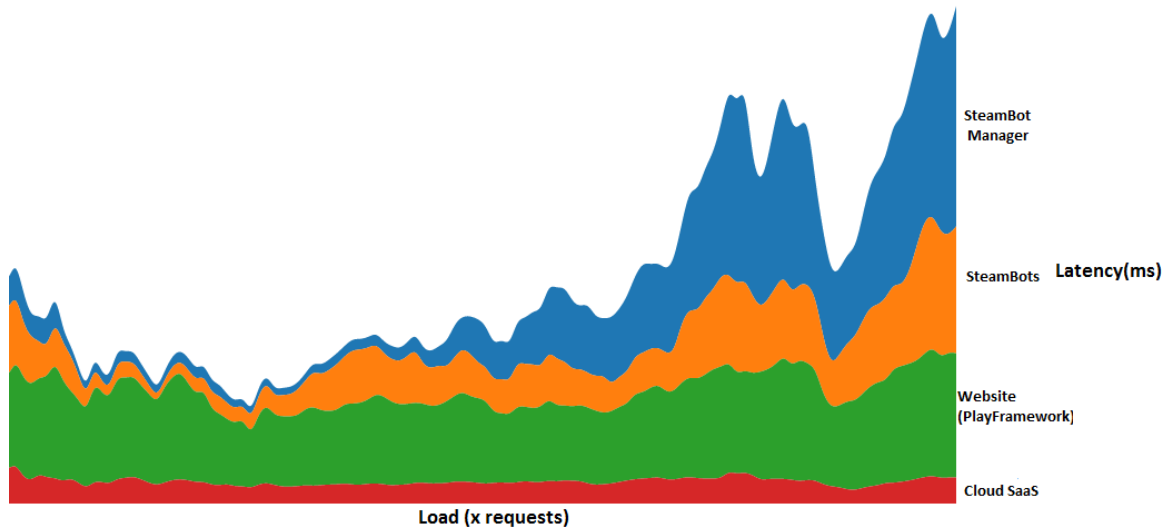
        User user = userDao.findbySteamId(01);

        Assert.assertEquals(expectedSteamId, user.steamid());
        Assert.assertEquals(expectedIconUrl, user.getIconUrl());
        Assert.assertEquals(expectedUsername, user.getUsername());
        return;
    }
}
```

LOAD TESTING

Load testing programmatically executes scripts to target the platforms components and "stress test" them each individually.

RESPONSE LOAD TEST



- Load was increased initially then reduced and incrementally increased until 500 requests per minute was reached.
- The test was conducted with the project hosted on cloud resources, no elastic scaling was used.
- The purpose of the test was to demonstrate which services are more likely to bottleneck
- The D3 JavaScript library was used to generate the graph from a programmatically generated csv load test result file.
- The test concludes that the steambot manager had the highest response rate when at the 500 requests per minute peak. This primarily due to the fact that the manager can only send jobs to available bots that aren't busy with other tasks.

CUSTOMER TESTING

Only users who have used other competitor platforms have been selected for testing & feedback.

USER: TESTER 1

“The website is pretty decent, it has all of the usual features expected from counters strike betting websites. I had no difficulty using the website as I’m familiar with other sites that have the same features and layouts. I logged in with my steam account with no difficulty and could instantly add items to my website inventory. A bot sent me a trade request within seconds and within several more seconds after accepting I could see my counterstrike item in my inventory. I placed one bet on a counter strike tournament but unfortunately lost.” ~ Tester 1 (01/05/2016)

Rating: 4/5

Conclusion: The website works as advertised, no difficulty with trading or using the website. I don’t know how it will hold up over time. I didn’t get a chance to use the website under high-load so I can’t fully recommend the website just yet.

USER: TESTER 2

“I have used websites like this in the past, so I expected similar features. Rightly so the project uses a simple design and has similar features to csgojackpot.com. The chat box was pretty cool as I don’t see it

enough on other sites. The inventory management is pretty standard the bot sends you a trade offer with items you're withdrawing or items you're depositing. I tested adding a low value skin to the website and it worked pretty well and quickly. I tried updating my profile trade token but ran into errors which weren't apparent to Thomas. After correcting the input errors I could successfully add my trade token, and placed a bet on a match and won. But I didn't receive any rewards as no other users had bet on the opposing team at the time." ~ Tester 2 (02/05/2016)

Rating: 3.9/5

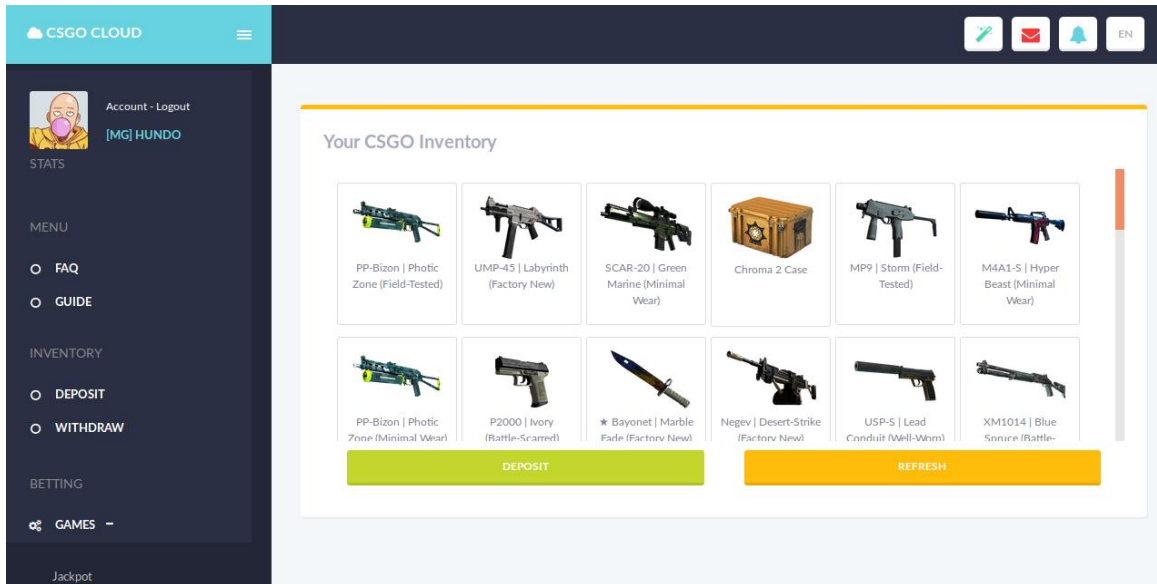
Conclusion: The site is good I like it, however some additional testing before a launch is required. If I see no more errors I would increase the rating by 1.

GRAPHICAL USER INTERFACE (GUI) LAYOUT

The layout has evolved over time, and has changed from the original mock-ups featured in the original project plan, to the now below.

GUI SCREENSHOTS

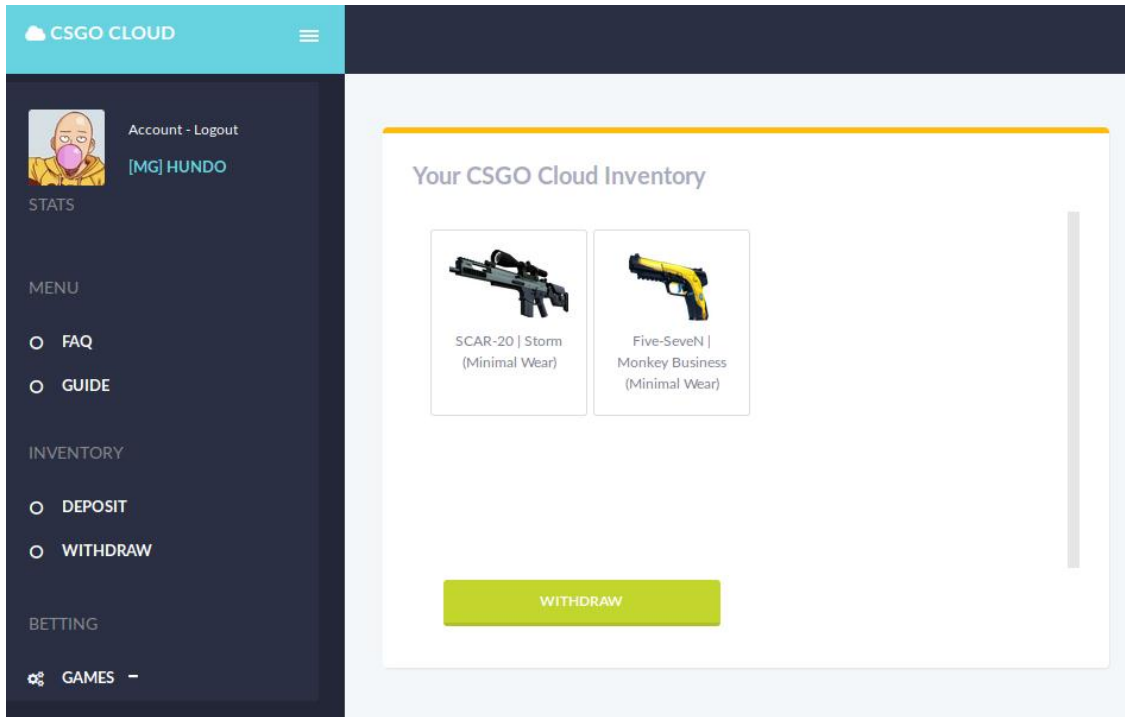
INVENTORY DEPOSIT



This inventory deposit has several elements:

- A multi-select image form for selecting all your CSGO items
- This page has the ability to deposit selected items
- The option to force a refresh of the cached inventory is also possible

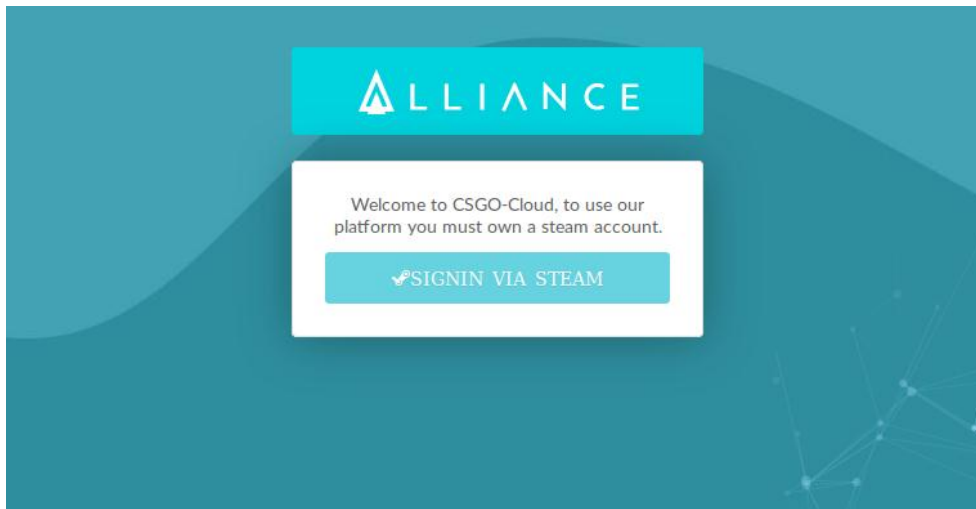
INVENTORY WITHDRAW



This inventory withdraw page has several elements:

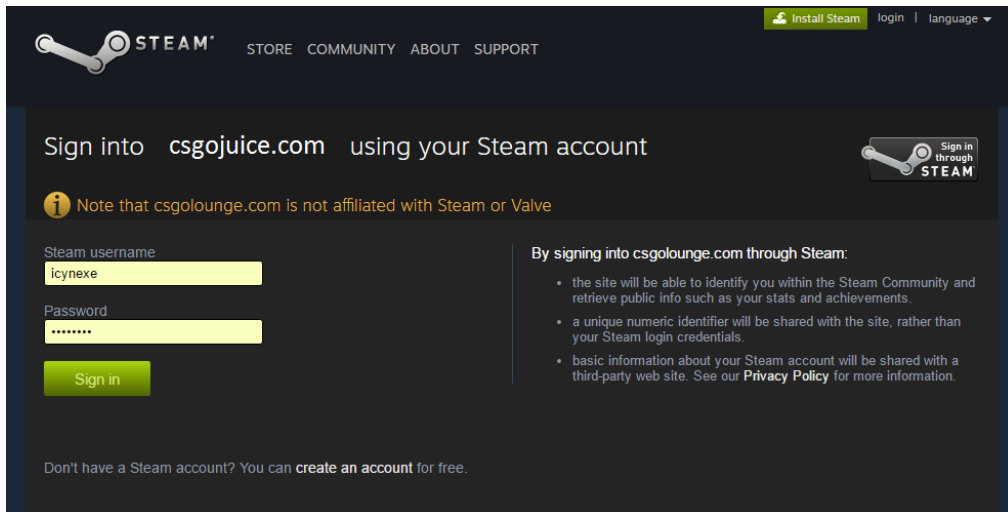
- The ability to withdraw your items
- The sidebar feature on the left

SIMPLE LOGIN



- The purpose of this page is redirect to the steam open id page

STEAM OPENID LOGIN PAGE

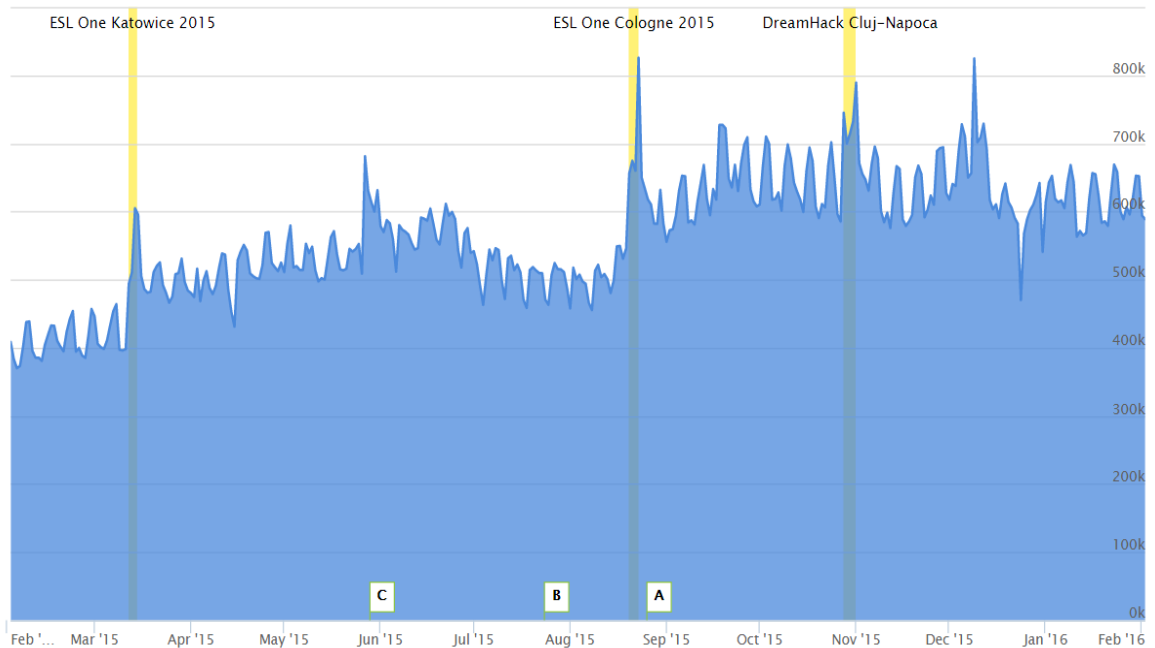


- After a user successfully authenticates they are returned to the website and their user information is stored/updated.

EVALUATION

Counterstrike Player Statistics

- 18.6 Million total owners
- 7.5 Million unique monthly users



Counterstrike active player count [2015/2016]

USER/CUSTOMER FEEDBACK

I brainstormed a couple of project features with a few friends who regularly use similar trading & betting game item platforms. These friends also all own Steam accounts and are familiar with trade offers and virtual game items. I compiled a short list of important features they felt were important to the project.

Features:

- Provably Fair algorithm
 - Validate that the platform is not cheating
- Steam OpenID support
 - Example: “Login using your steam account”
- Simple UI
- Bonus Rewards for referrals to the website.

CONCLUSIONS

ADVANTAGES

The project is complicated in several aspects, the cloud aspect provides more than enough scalability for the project. This has some drawbacks, it increases development time and more testing is required. By building it for the cloud it has to be loose-coupled and fault tolerant. However the cloud will provide an advantage over current competitors in the same market so the extra development time is worth it.

The project is designed to support more than 1 game, each class and component of the project is designed with reusability in mind for future game support.

DISADVANTAGES

The project components are distributed in nature and this requires a lot of effort from a development point of view before end-results can be pushed for all the users of the project.

LIMITS

The STEAM API is rate limited and several extra IP blocks may be need to circumvent this limitation if the project begins to approach those limits. For the final project release each set of 4 bots is given 1 IP, meaning each container/host is only hosting 4 bots at any given time.

The C# Steambot is extremely useful, however if Steam update their network or make any drastic changes to their API, then this could cause service downtime or outages. However these changes are also likely to affect competition as they all use the same API.

FURTHER DEVELOPMENT OR RESEARCH

With more time and resources a service orientated architecture approach could be taken, this would increase the fault tolerant capabilities of the project and would provide an even smoother experience for end users.

Service orientated architectures are tricky and hard to design, but the investment is worth it. Each component of the architecture could be separated into even smaller one function services. For example the SteamBotManager could only handle steam bot management and could allow another services to handle virtual inventory management for the platform users. When one component of a service oriented architecture fails it becomes easy to identify the service at fault, this reduces recovery time.

REFERENCES

Amazon Web Services, Inc. "Amazon SQS – Message Queuing Service – AWS". 2016. Web. 1 Feb. 2016.

Amazon Web Services, Inc. "AWS | Amazon Dynamodb - Nosql Cloud Database Service". 2016. Web. 1 Feb. 2016.

Playframework.com. "Play Framework - Build Modern & Scalable Web Apps with Java and Scala", 2016. Web. 2 Feb. 2016.

www.tutorialspoint.com. "Design Pattern Data Access Object Pattern". 2016. Web. 4 Feb. 2016.

"JesseCar96/SteamBot". *GitHub*, 2016. Web. 5 December 2015.

APPENDIX

PROJECT PROPOSAL

OBJECTIVES

Project Objectives

Virtual Items are entities with no physical presence. Gaming micro-transactions are becoming a huge source of revenue for game companies. Game companies are selling virtual goods to players for real currency. Occasionally certain gaming platforms allow their users to trade these items or even sell these items for credit on that platform.

Steam is a PC gaming platform with an active virtual ecommerce marketplace. Users can sell digital game items on this market for steam store credit, which they can use to buy games or additional content.

My project will allow users to post a trade offer for their items (similar to an ad listing website), or the user can choose to sell their item in return for a digital currency on the platform (similar to eBay). The digital currency will lock users to the platform, but it will also let them to purchase available virtual items or goods.

Platforms already exist for this niche, however I believe my project could improve on the user experience and would also attract more users than other competitors.

Competitor Platforms

www.csgolounge.com (Specific Game-Item Trading/Betting)

www.tf2wh.com (Specific Game-Item Trading)

www.tf2outpost.com (Specific Game-Item Trading)

Steam API

Steam provides a web API for managing virtual-items, these libraries & API's will be heavily used in the project. They will provide the core functionality of the virtual-item management that the platform is built around.

Steam Web API Example: www.steamcommunity.com/id/cynix/inventory/json/753/6

The project will:

- Allow users to sell and trade their virtual-items
- Support at least 1 game at launch
- Hosted in the cloud & using supporting cloud services
- Be resilient & redundant for high traffic/load

- Exchange their virtual-items for credits
- Purchase new virtual-items for credits
- Deliver all virtual-items to the correct users

BACKGROUND

Project Background

I have a great interest in video-gaming, from “Super Mario Land 2 (1993)” to modern AAA titles such as “Metal Gear Solid (2015)”. I have noticed a trend over the past 4 years and that trend is the rise of virtual items. These virtual items are generally presented to the user as some form of game item or game booster. These items usually cost real currency and can be purchased immediately by the user via the game they’re playing.

These virtual game items can sometimes be worth thousands of euros depending on the rarity of an item. The more rare an item is, the more demand exists for that item and the value reflects that demand.

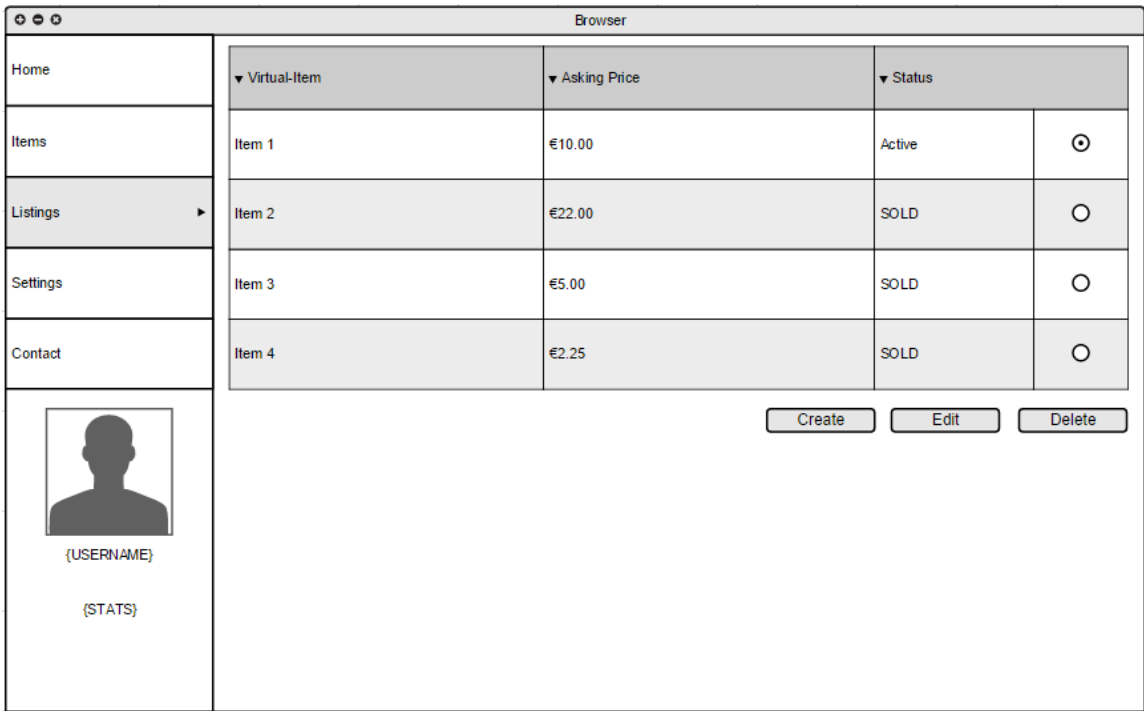
Further Reading: <http://www.pcgamer.com/microtransactions-the-good-the-bad-and-the-ugly/>

A virtual marketplace with automatic trading of these items would allow people to find the items they want or to get rid of the virtual items they don’t want. A niche currently exists for that purpose and this project aims to improve on current solutions and be the most reliable cloud hosted item trading solution.

Some other sites take a different approach to the virtual-item platform and instead offer betting platforms for competitive tournaments which users can then bet their own virtual-items.

WIREFRAME UI

The user-interface for the front-end will be simple and easy to use. It will make use of bootstrap to quickly build a responsive website.



TECHNICAL APPROACH

The website will be a lightweight java webserver, the front-end will be load balanced to ensure service stability. The front-end and back-end will both be cloud hosted.

The front-end technologies will include:

- JSON & GSON
- Bootstrap HTML5
- Java Web-Server (Tomcat, etc.)

The back-end technologies will include:

- Java
- JSON & GSON
- AWS Dynamo DB
- AWS SQS
- Steam Web-API
- Steam Login-API (OpenID)

More information on the Steam API can be found here:

<https://steamcommunity.com/dev>

STEAM OPENID

Will be used as the main authentication method, however we supply alternative registration options as well. A steam login will be required however to trade Steam virtual-items.

STEAM WEB-API

This will API will be used primarily by back-end servers to send virtual items to users or to request virtual items from users.

API Call Examples

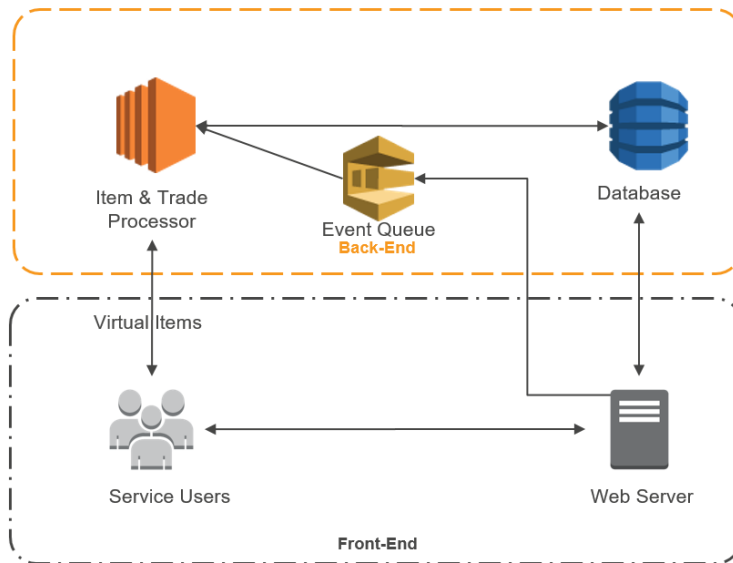
GET GetOwnedGames

"http://api.steampowered.com/IPlayerService/GetOwnedGames/v0001/?key=XXXXXXXXXXXXXXXXXX&steamid=76561197960434622&format=json"

GET PlayerSummaries

"http://api.steampowered.com/ISteamUser/GetPlayerSummaries/v0002/?key=XXXXXXXXXXXXXXXXXXXXXXXXXX&steamids=76561197960435530&format=json"

TECHNICAL DETAILS



ARCHITECTURE OVERVIEW

Item & Trade processor will be written in Java. It will be code tested using readymade testing frameworks such as Junit. Any unit or integration tests will be documented and their purposes explained.

Event-Queue will be an Amazon Web Services service called “SQS” or “Simple Queue Service” which will let me queue up tasks for the item & trade processor.

Database will be Amazon Web Service Dynamo DB, it is a very fault tolerant No SQL database.

Web Server will be a java web server, most likely tomcat.

USER DATA

User data will be stored in a NoSQL database such as Dynamo DB. This data will be used to authenticate users and store meta-data on the service users.

ITEM DATA

Virtual-Item Data will be ingested again via steam’s API. It will allow you to pull every item for a particular game as well as its current market price.

EVALUATION

The software written by me, will all be thoroughly tested with the correct unit and integration tests. Using testing frameworks such as Mockito or Junit will instantly give me results from my software tests.

Evaluation Tools

- Junit (Java Library)
 - For unit & integration testing
- Mockito (Java Library)
 - Powerful library for mocking classes for further testing
- Powermock (Java Library)
 - Another powerful mocking library with reflection capabilities

The service should withstand high-load by utilizing cloud solutions for load-balancing and workload distribution. This should be tested and the service should demonstrate fail-over capabilities in the event of a server outage.

Thomas Boyle / 02-10-2015

PROJECT PLAN

I used “smartsheet” to plan the project Gantt chart. I had some issues when using “Microsoft Project”. Smartsheet saves all my progress online and the link below will always show the latest version.

Task Name	Start Date	End Date	Duration	Q3			Q4			Q1			Q2			
				Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul
[-] Virtual Item Market Project	09/26/15	03/04/16	116d													
[-] Project Documentation	09/26/15	11/09/15	32d													
Project Proposal Doc	09/26/15	10/02/15	6d													
Cloud Architecture Design	10/28/15	11/02/15	4d													
Project Requirements Doc	10/05/15	11/09/15	26d													
[-] Back-End Inventory Server	11/02/15	02/12/16	75d													
Prototype	11/02/15	11/05/15	4d													
Coding	12/21/15	02/12/16	40d													
Unit & Integration Tests	02/02/16	02/05/16	4d													
[-] Front-End Web Server	11/05/15	01/08/16	47d													
Coding & Design	11/16/15	12/25/15	30d													
Unit & Integration Tests	01/05/16	01/08/16	4d													
Database configuration	11/05/15	11/11/15	5d													
[-] Quality Assurance Testing	03/04/16	03/04/16	1d													
UI - Review																
Prototype Review & Feedback	03/04/16	03/04/16	1d													

The current plan needs to be update with newer information.

HIGH LEVEL ANALYSIS & DESIGN

INTRODUCTION

Purpose of the Product Design Specification Document

The Product Design Specification document documents and tracks the necessary information required to effectively define architecture and system design of the virtual item game item trading platform in order to give the development team guidance on architecture of the platform to be developed. The Product Design Specification document is created during the Planning Phase of the project. Its intended audience is the project manager, project team, and development team. Some portions of this document such as the user interface (UI) may on occasion be shared with the client/user, and other stakeholder whose input/approval into the UI is needed.

General Overview and Design Guidelines/Approach

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

Assumptions / Constraints / Standards

1. The Steam API has no immediate rate limits (Assumption)
 - a. Testing is needed to proof this
 - b. If it does a plan must be put in to distribute the API calls.
2. The Steam API can be volatile (Constraint)

- a. Health checks need to be put in place to monitor the steam service.
 - b. If it is down, no trades should be issues until it recovers.
3. Software Testing (Standards)
 - a. Unit testing must be used thoroughly to ensure bug free code
 - b. Integration testing must be used selectively to test the service before deployment of each new revision.

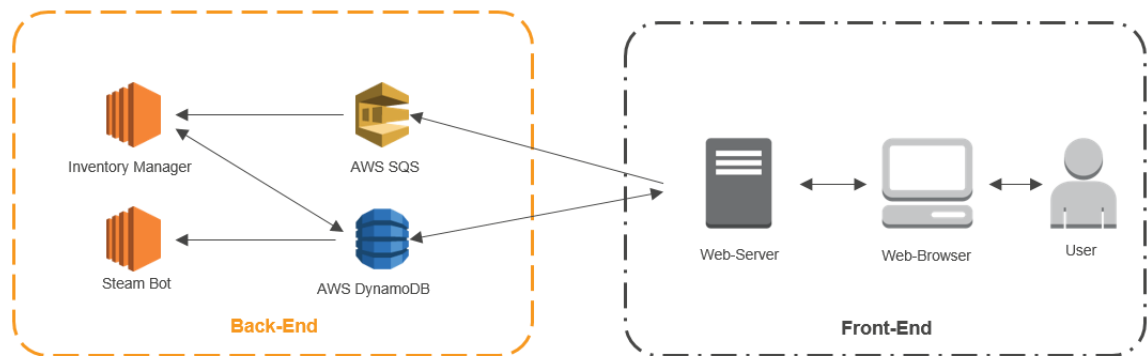
ARCHITECTURE DESIGN

This section outlines the system and hardware architecture design of the system that is being built.

Logical View

This describes the platforms architecture, including both front-end & back-end. The diagrams below will illustrate how the individual components are linked together.

HIGH-LEVEL SYSTEM ARCHITECTURE

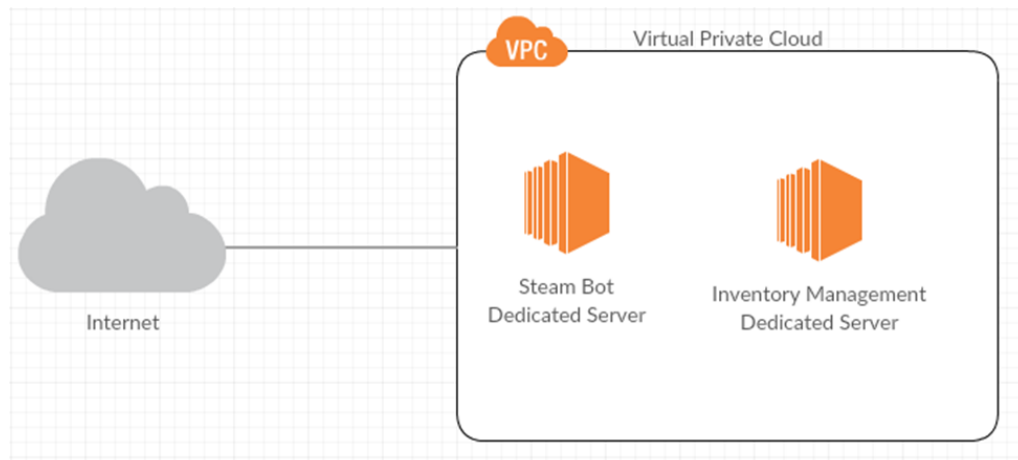


- Inventory Manager (Java)
 - Java Coded Server handling virtual item inventories
- AWS SQS (Java API)
 - Amazon Web Services message queueing system
- AWS DynamoDB (Java API)

- Amazon Web Services NoSQL Database
- Steam Bot (C#)
 - C# Steam Bot that will accept trades and perform trades.
- Web-Server (Java Tomcat)
 - Tomcat java webserver
- Web-Browser
 - Modern HTML5 ready browser (Chrome, Firefox)

HARDWARE ARCHITECTURE

High-Level Server Architecture Diagram



- The diagram features two servers running one program each.
- The first server the “Steam Bot”, will run a Steam Bot that will handle the automated creation of trade offers for the platform users.
- The second server will handle the user’s inventory and will work autonomously from the Steam Bot.

SOFTWARE ARCHITECTURE

Steam Bot

- C# Steam Bot that will accept trades and perform trades.

Inventory Manager

- Java Coded Server handling virtual item inventories

Security Architecture

Personal Information Protection

1. All user personal information will be stored in a secure database.
2. Passwords will not be stored, only the hashed equivalent.
3. SSL connections will be forced for all users.
4. The system should automatically block IP addresses that attempt to brute force user accounts.

Virtual Item Protection

1. Steam API requests should be transmitted using SSL.
2. The Steam account holding the items should use very complex passwords.
3. A “safe word” should be used by the trading bot to indicate that is in fact the bot and not an impersonator.

APPLICATION PROGRAM INTERFACES

Steam WEB API:

GetTradeOffers (v1)

- This API gets a list of trade offers (up to a maximum of 500 sent or 1000 received) for the account associated with the WebAPI key.
- This will be used to view all trade offers that have been sent to the platform users, it will help with keeping track of trades and if a trade has been accepted or if a trade has timed out.

GetTradeOffer (v1)

- This API gets details about a single trade offer. The trade offer must have been sent to or from the account associated with the WebAPI key.
- This may be used to look information about a single trade request, this will be used to check if a trade has been accepted.

CancelTradeOffer (v1)

- Cancel a trade offer that you sent. The trade offer must have been sent by the account associated with the WebAPI key.
- An expiry time of five minutes will be set on each trade offer sent. After five minutes passes, the status of the trade should be checked using “GetTradeOffer(v1)”.If the trade has not been accepted the trade should be cancelled using this API call.

More information: https://developer.valvesoftware.com/wiki/Steam_Web_API/IEconService

An un-official website exists which keeps updated documentation on the APIs here:

<http://steamwebapi.azurewebsites.net/#interfaces>

STEAM CONDENSER (JAVA)

This Java library will be used heavily in the project it provides a very convenient API for querying the above steam Web API. It has support for retrieving user's inventory, all using java. It wraps all the API calls mentioned above for the steam API into easy to use Java methods.

Description:

“The Steam Condenser is a multi-language library for querying the Steam Community, Source and GoldSrc game servers as well as the Steam master servers. Currently it is implemented in Java, PHP and Ruby.”

GitHub: <https://github.com/koraktor/steam-condenser-java>

Requirements:

- Linux, MacOS X or Windows
- Java 7 or newer

AWS DynamoDB Java API

Listings all tables Java example:

```
DynamoDB dynamoDB = new DynamoDB(new AmazonDynamoDBClient(  
    new ProfileCredentialsProvider()));
```

```
TableCollection<ListTablesResult> tables = dynamoDB.listTables();
```

```
Iterator<Table> iterator = tables.iterator();
```

```
while (iterator.hasNext()) {  
    Table table = iterator.next();  
    System.out.println(table.getTable_name());  
}
```

<http://docs.aws.amazon.com/2015>

AWS SQS Java API

Sending an SQS message Java example:


```
AmazonSQS sqs = new AmazonSQSClient(credentials);
```

```
sqs.sendMessage(new SendMessageRequest(myQueueUrl, "This is my message text."));
```

[https://docs.aws.amazon.com, 2015](https://docs.aws.amazon.com/2015)

USER INTERFACE DESIGN

The user interface will be rendered by a web-server for the user to consume in a web browser. The GUI will be responsive to the size of the screen the user is using. Using bootstrap will enable responsive support when using a row & column structure. I plan to design the UI using some form of material design style.

The mock-ups below may not look like the finished product, but I will use it as a guide when creating the web pages. I don't believe a lot will change but I may add a navigation bar along the top instead of the banner.

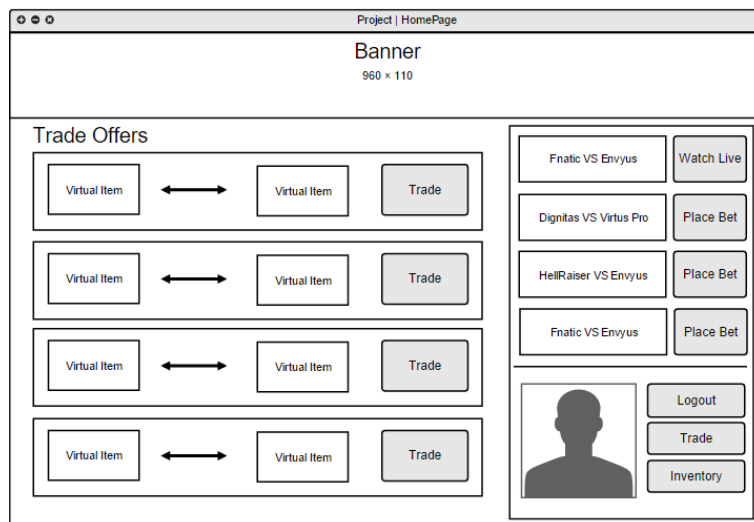
I will be using bootstrap and it will be rendered by a tomcat web server. The web-server will use a MVC (Model View Controller) pattern to render dynamic web pages.

MATERIAL DESIGN

- <http://www.material-ui.com> – Google's Responsive Material design framework
- <http://materializecss.com> - Modern responsive front-end framework based on Material Design

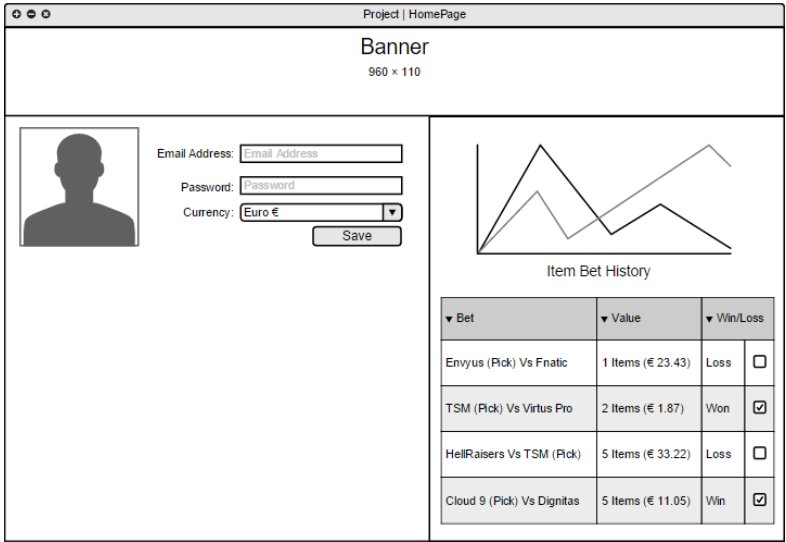
HOMEPAGE MOCKUP

This will be the user's homepage if they are logged in. The alternative version will be the same except the bottom right user panel, will have login & registration buttons.



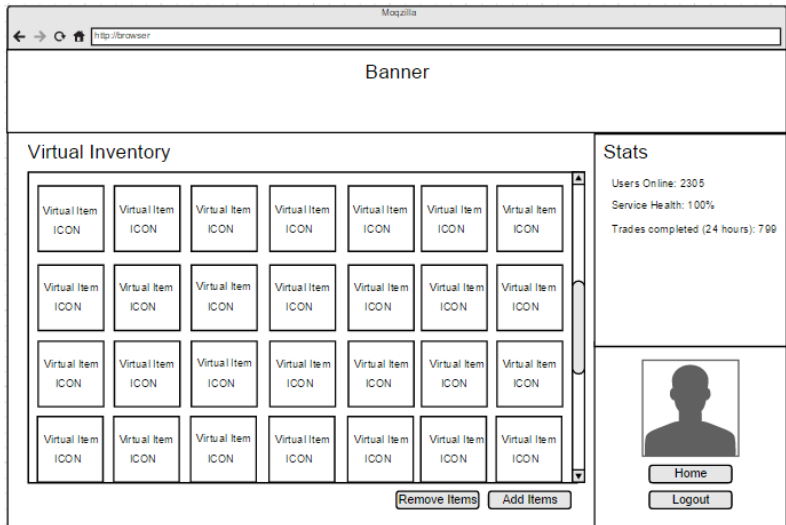
PROFILE MOCKUP

This is the user profile page, here the user can view their profile. Past bets will also be displayed as well a graph will display the users betting history statistics.



VIRTUAL INVENTORY MOCKUP

This page will show the user all the items they have in their virtual inventory. They can select multiple items by clicking on the item icons and then selecting an action below. The service stats are also shown to the user on the right sidebar.



Appendix A: References

The following table summarizes the documents referenced in this document.

Document Name and Version	Description	Location
<i>Project Requirements Spec</i>	The document outlines the project requirements, including functional and non-functional requirements as well as interface requirements.	<i>Moodle Submission & available by email.</i>
<i>Project Proposal</i>	The document outlines the overall scope of the project, the idea for the project as well as various other features about the project.	<i>Moodle Submission & available by email.</i>

APPENDIX B: KEY TERMS

The following table provides definitions for terms relevant to this document.

Term	Definition
<i>Steam</i>	Online gaming platform, selling downloadable games, and contains virtual game items.
<i>Virtual Game Item</i>	An in-game tradable virtual game item with real monetary value, that can be traded between Steam users.
<i>Trade Offer</i>	Users can post an “offer” to another user with items they would like to exchange.
<i>AWS</i>	Amazon Web Services (Cloud based web services that will be used to support the project)

MONTHLY JOURNALS

REFLECTIVE JOURNAL SEPTEMBER

Student name: Thomas Boyle

Programme BSHC4 CC

Month: Sept

MY ACHIEVEMENTS

Submitted the first project proposal for my project. It is not the final proposal for my project. The first idea was a simple end-to-end voice encryption app. It listed some cloud Amazon Web Services technologies as requirements. I do plan to use some of AWS's services that they offer to help build the project. I needed an idea I liked so that I would be passionate about the project.

The end-to-end app would have certainly used AWS Simple notification service as well as SQS (Simple Queue Service) to manage all the encrypted messages being sent.

The second idea is nearly complete, I have a second project proposal ready to upload with a fresh idea instead. I have a good understanding of the platform I'll be using for my project.

This last week was spent finishing the original project proposal and then starting a second project proposal.

MY REFLECTION

Coming up with an idea was honestly the most difficult part this month. The project proposal did not take long to create after I had an idea.

I did get a good chance to brainstorm and get rid of a lot of rubbish ideas I originally had.

INTENDED CHANGES

I'll pick up the pace with the project now. I had trouble coming up with an idea, especially an idea I wanted to work on.

Now that I have the new project proposal done, I'll arrange a supervisor meeting as soon as possible. Getting external feedback on the idea will be a great help, and should let me know if the project is feasible.

SUPERVISOR MEETINGS

None yet, will try to arrange a meeting for the following Monday and I will discuss the revised project proposal & plan.

Date of Meeting:

Items discussed:

Action Items:

REFLECTIVE JOURNAL OCTOBER

Student name: x12410922 (Thomas Boyle)

Programme: BSHC4

Month: October

MY ACHIEVEMENTS

This month, I was able to complete the first version of my requirements spec. This document took a lot of effort as I had spent numerous hours working on it. It still requires a few updates before I'm happy with it overall. The document contains over 10 functional use-cases. It has several UI mock-ups for the website interface. The use cases are covered in great detail and provide a good overview of how each use case functions for the different actors within the system.

The document outlines several technologies and their use in the project. It also includes libraries and other services that project will use such as AWS & steambot the open source steam trading bot.

This month was mainly focused towards the requirements spec.

MY REFLECTION

I felt I had done enough work for the requirements spec.

However, it is not completed as a whole and I will continue to refine it over the course of November.

INTENDED CHANGES

Next month, I will update the project spec with more information and expand on detailing some of the previous points. I realised that I need to expand further on the use classes. I will be working on the project interface and the back-end trading. I will do some trading tests and begin focusing on getting the underlying system working before creating the front-end of the website. I will create the website html bootstrap code but I will not focus on getting that functionally working this month, rather I will focus on the back-end coding of the platform.

SUPERVISOR MEETINGS

Date of Meeting: 27/08/2015

Items discussed: Project proposal, small talk about Requirements spec

Action Items: Starting the requirements spec

Date of Meeting: 03/11/2015

Items discussed: Requirements spec

Action Items: Submitting the requirements spec and getting drafts reviewed by the supervisor.

REFLECTIVE JOURNAL NOVEMBER

Student name: x12410922 (Thomas Boyle)

Programme: BSHC4

Month: November

MY ACHIEVEMENTS

- Early November, first draft of the requirements specification completed & submitted.
- Late November, finishing the high level design and analysis documentation.

No dev work took place this month and I solely choose to focus on documentation for the project.

I will need to have both documents peer reviewed by my supervisor at some stage. (At least before the mid-point)

MY REFLECTION

The high level design and analysis documentation is still weak at best and will require more refactoring and updating to fully complete it.

INTENDED CHANGES

December should be a development month in preparation for the mid-point presentation.

Next month, I will update the project spec with more information and expand on detailing some of the previous points. I realised that I need to expand further on the use classes. I will begin work on the project interface and the back-end trading. I will do some trading tests and begin focusing on getting the underline system working before creating the front-end of the website. I will create the website html bootstrap code but I will not focus on getting that functionally working this month, rather I will focus on the back-end coding of the platform.

SUPERVISOR MEETINGS

Date of Meeting: 03/11/2015

Items discussed: Requirements spec

Action Items: Submitting the requirements spec and getting drafts reviewed by the supervisor.

REFLECTIVE JOURNAL DECEMBER

Student name: x12410922 (Thomas Boyle)



Programme: BSHC4

Month: December

MY ACHIEVEMENTS

- Early December:
 - Created high level technical architecture diagram to aid with the project development.
 - Started a to-do list, using www.todoist.com
 - Began development work on the project and used todoist to keep track of all work done.
- Late December, some code refactoring and testing.

TO-DO LIST DEVELOPMENT TASKS (COMPLETED):

<input checked="" type="checkbox"/>	Completed Trades, add items to user virtual inventory.		
<input type="checkbox"/>	Activate SteamAuthenticator on All new accounts, and premium the accounts.	4 Jan	of
<input type="checkbox"/>	API Market Pricing  1		of
<input type="checkbox"/>	Figure out why ProfileCredentialsProvider dosent work		of
<input type="checkbox"/>	Send BotHealth checks every x minutes  1		of
<input type="checkbox"/>	Cross Reference BotHealth checks with bot profiles		of
<input type="checkbox"/>	Track Completed-Trades in DB		of
<input type="checkbox"/>	Add file based configuration for, dev, prod etc.		of
<input type="checkbox"/>	Virtual Inventory manager, should save snapshot of bot's inventory and reserve items, so that asset ids are guaranteed to be removed from the bot's inventory without error.		of
<input type="checkbox"/>	decide how jobs should be sent to the manager, and how jobs will be timed-out, how will the front-end integrate?		of
<input type="checkbox"/>	Trade offers should use tokens.		of
<input type="checkbox"/>	Auto-elect bot, based on health, inventory space.		of
<input type="checkbox"/>	Complete first refactoring		of

TO-DO LIST (NOT COMPLETED)

MY REFLECTION

I did not do as much development work as I wanted during December and the holiday season slowed down my progress but I had to balance between project work and exam studying.

INTENDED CHANGES

I intend to complete all of the non-completed “to-dos” by the end of February before the prototype mid-point presentation.

SUPERVISOR MEETINGS

Date of Meeting: 06/12/2015 (Unsure exactly of date)

Items discussed: Development work for December, preparing for the mid-point

Action Items: Start development work immediately, so that exams are not interfering with progress and that a prototype will be ready for February.

REFLECTIVE JOURNAL JANUARY

Student name: x12410922 (Thomas Boyle)

Programme: BSHC4 Cloud Computing

Month: January

MY ACHIEVEMENTS

Early January

Finished remaining to-do items on the project list for the SteamBotManager [Java Component].

- Using www.todoist.com

I have created 4 development bot accounts that are used to store items for development/testing purposes, this gives a total of 4000 slots. I will create more bot accounts before the site goes live.

Basic UI connected to the back-end.

Mid-January

Technical report document started.

Late January

Finished my mid-point PowerPoint presentation slides. Had a meeting with my Supervisor and improved the slides as a result of this meeting.

Submitted Technical Report document.

MY REFLECTION

I’ve been late twice on previous journal submission, I feel bad about this but I intend to upload the journals the day before each deadline from now on. I didn’t understand the importance of them at first but now I see how they are useful.

INTENDED CHANGES

I intend to upload the journals before each deadline from now on.

I will need to finish the front-end work before implementing the betting features.

SUPERVISOR MEETINGS

Date of Meeting: 03/02/2015

Items discussed: Presentation check before mid-point

Action Items: Small tweaking to my presentation power point slides, discussed various mid-point requirements.

REFLECTIVE JOURNAL

Student name: x12410922 (Thomas Boyle)

Programme: BSHC4 Cloud Computing

Month: February

MY ACHIEVEMENTS

Early February

Midpoint presentation draft review, completed.

Mid-February

Midpoint presentation, completed. The presentation went smooth and was received well. I am happy with the outcome.

Late February

Some development work took place but this month towards the end, however I focused more on other projects that required my attention.

MY REFLECTION

More development works needs to be completed to remain on track for the project deadline. I'm very happy with my grade for the midpoint presentation.

INTENDED CHANGES

I will continue to add more tasks to todist.com and will complete all of the current tasks before the end of March. All current development to-do tasks will be completed by the end of reading week.

SUPERVISOR MEETINGS

Date of Meeting: 03/02/2015

Items discussed: Presentation check before mid-point

Action Items: Small tweaking to my presentation power point slides, discussed various mid-point requirements.

Date of Meeting: 19/02/2015

Items discussed: Mid-Point recap

Action Items: Ensure I don't slack off with development work.

REFLECTIVE JOURNAL

Student name: x12410922 (Thomas Boyle)

Programme: BSHC4 Cloud Computing

Month: March

MY ACHIEVEMENTS

Early March


Completed all of the current todoist tasks:

Yesterday

Auto-elect bot, based on health, inventory space.  1 Final College Project ● 13:42

4 days ago

Send BotHealth checks every x minutes  1 Final College Project ● 17:15

Virtual inventory manager, should save snapshot of bot's inventory and reserve items, so that asset ids are guaranteed to be removed from the bot's inventory without error.  1 Final College Project ● 17:14


10 days ago

decide how jobs should be sent to the manager, and how jobs will be timed-out, how will the front-end integrate? Final College Project ● 18:42

12 days ago

Trade offers should use tokens. Final College Project ● 23:27

Self ● 23:20

Activate SteamAuthenticator on All new accounts, and premium the accounts.  1 Final College Project ● 23:20

Mid- March

Adding additional todoist tasks. Starting to integrate front-end with back-end.

Late March

The UI for the website is coming along nicely. I have all the styling completed and all the pages have been completed (not all pages are functioning yet).

MY REFLECTION

More development works needs to be completed to remain on track for the project deadline. I'm very happy with my grade for the midpoint presentation.

INTENDED CHANGES

I will continue to add more tasks to todist.com and will complete all of the current tasks before the end of March. I need to finalize the project for the project code submission and final demo.

The project is in a stable state but needs some polishing.

SUPERVISOR MEETINGS

Date of Meeting: 18/04/2015

Items discussed: Quick update on project

Action Items: Finish project, send report draft and final presentation draft

GLOSSARY OF TERMS

TERM	DEFINITION
Steam	Steam guarantees instant access to more than 1,800 game titles and connects its 35 million active users to each other—and to us. Through Steam, fans can easily buy, play, share, modify, and build communities around Valve products as well as titles from other independent game studios. Steam is available in 237 countries and 21 different languages.
SteamBot	SteamBot is C# library hosted here: https://github.com/JesseCar96/SteamBot
JSON	JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.
Virtual Item	A virtual item is typically a game item with no physical existence, and is tied to a game via the game's platform. Steam for example has support for virtual game items.
Trade Offer	Users can post an "offer" to another user with items they would like to exchange.
AWS	Amazon Web Services (Cloud based web services that will be used to support the project)
DynamoDB	An Amazon NoSQL cloud database service
SQS	An Amazon cloud hosted message queue service
Match Betting	Matches are essentially CSGO competitive games taking place between two teams in a given tournament.
PlayFramework	An asynchronous MVC framework for Java & Scala
CSGO	Counter strike : Global Offensive (Video Game)

