Software Requirements Specification

for

Second Screen Experience

Version 1.0 approved

Prepared by Connor Brooks,

Matthew Clark,

Christopher Goulet,

And Jared Prince

Western Kentucky University

2/10/2017

Table of Contents

Table of Contents ii

Revision History ii

1 Introduction 1

1.1 Purpose 1

1.2 Intended Audience and Reading Suggestions 1

1.3 Product Scope 1

1.4 References 1

2 Overall Description 2

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 3

2.6 User Documentation 3

2.7 Assumptions and Dependencies 3

3 External Interface Requirements 3

3.1 User Interfaces 3

3.2 Hardware Interfaces 3

3.3 Software Interfaces 4

3.4 Communications Interfaces 4

4 System Features 6

4.1 Hosting a Game 6

4.2 Joining a Game 6

4.3 Creating a New Game 7

5 Other Nonfunctional Requirements 7

5.1 Performance Requirements 7

5.2 Safety Requirements 7

5.3 Security Requirements 8

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

This system requirements specification is for all elements of Party Screen version 1.0, including the web client, web server, game server, games, and game and controller APIs.

## Intended Audience and Reading Suggestions

This document is intended for three different readers: users of the website, game developers using our API, and back-end developers using our system. The first two groups should refer to the system features (Section 4), while the back-end developers should focus more on the overall description (Section 2) and the external interface requirements (Section 3).

## Product Scope

### Front-End

The front-end software consists of the game API and individual game modes.  These are intended to provide the user with a platform to upload, create, and play online multiplayer games and allow multiple users to join simultaneously on a single computer screen.  Players use their smartphones, tablets, and other computers as a controller. The game and controller APIs allow users to build their own games and publish them to the website for others to enjoy.

### Back-End

The back-end software consists of the web server and game server.  The web server will allow the user easy navigation and requesting of game “rooms”, while the game server will handle routing of data between the second screen host and the controllers.  Both of these systems will be designed in a way to minimize communication delay to decrease lag between a player’s actions and what they see on the screen.

## References

### WebSocket Protocol

Internet Engineering Task Force (IETF), 2011; <https://tools.ietf.org/html/rfc6455>

### json11

Dropbox, Inc., 2013; <https://github.com/dropbox/json11>

### SHA-1 implementation in C++

Steve Reid, Bruce Guenter, Volker Grabsch, Eugene Hopkinson, 2015; <https://github.com/vog/sha1>

# Overall Description

## Product Perspective

This product is a replacement for regular party game systems.  In general party game systems require multiple controllers and having to manually set up ports, static IP addresses, and open/forward ports on a computer to get everything connected.  If using a gaming console, this requires a large upfront cost of the gaming console, controllers, and games; all of which are proprietary to the console.  Using our system, players can use devices that they usually have with them (tablets, smartphones, or practically any other computer), and connect to a webpage that handles setting up connections for them.  It will also allow them to play and create non-proprietary games without any large upfront costs.

## Product Functions

* Front-End (General Users)
  + Host a Game
  + Join a Game
  + Create/Upload New Game
* Back-End (Developers)
  + Pass controller information to host screen
  + Create necessary game “rooms”

## User Classes and Characteristics

* Game hosts & players - **Most Important**
  + Host and play the actual games
  + Use smartphones, tablets, other computers to control and display the game
* Game developers
  + Use the game API to create new games for the hosts and players
* System Developers
  + Set the servers and system up for users

## Operating Environment

This system will be tested and operate on a Linux machine with Apache and the PHP server. Users can connect with practically any device within the last 10 years that can access the internet and has JavaScript and HTML5 capabilities.

## Design and Implementation Constraints

The main limitations of this system are bandwidth and Internet speed. Having multiple devices simultaneously sending and receiving data through our game server will require a larger infrastructure based on how many players need to play concurrently. Also, memory and CPU requirements have to be taken into account for host screens and game optimization for those running on it. The third limitation is there being no standard yet implemented for sending controller data over UDP, a large standard for fast-paced, online games.

## User Documentation

Along with this software system, an online guide will be created for the game API and steps to join/host games.

## Assumptions and Dependencies

This system requires the SHA1 C++ library to complete the WebSocket handshake, and json11 library to help parse data from the web server, host screen, and controllers.

# External Interface Requirements

## User Interfaces

The menu interface has several screens: the main menu, the host menu, the join menu, and the game menu. The main menu has buttons directing to each of the other menus. These menus have buttons to return to the main menu. The host menu shows options for hosting a game and the join menu allows the user to join a specific game. The game menu displays an information page for each game. This page includes the title of the game, a gallery of game photos, a description, and instructions and controls. Each game must have a user interface to act as the controller, and must display the game itself. The game and controller interfaces are left up to the developer, but they must each be fixed to the size of the screen on which they are displayed.

## Hardware Interfaces

### Web/Game Server

This server needs access to the internet over ports 80 and 443, and needs enough processing power to host the web server, PHP server, and game server. The game server needs port 2000 open and port 3000 and up open depending on how many games need to be hosted. The upper limit can change based on how many games the server can handle. The bandwidth required will probably need to be at least 10MB/s for a small user base (<100) to stay connected and transmit controller and game information. The game server won’t need much more RAM than normal since it will just forward controller data to the hosting screen. For the CPU, this is dependent on how many games will be hosted, and more testing will be needed. During initial development, our recommendation will be at least a dual-core processor at 2GHz to utilize our multi-threading more.

The game server and web server will communicate over a simple socket connection with POST variables. The web server communicates with the screen and controller clients via AJAX requests and HTTP connections to deliver game server information. The game server communicates with the screen and controllers via WebSocket protocol.

### Screen/Controller Clients

The screen client will require a web browser that has access to HTML5, CSS3, and JavaScript. The controller will require a web browser that just has access to CSS3 and JavaScript. Both have to be able to connect to the internet over ports 80, 443, and 3000+. The upper bound on port 3000 will depend on the max amount of games the server can host. The hosting screen will need a computer able to process rendering a game screen in real-time and controlling game objects; without any other programs running, a single-core processor at 2GHz will be enough for the least performance-intensive games. The controller clients will generally be optimized for smartphones since they mimic a game controller the easiest. Once controllers are connected, game data gets sent through the game server as JSON to the hosting screen. The game API will determine how to process this data.

## Software Interfaces

### PHP 5.6

PHP will be the interface between connecting to the game server to create a new game “room” and return room data for new room or requested room to join. Both sending data and receiving data will be in the form of a JSON object to allow easy parsing of necessary data. PHP will be used since it comes with most Apache servers, has easy creation and socket connection, and is widely used in case troubleshooting is necessary.

### Apache 2.4

This service will deliver HTML, CSS, and JavaScript files so users can play and host the games. We use this software since it is widely used, lightweight, and easy to set up.

### Ubuntu 16.04

We will primarily be testing and implementing our system using this operating system, but pretty much any Linux server will be able to host this system. We are using this system since it is very lightweight, leaving more system resources just for the game and web servers.

### json11

This library will be used for the game server to parse data from the controllers, host screen game info, and PHP server that is requesting game room information. Using this library will decrease production time since we can focus on the core requirements of the server without worrying about how to parse other component data.

## Communications Interfaces

Since the Second Screen Experience is a game website, the main communication interfaces are based around web standards. The main one is the use of the Hypertext Transfer Protocol via a web browser, in our case apache with PHP 7 installed alongside it. To keep the connection secure and to make sure the client knows they are connecting to the right server, we have choose to use the HTTPS standard. For the clients this means it requires the use of a modern web browser that has support for canvas output and WebSocket communications.

For the non-game side of the website, the client connects directly to our web server to request pages. For forums, we use a Representational state transfer (REST) standard that is used by almost all websites for requesting and posting data. We try to use AJAX (Asynchronous JavaScript and XML) so that it doesn’t interrupt the user experience when the user has to send these requests. An example of this would be if the client submitted a forum. To respond to these request, exists some php scripts that trigger when a request is sent and respond back. Message passing throughout the system is done with JSON message formatting.

**

***Figure: Example of JSON content and formatting***

The reason why we chose to use JSON as the formatting of the messages, is that it can represent complicated data in a reasonable amount of space and still be human readable if needed to be debugged.

The following graphic shows the three basic components of our system. To create or connect to a room, the client connects to the web server, which then requests room data from the game server. The game server returns the room data (game server port, room code, etc.), and the web server passes that information to the client. From then on, the client and the game server communicate directly on the game room specific port.

*XNVCXMNJVjA424Qutn3kH6_btuvQChLMXzNpd1_bUHG7yn7W1RIg-5jqj9cNFC0_16IjD6nLhtIlGxXBWUhdCU4C3KBb8dluVYFB5f0LhX2xNDehYNWsHu-tFYvJZz-Nila0COCi*

# System Features

## Hosting a Game

### Description and Priority

The user must be able to start a game and have their device act as the main game screen. He/she must be able to provide a code to specific users which will allow them to join the game. This is a high priority feature.

### Stimulus/Response Sequences

The user will select “Host a Game” from the main menu, then choose a game and select “Private”. The system will provide a unique code for the game. They will then select “Start Game”.

### Functional Requirements

REQ-1: User screen becomes game screen

REQ-2: Login authorization code generated

REQ-3: New game instance created

## Joining a Game

### Description and Priority

The user must be able to join a specific game with a code provided by the host. This is a high priority feature.

### Stimulus/Response Sequences

The user will select “Join a Game” from the main menu. Then they will enter a code provided by another user into the space provided and submit. The server will then connect them to the host’s game.

### Functional Requirements

REQ-1: User able to successfully join existing game with proper code

REQ-2: Notification to user when game code is not valid

## Creating a New Game

### Description and Priority

Developers are able to create their own games for users to play using the platform’s API. This is a high priority feature.

### Stimulus/Response Sequences

The API will be provided to developers. Developers will be able to upload games for submission. Once the game has been approved, it will appear in the game menu as a possible selections for users to play.

### Functional Requirements

REQ-1: Submission of new game

REQ-2: Adding new game to queue to be verified by authorized user

REQ-3: Authorized user able to approve/delete game

REQ-4: Notification of developer when game has been approved/disapproved

REQ-5: Adding new game to menu for user selection once approved

# Other Nonfunctional Requirements

## Performance Requirements

Because this product connects multiple users to a multiplayer game, the communication between the external devices and the server must be very quick. Any communication delays or interruptions will negatively impact the user experience. Our goal for input delay from the controller to the host screen is to be below 100ms at all times.

## Safety Requirements

The largest safety requirement that our system will have is a system in place to reduce risk of seizures from flashing lights in our games. Since we can’t regulate what users create, we want to minimize the risk of harm by displaying a notification of some kind for new users that games might cause seizures.

## Security Requirements

### General Requirements

In order to provide a friendly environment between people in the same group, our system will have a 5 digit code randomized for each room, and a UUID for the host to connect to a room. The room code allows only players who know this code to connect to a specific port, and the UUID makes it so only the host screen that requested a new room can join as the host. The latter is important to keep the same screen operational at all times and make it so no other users can modify game settings to cheat.

### Denial of Service

In order to reduce DoS attacks, the game server will blacklist an IP address from connecting to a port if they have repeatedly tried to connect within a short period of time.