**Software Design  
Document**

for

Centipede Army Checkers

**Version 1.0 approved**

**Prepared by Adrienne Bowman,   
Cynthia Cordova, Rogelio Garza,   
Jacob Gibson, Brent Redmon**

**Centipede Army**

**October 29th 2018**

# Table of Contents

**Table Of Contents 1**

**Revisions 2**

**1 Introduction 3**

1.1 Purpose 3

1.2 System Overview 3

1.3 Definitions, Acronyms and Abbreviations 3

1.4 Supporting Materials 3

1.5 Document Overview 4

**2 Architecture 4**

2.1 Overview 4

2.2 NGROK Component 5

2.3 Server Component 6

2.4 Socket Component 6

2.5 Client Component 6

2.6 Twilio Component 7

2.7 Router Component 7

2.8 API Component 8

**3 High-Level Design 9**

3.1 View / Model Component 1..n 9

3.2 Subscriber Sequence Diagram 10

3.3 Client Component Diagram 10

3.4 Twillo Component Diagram 12

## 

# 

# Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Primary Author(s)** | **Description of Version** | **Date Completed** |
| 1.0 | Brent R.  Cynthia C. Adrienne B.  Rogelio G. | Created Diagrams and Detailed Functionality | 10/28/2018 |
| 0.75 | Rogelio G.  Jacob G. | Fix formatting and layout | 10/28/2018 |
| 0.5 | Brent R. | This is the initial version of the SDD. Note that the implemented drawings and models may be subject to change as more versions of this document are released. | 10/07/2018 |

# Introduction

## Purpose

The purpose of this document is to specify the relationship between each component in the centipede army checkers application, and to show implementations of each requirement as specified in the SRS. This document will also serve as a guideline for the developers assigned to work on this project, such that that have a basis with which to implement our designs.

## System Overview

Centipede Army Checkers shall have two components, the front end client and the backend server. The front end will be written in HTML, CSS and Javascript, and the back end will be written with the latest version of NodeJS (server side asynchronous Javascript). The relationship between the server and client will be maintained via the Socket protocol, which enables bidirectional persistent communication between the server and client. The server also will have a medium of sending SMS messages to clients updating them on who is currently winning multiplayer matches through the Twilio API. The basis of this mechanism will derive from the Observer/Observable pattern, in which users who want to receive updates must subscribe to receive such updates.

## Definitions, Acronyms, and Abbreviations

1. NGROK - the program that allows us to tunnel our localhost to the Web
2. CAC - Centipede Army Checkers
3. LAN - Local Area Network
4. SMS - Short Message Service
5. NPM - Node Package Manager
6. Client - a user who connects to the CAC server
7. API - Application Programming Interface

## Supporting Materials

1. SocketIO reference documentation
   1. <https://socket.io/docs/>
2. Twilio SMS API
   1. <https://www.twilio.com/docs/sms>
3. NGROK localhost tunneling API
   1. <https://ngrok.com/docs>
4. Express documentation
   1. <https://expressjs.com/en/4x/api.html>
5. HTML Canvas documentation
   1. <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API>

## Document Overview

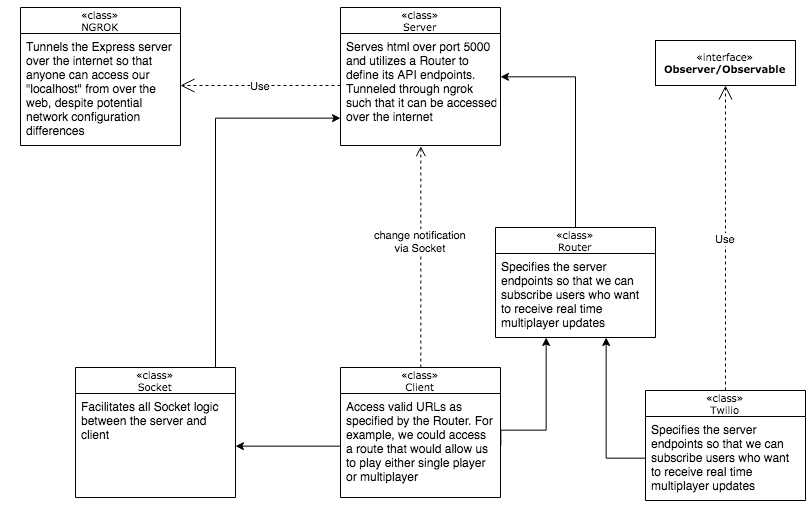
The next section, Architecture, of this document gives an overview of the functionality of the product. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next section. The third section, High-Level Design, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product. It also gives a more detailed breakdown of the architectural components.

# Architecture

This section includes many different diagrams and descriptions of our system’s core components.

# Overview

This top level design of our application shows all the higher order components that comprise of the system as well as their relationships and interactions. We will focus on each individual component and go into detail about how they work and their role in the application

**

# NGROK Component

This is arguably the highest order dependency that Centipede Army Checkers relies on. Because this is a web based application, we must consider the environments that this application would run in, and acknowledge that different environments will not all have the same network configurations. This can lead to problems when trying to access the server as a client from a different machine on the same network (ie - not from the localhost). To remedy this problem, we must instill a network protocol/group policy such that all server endpoints are accessed in the same fashion.

The NGROK application scans the computer that is running the Centipede Army Checkers server and tunnels it to the NGROK servers. For example, if we ran our Centipede Army Checkers server over port 5000 on our local machine, then we can run this command in NGROK:



then NGROK will allow us to access the localhost port 5000 on any computer running the Centipede Army Checkers server at [*https://www.centipedecheckers.ngrok.io*](https://www.centipedecheckers.ngrok.com). From a network perspective, it looks like we are just accessing a regular website. In reality, we are emulating a LAN environment over the web so that all files can be accessed in the same manner from the same domain.

# Server Component

Our localhost server will be created by a library called Express, which can be downloaded via the NPM database. Its primary purpose is to house our server, which, in and of itself, shall be an action listener. It listens for a user to make a request, and then serves or denies that request accordingly. The server will serve web pages over port 5000, even though in reality this port will be tunneled to an external server via NGROK. Additionally, the server will print to the console when a user has connected, and when a user has disconnected.

# Socket Component

Our application will take advantage of the Socket protocol to establish persistent bidirectional communication between each client and the server. Additionally, the Socket class, where the Socket protocol is implemented, will be able to differentiate between the different roles that the client could be (refer to section 2.5 for further details about Client roles). The Socket class will not allow more than 2 people to join the multiplayer lobby, but allow an unlimited amount of spectators to view the match. Note that spectators and players in the lobby utilize the same socket implemented server side, so clients who wish to utilize the Socket must specify who they are upon connecting. This can be implemented as such:



We see that along with telling the client where to find the Socket endpoint, we also need to send an object as well, which specifies the role of the client. If the client is a spectator, then a socket connection shall be established, but the client will only be served a webpage that allows them to view the match, rather than participating in it.

# Client Component

In CAC, the client can be one of four roles:

* Single Player: If a client accesses the home directory of the CAC server, then they will obtain the role of a Single Player. This client does not utilize a socket connection, and is only fed a checkers game that is processed client side. No communication will be necessary between the Single Player and the Server after the initial GET request.
* Lobby Player: If a client accesses the */multiplayer* directory of the CAC server, and the lobby is not full (ie - comprised of two players, which is the maximum amount of players that it can hold), then they will assume the role of a Lobby player. This role will require the utilization of a socket connection. This is because each move that a player makes needs to be seen by the other player in the lobby. If a Lobby Player makes a valid move, then their turn will end, the board will send its state to the server, and the server will broadcast the new board state to the other client.
* Spectator: If a client accesses the */spectator* directory of the CAC server, then they will assume the role of a Spectator. A socket connection will be utilized and the user shall be served a web page that shows the multiplayer game in real time. This user, however, shall not be able to contribute to the game, as they are only a spectator, not a Lobby Player.
* Subscriber: If a client access the */subscribe* directory of the CAC server, then the server shall present this client with a web page that consists of a label that states “**Enter your number to subscribe or unsubscribe to real time text updates”,** a corresponding text box, and two buttons. One that says **Subscribe**, and another that says **Unsubscribe**. If the client types in their phone number and presses submit, then they will assume the role of a Subscriber. At this point, the Twilio class (see section 2.6 for more details) will add their number to an existing list of numbers. The client will then receive text messages consisting of real time updates pertaining to the match so long as they have service on their phone and are in a place with good enough cell phone reception to receive text messages.

# Twilio Component

The CAC application will implement the Twilio API inside a Twilio class, which handles all the logic pertaining to sending SMS messages to subscribing clients. Each time a Lobby Player scores a point, Sockets will call the Twilio class and request that it push a notification to all of its subscribers, alerting them of who scored against who and what the current score is. Then when the game ends, Sockets will query the Twilio class to send a notification all subscribers that alerts them of who won the match. Conversely, a user can access the */subscribe* route, type in their number, and unsubscribe from the list of subscribers.

# Router Component

The CAC server needs to know what information to send to connecting clients depending on the directory or API that a client accesses. This is where the Router class comes into play. The Router class is included in the Express server library, and is designed to specify what happens when clients access different directories. For each client role that we have, which is four, the Router class shall provide that many different routes that users can access.

Here are the valid routes that clients shall be able to access upon querying the CAC server:

* *“/”*: This will be the home directory. If a client accesses this directory, then the Router will serve the client the single player web pages.
* *“/multiplayer”*: This will be the multiplayer directory. If a client accesses this directory, then the Router will serve the client the Lobby Player web pages. Note that the Router has no context related to the existence of the Socket class. The Router will serve these pages blindly to the client, but it is up to the Socket class if the client who is accessing these web pages will remain connected. Refer to section 2.4 for more details.
* *“/spectator”*: This will be the spectator directory. If a client accesses this directory, then the Router will serve the client the spectator player web page. Though the Router has no relationship with the Socket class, it is important to note that all clients that are spectators will never have their socket connection denied by the server. Assuming that a multiplayer match is in effect, two socket connections out of the entire pool of socket connections will be Lobby Players.
* *“/subscribe”*: This will be the subscribe directory. If a client accesses this directory, then the Router will serve the client the subscriber the webpage where they will be able to subscribe to real time text message updates or unsubscribe through the Twilio class (see section 2.6 for more details regarding Twilio).

# API Component

As part of the test plan, we will implement a RESTful API that allows for a 3rd entity to check:

* What port the server is running on
* To check to see if there are any socket connections present. Because we recommend that this check be tested right after the server is launched, there should be no sockets connected to it.
* To see if NGROK is live by checking to see if we are returned a valid IP after querying its servers
* Checking to see if a number that we input in the subscription page is being formatted correctly.

This API will be implemented through the Router() portio of the Express app, which is different from the Router described in section 2.7. The router for the API will determine what will be returned when querying particular API routes.

# High-Level Design

The following diagrams and descriptions describe our product’s actual implementation based on the function of its various components

## Spectator Sequence Diagram

****

The above diagram demonstrates the sequence of the user path to become a spectator They are able to view an ongoing match either when the game lobby is full or if the user chooses to view only.

## **3.2** Subscriber Sequence Diagram

****

The above diagram represents the sequence of the user’s path to become a subscriber. They will receive real-time updates of when each player scores and which player wins the match. The client retains the option of cancelling their subscription at any point in the match

## 3.3 Client Component Diagram

****

The above diagram demonstrates the activity flow of the client depending on which roles they choose. They are able to take on the roles of subscriber, spectator, or player. This diagram shows the path of each, with conditions that explain how the client will reach a particular outcome.

## 3.4 Twilio Component Diagram



The above diagram is to demonstrate how a subscriber receives SMS updates from the Twilio API. This component handles storing and deleting client phones numbers on the list of those who want to receive information about which players have scored, and ultimately who wins the match.