Assignment 4

CSCI131

Name: James Marino

Username: jm617

Student Number: 4720994

Table Of Contents

1. Executive Summary

This report aims to simulate a captcha image generator. It has functions to add background and target images, grouping the target images with tags as well as a generator which combines these target and background images to make a captcha. Please see code and results below.

See comments for details on structure and outline of functionality of program.

2. Body

2.1 C Program

Header and Implementation file for Base64 Conversions below.
See comments for details.

base64.h

```
/*
 * File:   base64.h
 * Author: Polarssl
 * See https://polarssl.org/api/base64_8h.html
 * Created on 4 December 2014, 12:18 PM
 */

#ifndef BASE64_H
#define    BASE64_H
#include <string.h>

#define POLARSSL_ERR_BASE64_BUFFER_TOO_SMALL               -0x002A
/**< Output buffer too small. */
#define POLARSSL_ERR_BASE64_INVALID_CHARACTER              -0x002C
/**< Invalid character in input. */

#ifdef     __cplusplus
extern "C" {
#endif


/**
 * \brief          Encode a buffer into base64 format
 *
 * \param dst      destination buffer
 * \param dlen     size of the buffer
 * \param src      source buffer
 * \param slen     amount of data to be encoded
 *
 * \return         0 if successful, or
POLARSSL_ERR_BASE64_BUFFER_TOO_SMALL.
 *                 *dlen is always updated to reflect the amount
 *                 of data that has (or would have) been written.
 *
 * \note           Call this function with *dlen = 0 to obtain the
 *                 required buffer size in *dlen
 */
int base64_encode( unsigned char *dst, size_t *dlen,
                   const unsigned char *src, size_t slen );

/**
 * \brief          Decode a base64-formatted buffer
 *
```

```
 * \param dst      destination buffer (can be NULL for checking
size)
 * \param dlen     size of the buffer
 * \param src      source buffer
 * \param slen     amount of data to be decoded
 *
 * \return         0 if successful,
POLARSSL_ERR_BASE64_BUFFER_TOO_SMALL, or
 *                 POLARSSL_ERR_BASE64_INVALID_CHARACTER if the
input data is
 *                 not correct. *dlen is always updated to reflect
the amount
 *                 of data that has (or would have) been written.
 *
 * \note           Call this function with *dst = NULL or *dlen =
0 to obtain
 *                 the required buffer size in *dlen
 */
int base64_decode( unsigned char *dst, size_t *dlen,
                   const unsigned char *src, size_t slen );

/**
 * \brief          Checkup routine
 *
 * \return         0 if successful, or 1 if the test failed
 */
int base64_self_test( int verbose );

#ifdef     __cplusplus
}
#endif

#endif     /* BASE64_H */
```

base64.c

```c
#include <inttypes.h>
#include <stdio.h>
#include "base64.h"



static const unsigned char base64_enc_map[64] =
{
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
    'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
    'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd',
    'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
    'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
    'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7',
    '8', '9', '+', '/'
};

static const unsigned char base64_dec_map[128] =
{
    127, 127, 127, 127, 127, 127, 127, 127, 127, 127,
    127, 127, 127, 127, 127, 127, 127, 127, 127, 127,
    127, 127, 127, 127, 127, 127, 127, 127, 127, 127,
    127, 127, 127, 127, 127, 127, 127, 127, 127, 127,
    127, 127, 127,  62, 127, 127, 127,  63,  52,  53,
     54,  55,  56,  57,  58,  59,  60,  61, 127, 127,
    127,  64, 127, 127, 127,   0,   1,   2,   3,   4,
      5,   6,   7,   8,   9,  10,  11,  12,  13,  14,
     15,  16,  17,  18,  19,  20,  21,  22,  23,  24,
     25, 127, 127, 127, 127, 127, 127,  26,  27,  28,
     29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
     39,  40,  41,  42,  43,  44,  45,  46,  47,  48,
     49,  50,  51, 127, 127, 127, 127, 127
};

/*
 * Encode a buffer into base64 format
 */
int base64_encode( unsigned char *dst, size_t *dlen,
                   const unsigned char *src, size_t slen )
{
    size_t i, n;
    int C1, C2, C3;
    unsigned char *p;

    if( slen == 0 )
        return( 0 );

    n = ( slen << 3 ) / 6;

    switch( ( slen << 3 ) - ( n * 6 ) )
```

```
    {
        case  2: n += 3; break;
        case  4: n += 2; break;
        default: break;
    }

    if( *dlen < n + 1 )
    {
        *dlen = n + 1;
        return( POLARSSL_ERR_BASE64_BUFFER_TOO_SMALL );
    }

    n = ( slen / 3 ) * 3;

    for( i = 0, p = dst; i < n; i += 3 )
    {
        C1 = *src++;
        C2 = *src++;
        C3 = *src++;

        *p++ = base64_enc_map[(C1 >> 2) & 0x3F];
        *p++ = base64_enc_map[(((C1 &  3) << 4) + (C2 >> 4)) &
0x3F];
        *p++ = base64_enc_map[(((C2 & 15) << 2) + (C3 >> 6)) &
0x3F];
        *p++ = base64_enc_map[C3 & 0x3F];
    }

    if( i < slen )
    {
        C1 = *src++;
        C2 = ( ( i + 1 ) < slen ) ? *src++ : 0;

        *p++ = base64_enc_map[(C1 >> 2) & 0x3F];
        *p++ = base64_enc_map[(((C1 & 3) << 4) + (C2 >> 4)) &
0x3F];

        if( ( i + 1 ) < slen )
             *p++ = base64_enc_map[((C2 & 15) << 2) & 0x3F];
        else *p++ = '=';

        *p++ = '=';
    }

    *dlen = p - dst;
    *p = 0;

    return( 0 );
}

/*
 * Decode a base64-formatted buffer
```

```
 */
int base64_decode( unsigned char *dst, size_t *dlen,
                   const unsigned char *src, size_t slen )
{
    size_t i, n;
    uint32_t j, x;
    unsigned char *p;

    /* First pass: check for validity and get output length */
    for( i = n = j = 0; i < slen; i++ )
    {
        /* Skip spaces before checking for EOL */
        x = 0;
        while( i < slen && src[i] == ' ' )
        {
            ++i;
            ++x;
        }

        /* Spaces at end of buffer are OK */
        if( i == slen )
            break;

        if( ( slen - i ) >= 2 &&
            src[i] == '\r' && src[i + 1] == '\n' )
            continue;

        if( src[i] == '\n' )
            continue;

        /* Space inside a line is an error */
        if( x != 0 )
            return( POLARSSL_ERR_BASE64_INVALID_CHARACTER );

        if( src[i] == '=' && ++j > 2 )
            return( POLARSSL_ERR_BASE64_INVALID_CHARACTER );

        if( src[i] > 127 || base64_dec_map[src[i]] == 127 )
            return( POLARSSL_ERR_BASE64_INVALID_CHARACTER );

        if( base64_dec_map[src[i]] < 64 && j != 0 )
            return( POLARSSL_ERR_BASE64_INVALID_CHARACTER );

        n++;
    }

    if( n == 0 )
        return( 0 );

    n = ( ( n * 6 ) + 7 ) >> 3;
    n -= j;
```

```
    if( dst == NULL || *dlen < n )
    {
        *dlen = n;
        return( POLARSSL_ERR_BASE64_BUFFER_TOO_SMALL );
    }

   for( j = 3, n = x = 0, p = dst; i > 0; i--, src++ )
   {
        if( *src == '\r' || *src == '\n' || *src == ' ' )
            continue;

        j -= ( base64_dec_map[*src] == 64 );
        x  = ( x << 6 ) | ( base64_dec_map[*src] & 0x3F );

        if( ++n == 4 )
        {
            n = 0;
            if( j > 0 ) *p++ = (unsigned char)( x >> 16 );
            if( j > 1 ) *p++ = (unsigned char)( x >>  8 );
            if( j > 2 ) *p++ = (unsigned char)( x       );
        }
    }

    *dlen = p - dst;

    return( 0 );
}



static const unsigned char base64_test_dec[64] =
{
    0x24, 0x48, 0x6E, 0x56, 0x87, 0x62, 0x5A, 0xBD,
    0xBF, 0x17, 0xD9, 0xA2, 0xC4, 0x17, 0x1A, 0x01,
    0x94, 0xED, 0x8F, 0x1E, 0x11, 0xB3, 0xD7, 0x09,
    0x0C, 0xB6, 0xE9, 0x10, 0x6F, 0x22, 0xEE, 0x13,
    0xCA, 0xB3, 0x07, 0x05, 0x76, 0xC9, 0xFA, 0x31,
    0x6C, 0x08, 0x34, 0xFF, 0x8D, 0xC2, 0x6C, 0x38,
    0x00, 0x43, 0xE9, 0x54, 0x97, 0xAF, 0x50, 0x4B,
    0xD1, 0x41, 0xBA, 0x95, 0x31, 0x5A, 0x0B, 0x97
};

static const unsigned char base64_test_enc[] =
    "JEhuVodiWr2/F9mixBcaAZTtjx4Rs9cJDLbpEG8i7hPK"
    "swcFdsn6MWwINP+Nwmw4AEPpVJevUEvRQbqVMVoLlw==";

/*
 * Checkup routine
 */
int base64_self_test( int verbose )
{
    size_t len;
```

```
    const unsigned char *src;
    unsigned char buffer[128];

    if( verbose != 0 )
        printf( "  Base64 encoding test: " );

    len = sizeof( buffer );
    src = base64_test_dec;

    if( base64_encode( buffer, &len, src, 64 ) != 0 ||
         memcmp( base64_test_enc, buffer, 88 ) != 0 )
    {
        if( verbose != 0 )
            printf( "failed\n" );

        return( 1 );
    }

    if( verbose != 0 )
        printf( "passed\n  Base64 decoding test: " );

    len = sizeof( buffer );
    src = base64_test_enc;

    if( base64_decode( buffer, &len, src, 88 ) != 0 ||
         memcmp( base64_test_dec, buffer, 64 ) != 0 )
    {
        if( verbose != 0 )
            printf( "failed\n" );

        return( 1 );
    }

    if( verbose != 0 )
        printf( "passed\n\n" );

    return( 0 );
}
```

Main functions for Captcha and helper functions below.
See comments for details.

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hiredis.h>
#include <gd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <regex.h>
#include "base64.h"

/*
 * Data
 */
static const char *HostName = "127.0.0.1";
static redisContext *Context;
static const int Port = 6379;
static const char *BackgroundDIR = "background";
static const char *TargetDIR = "target";

// Images
static const int MIN_IMAGE_TYPES = 5;
static const int MIN_EACH_TYPE = 3;

static const double BACKGROUND_WIDTH = 700.0;
static const double TARGET_WIDTH = 100.0;

// Redis groups
static const char *GRP_BACKGROUND_NAME = "bkgrdimgs";
static const char *GRP_TARGET_NAME = "targetname";
static const char *GRP_TARGET_COUNTER = "targetcnt";

/*
 * Definitions
 */
void menuSelect();
void setupConnection();
void exit();
void closeConnection();
void addBackground();
void addTarget();
void generatePuzzle();
gdImagePtr loadImage(char* fileName, char* fileType);
unsigned char* convertToBase64(gdImagePtr im);
void storeImageBackground(gdImagePtr image, const char
*fileExtension,
                          const char *dir, const char *fileName, const
char *groupName);
```

```
void setImage(unsigned char *image);
char* getImage(char *group);
void createDirectory(const char *directory);
void generateHTML(gdImagePtr imageMain, gdImagePtr looking);

int main(int argc, const char * argv[])
{
	/*
	 * Actual app
	 */
	// Setup Connection to DB
	setupConnection();

	// Create dirs
	createDirectory(BackgroundDIR);
	createDirectory(TargetDIR);

	// Run Menu
	menuSelect();

	// Close Connection
	closeConnection();
	/*
	 * END Actual app
	 */

    return 0;
}

/*
 * Utilities
 */
int generateRandomNumber(int minNum, int maxNum)
{
	int result = 0, low = 0, high = 0;

	if (minNum < maxNum) {
		low = minNum;
		high = maxNum + 1;
	} else {
		low = maxNum + 1;
		high = minNum;
	}

	struct timeval t1;
	gettimeofday(&t1, NULL);
	srand((int)t1.tv_usec * (int)t1.tv_sec);

	result = (rand()%(high-low)) + low;

	return result;
}
```

```
void generateRandomTargetGroups(int minNum, int maxNum, int
*randomArray)
{
	int result = 0, low = 0, high = 0, foundSpots = 1;
	int isDuplicate = 0, i = 0, j = 0;

	while (foundSpots == 1) {

		// Reset
		foundSpots = 0;
		isDuplicate = 0;

		if (minNum < maxNum) {
			low = minNum;
			high = maxNum + 1;
		} else {
			low = maxNum + 1;
			high = minNum;
		}

		srand((unsigned int)time(NULL));
		result = (rand()%(high-low)) + low;

		for (i = 0; i < maxNum; i++) {
			if (randomArray[i] == 0) {
				foundSpots = 1;
				break;
			}
		}

		for (j = 0; j < maxNum; j++) {
			// Check if random value already in list
			if (randomArray[j] == result) {
				isDuplicate = 1;
				break;
			}
		}

		if (foundSpots && !isDuplicate) {
			randomArray[i] = result;
		}
	}
}

char* getFileName(char *absoluteAddress)
{
	return strrchr(absoluteAddress, '/');
}

void createDirectory(const char *directory)
{
```

```
	struct stat current = {0};

	if (stat(directory, &current) == -1) {
		mkdir(directory, 0700);
	}
}

gdImagePtr blendTransparency(gdImagePtr image, int height)
{
	gdImageAlphaBlending(image, 0);

	int w = TARGET_WIDTH;
	int h = height;
	int h1 = h / 6;
	int h2 = 5 * h1;
	int l1 = w / 6;
	int l2 = 5 * 11;
	int row, col;

	for (row = 0; row < h; row++) {
		for (col = 0; col < w; col++) {
			int aval = 40;

			if ((row < h1) || (row > h2))
				aval = 80;
			else
				if ((col < l1) || (col > l2))
					aval = 80;

			int c = gdImageGetPixel(image, col, row);
			int r = gdImageRed(image, c);
			int g = gdImageGreen(image, c);
			int b = gdImageBlue(image, c);
			int c1 = gdImageColorAllocateAlpha(image, r, g, b,
aval);

			gdImageSetPixel(image, col, row, c1);
		}
	}

	gdImageSaveAlpha(image, 1);

	return image;
}

void storeImageBackground(gdImagePtr image, const char
*fileExtension,
			const char *dir, const char *fileName, const
char *groupName)
{
	char directoryFinal[64];
	char redisIndex[4];
```

```
	long long index;

	strcpy(directoryFinal, "");
	strcat(directoryFinal, dir);
	strcat(directoryFinal, "/");
	strcat(directoryFinal, dir);

	// Get current image count in database
	redisReply *getReply = redisCommand(Context, "SCARD %s",
groupName);

	index = getReply->integer;

	// Get the next
	index++;

	snprintf(redisIndex, 4, "%lld", index);

	freeReplyObject(getReply);

	// Regardless store the filename
	redisReply *setReply = redisCommand(Context, "SADD %s %s",
groupName, fileName);
	freeReplyObject(setReply);

	strcat(directoryFinal, redisIndex);
	strcat(directoryFinal, fileExtension);

	// Scale the image
	double imageHeight =
BACKGROUND_WIDTH*((double)image->sx/(double)image->sy);
	int imageHeightRounded = (int)imageHeight;
	image = gdImageScale(image, BACKGROUND_WIDTH,
imageHeightRounded);

	FILE *output = fopen(directoryFinal, "w");
	gdImageJpeg(image, output, 100);
	fclose(output);
}

void storeImageTarget(gdImagePtr image, const char *fileExtension,
                      const char *dir, const char *fileName, const
char *groupName)
{
	char directoryFinal[64];
	char fileNameFinal[64];
	char redisIndex[4];
	long long index;

	strcpy(directoryFinal, "");
	strcat(directoryFinal, dir);
	strcat(directoryFinal, "/");
```

```
	strcat(directoryFinal, dir);

	// Get current image count in database
	redisReply *setIncrement = redisCommand(Context, "INCR %s",
GRP_TARGET_COUNTER);

	index = setIncrement->integer;

	snprintf(redisIndex, 4, "%lld", index);

	freeReplyObject(setIncrement);

	strcpy(fileNameFinal, "");
	strcat(fileNameFinal, dir);
	strcat(fileNameFinal, redisIndex);

	// Regardless store the filename
	redisReply *setReply = redisCommand(Context, "SADD %s %s",
groupName, fileNameFinal);
	freeReplyObject(setReply);

	strcat(directoryFinal, redisIndex);
	strcat(directoryFinal, fileExtension);

	// Scale the image
	double imageHeight =
TARGET_WIDTH*((double)image->sx/(double)image->sy);
	int imageHeightRounded = (int)imageHeight;
	image = gdImageScale(image, TARGET_WIDTH,
imageHeightRounded);

	// Add transparency
	image = blendTransparency(image, image->sy);

	FILE *output = fopen(directoryFinal, "w");
	gdImagePng(image, output);
	fclose(output);
}

unsigned char* convertToBase64(gdImagePtr im) {
	// Convert the image to 1peg, default quality, getting
	// back pointer to allocated bytes and length
	int size;
	void* jpegbytes = gdImageJpegPtr(im, &size, 75);
	size_t encodedsize = 0;

	unsigned char* source = (unsigned char*) jpegbytes;
	unsigned char* b64codes = NULL;

	// Invoke base64 encoder first time just to get length of
	// base 64 string
	base64_encode(b64codes, &encodedsize, source, size);
```

```
	// Allocate space
	b64codes = malloc(encodedsize + 1);

	// Convert
	int res = base64_encode(b64codes, &encodedsize, source,
size);
	gdFree(jpegbytes);

	if (res != 0) {
		printf("Failed to base 64 encode data\n");
		if (b64codes != NULL)
			free(b64codes);
		return NULL;
	}

	return b64codes;
}

gdImagePtr loadImage(char* fileName, char* fileType)
{
	// Read in image
	FILE *file = NULL;
	file = fopen(fileName, "r");

	// Check for error
	if (file == NULL) {
		printf("\n");
		perror("fopen");
		return NULL;
	}

	gdImagePtr image = NULL;

	// Process appropriate image
	if (strcmp(fileType, ".gif") == 0) {
		image = gdImageCreateFromGif(file);
	} else if (0 == strcmp(fileType, ".png")) {
		image = gdImageCreateFromPng(file);
	} else if ((strcmp(fileType, ".jpg") == 0) ||
			(strcmp(fileType, ".jpeg") == 0)) {
		image = gdImageCreateFromJpeg(file);
	} else {
		printf("\nCannot handle image type %s\n", fileType);
	}

	fclose(file);

	return image;
}

void addBackground()
```

```
{
	// Setup
	static char absoluteAddress[64];
	static char *extension;
	static char *fileName;

	// Clear array, avaoid issues
	int i;
	for (i = 0; i < 64; i++)
		absoluteAddress[i] = '\0';

	printf("Enter full pathname of background image:\n");
	printf("Path: ");

	// Get Filename
	fflush(stdin);
	fgets(absoluteAddress, sizeof(absoluteAddress), stdin);
	absoluteAddress[strcspn(absoluteAddress, "\n")] = '\0';

	fflush(stdin);
	fflush(stdout);
	printf("Reading File: '%s' ", absoluteAddress);

	// Copy the string
	char absoluteAddressFileName[64];
	char absoluteAddressExtension[64];
	strcpy(absoluteAddressFileName, absoluteAddress);
	strcpy(absoluteAddressExtension, absoluteAddress);

	// Get the extension
	extension = strrchr(absoluteAddressExtension, '.');

	// Get the filename
	fileName = strrchr(absoluteAddressFileName, '/');
	fileName[strlen(fileName)-strlen(extension)] = '\0';
	for (i = 0; i < strlen(fileName); i++)
		fileName[i] = fileName[i+1];

	gdImagePtr background = loadImage(absoluteAddress,
extension);

	if (background != NULL) {

		// Store data in directory
		storeImageBackground(background, ".jpg", BackgroundDIR,
fileName, GRP_BACKGROUND_NAME);

		// Successful
		printf("\nAdded to collection of background images\n");

	} else {
		printf("Cannot Find Image.\n");
```

```
	}

	free(background);
}

void addTarget()
{
	// Setup
	static char absoluteAddress[64];
	static char targetTag[64];
	static char *extension;
	static char *fileName;

	// Clear array, avaoid issues
	int i;
	for (i = 0; i < 64; i++)
		absoluteAddress[i] = '\0';

	printf("Enter full pathname of background image:\n");
	printf("Path: ");

	// Get Filename
	fflush(stdin);
	fgets(absoluteAddress, sizeof(absoluteAddress), stdin);
	absoluteAddress[strcspn(absoluteAddress, "\n")] = '\0';

	fflush(stdin);
	fflush(stdout);
	printf("Reading File: '%s' ", absoluteAddress);

	// Copy the string
	char absoluteAddressFileName[64];
	char absoluteAddressExtension[64];
	strcpy(absoluteAddressFileName, absoluteAddress);
	strcpy(absoluteAddressExtension, absoluteAddress);

	// Get the extension
	extension = strrchr(absoluteAddressExtension, '.');

	// Get the filename
	fileName = strrchr(absoluteAddressFileName, '/');
	fileName[strlen(fileName)-strlen(extension)] = '\0';
	for (i = 0; i < strlen(fileName); i++)
		fileName[i] = fileName[i+1];

	gdImagePtr target = loadImage(absoluteAddress, extension);

	if (target != NULL) {

		// Get Tag
		printf("\nEnter a tag for this target: ");
		fflush(stdin);
```

```
            fgets(targetTag, sizeof(targetTag), stdin);
            targetTag[strcspn(targetTag, "\n")] = '\0';

            // Store data in directory
            storeImageTarget(target, ".png", TargetDIR, fileName,
targetTag);

            // Store Tag
            redisReply *setReply = redisCommand(Context, "SADD %s
%s", GRP_TARGET_NAME, targetTag);
            freeReplyObject(setReply);

            // Successful
            printf("Added to collection of background images\n");

      } else {
            printf("Cannot Find Image.\n");
      }

      free(target);

}

void generatePuzzle()
{
      printf("Generating Puzzle:\n");

      size_t keyCount = 0;
      size_t targetCount = 0;
      int i = 0;

      // Get current image count in database
      redisReply *getReply = redisCommand(Context, "SCARD %s",
GRP_TARGET_NAME);
      keyCount = getReply->integer;
      freeReplyObject(getReply);

      // Check if there is right amount of images
      if (keyCount >= MIN_IMAGE_TYPES) {

            /*
             * Get background
             */
            // Get background size
            redisReply *backgroundSize = redisCommand(Context,
"SCARD %s", GRP_BACKGROUND_NAME);

            char backgroundNumber[4];
            char backgroundFileName[64];
            int randomBackgroundNumber = 1;

            targetCount = backgroundSize->integer;
```

```
            if (randomBackgroundNumber != (int)targetCount) {
                  randomBackgroundNumber = generateRandomNumber(1,
(int)backgroundSize->integer);
            }

            snprintf(backgroundNumber, 4, "%d",
randomBackgroundNumber);
            strcpy(backgroundFileName, "");
            strcat(backgroundFileName, BackgroundDIR);
            strcat(backgroundFileName, "/");
            strcat(backgroundFileName, BackgroundDIR);
            strcat(backgroundFileName, backgroundNumber);
            strcat(backgroundFileName, ".jpg");

            printf("\nPicked %s%s as background\n", BackgroundDIR,
backgroundNumber);

            freeReplyObject(backgroundSize);

            // Get a background image
            gdImagePtr background = loadImage(backgroundFileName,
".jpg");

            if (background == NULL) {
                  // Exit out
                  printf("Error opening background image\n");
                  return;
            }

            int randomArray[MIN_IMAGE_TYPES] = {0, 0, 0, 0, 0};
            generateRandomTargetGroups(1, MIN_IMAGE_TYPES,
randomArray);

            redisReply *targetList = redisCommand(Context,
"SMEMBERS %s", GRP_TARGET_NAME);

            printf("\nPicked targets Below:\n");

            for (i = 0; i < MIN_IMAGE_TYPES; i++) {

                  // Check if there is min amount of images
                  redisReply *minImages = redisCommand(Context,
"SCARD %s", targetList->element[i]->str);

                  if (minImages->integer >= MIN_EACH_TYPE) {

                        redisReply *currentTarget =
redisCommand(Context, "SMEMBERS %s", targetList->element[i]->str);

                        int randomTargetArray[MIN_EACH_TYPE] = {0,
0, 0};
```

```
                    generateRandomTargetGroups(1, MIN_EACH_TYPE,
randomTargetArray);

                    printf("\nFor %s we are using:\n",
targetList->element[i]->str);

                    int j = 0;

                    for (j = 0; j < MIN_EACH_TYPE; j++) {

                        /*
                         * Get Each Target
                         */
                        char targetFileName[64];

                        strcpy(targetFileName, "");
                        strcat(targetFileName, TargetDIR);
                        strcat(targetFileName, "/");
                        strcat(targetFileName,
currentTarget->element[j]->str);
                        strcat(targetFileName, ".png");

                        printf("\t%s\n",
currentTarget->element[j]->str);

                        // Get a background image
                        gdImagePtr targetImage =
loadImage(targetFileName, ".png");

                        int top = generateRandomNumber(100,
(background->sx - 100));
                        int bottom = top + targetImage->sy;
                        int left = generateRandomNumber(100,
(background->sy - 100));
                        int right = left + targetImage->sx;

                        // Write out info to file
                        FILE *coords = fopen("puzzle.txt",
"a");
                        fprintf(coords, "%d, %d, %d, %d\n",
top, bottom, left, right);
                        fclose(coords);

                        gdImageCopy(background, targetImage,
                                    top,
                                    left,
                                    0, 0, 100,
targetImage->sy);

                        gdImageDestroy(targetImage);
                    }
```

```
                    // Free
                    freeReplyObject(currentTarget);


                } else {
                    printf("Add %lld more target groups to
%s\n", (MIN_EACH_TYPE - minImages->integer),
                            targetList->element[0]->str);
                    break;
                }

                // Get random image category for user to chose
                redisReply *targetListCount =
redisCommand(Context, "SCARD %s", GRP_TARGET_NAME);
                int randomTargetGroup = generateRandomNumber(1,
(int)targetListCount->integer);
                freeReplyObject(targetListCount);

                redisReply *currentTarget = redisCommand(Context,
"SMEMBERS %s", targetList->element[randomTargetGroup-1]->str);
                char *fileName = currentTarget->element[0]->str;

                char fileNameFinal[64];
                strcpy(fileNameFinal, "");
                strcat(fileNameFinal, TargetDIR);
                strcat(fileNameFinal, "/");
                strcat(fileNameFinal, fileName);
                strcat(fileNameFinal, ".png");

                // Get image of random target
                gdImagePtr lookingFor = loadImage(fileNameFinal,
".png");

                // Make HTML page
                generateHTML(background, lookingFor);

                freeReplyObject(currentTarget);
                freeReplyObject(minImages);

            }

            gdImageDestroy(background);
            freeReplyObject(targetList);

        } else {
            printf("Add %lu more target groups\n", (MIN_IMAGE_TYPES
- keyCount));
        }

}

void generateHTML(gdImagePtr imageMain, gdImagePtr lookingFor)
```

```
{
	unsigned char *imageMain64 = convertToBase64(imageMain);
	unsigned char *lookingFor64 = convertToBase64(lookingFor);

	FILE *output = fopen("index.html", "w");

	fprintf(output, "<html><head></head><body>");
	fprintf(output, "<h1>Prove You Are Not A Bot</h1>");
	fprintf(output, "<p>Look at this image</p>");
	fprintf(output, "<img width='100px'
src='");
	fprintf(output, "%s", lookingFor64);
	fprintf(output, "' />");
	fprintf(output, "<p>The large picture below has three similar
images embedded in it.</p>");
	fprintf(output, "<p>Click on all three embedded images</p>");
	fprintf(output, "<img height='800px'
src='");
	fprintf(output, "%s", imageMain64);
	fprintf(output, "' />");
	fprintf(output, "</body></html>");

	fclose(output);
}

void closeConnection()
{
	redisFree(Context);
	printf("Disconnected from Redis Server\n");
}

void setupConnection() {
	struct timeval timeout = {1, 500000};

	Context = redisConnectWithTimeout(HostName, Port, timeout);

	if ((Context == NULL) || (Context->err)) {
		if (Context) {
			printf("Connection error: %s\n",Context->errstr);
			redisFree(Context);
		} else {
			printf("Connection error: can't allocate redis
context\n");
			fflush(stdout);
		}

		exit(1);
	}

	// All is fine
	printf("Connected to Redis Server\n");
}
```

```
void menuSelect()
{
	setvbuf(stdout, NULL, _IONBF, 0);

	while (1) {
		printf("> ");

		char command[32];
		fgets(command, 32*sizeof(char), stdin);
		command[strcspn(command, "\n")] = 0;

		if (command[0] == 'q') {
			printf("Goodbye.\n");
			break;
		} else if (command[0] == '?') {
			printf("Commands:\n");
			printf("\t?: Print this command list\n");
			printf("\tq: Quit\n");
			printf("\tBackground: Add another background
picture\n");
			printf("\tTarget: Add another target image\n");
			printf("\tGenerate: Create a puzzle\n");
		} else if (strcmp(command, "Background") == 0) {
			addBackground();
		} else if (strcmp(command, "Target") == 0) {
			addTarget();
		} else if (strcmp(command, "Generate") == 0) {
			generatePuzzle();
		}

	}
}
```

MakeFile for linking homemade objects from the above source with GD and Redis libraries.

makefile

```
CC=gcc
CFLAGS=-Wall -g

OBJFILES=main.o base64.o

LIBRARYFILES=-lgdbm -lgd -lm

Main: $(OBJFILES)
$(CC) -o Main $(OBJFILES) $(LIBRARYFILES)

Clean:
rm -f Main *.o *.html *~

main.o:base64.h
base64.o:base64.h
```

2.3 Results

Below is the output (in bold) and the input (in normal) from the program

Adding Backgrounds

Connected to Redis Server
> Background
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/back1.jpg
Reading File: '/Users/james/Desktop/Images/back1.jpg'
Added to collection of background images
> Background
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/back2.jpg
Reading File: '/Users/james/Desktop/Images/back2.jpg'
Added to collection of background images

Adding Targets

> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/brendan1.jpg
Reading File: '/Users/james/Desktop/Images/brendan1.jpg'
Enter a tag for this target: Brendan
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/brendan2.jpg
Reading File: '/Users/james/Desktop/Images/brendan2.jpg'
Enter a tag for this target: Brendan
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/brendan3.jpg
Reading File: '/Users/james/Desktop/Images/brendan3.jpg'
Enter a tag for this target: Brendan
Added to collection of background images

> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/cat1.jpg
Reading File: '/Users/james/Desktop/Images/cat1.jpg'
Enter a tag for this target: Cat
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/cat2.jpg
Reading File: '/Users/james/Desktop/Images/cat2.jpg'
Enter a tag for this target: Cat
Added to collection of background images

> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/cat3.jpg
Reading File: '/Users/james/Desktop/Images/cat3.jpg'
Enter a tag for this target: Cat
Added to collection of background images

> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/doge1.jpg
Reading File: '/Users/james/Desktop/Images/doge1.jpg'
Enter a tag for this target: Doge
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/doge2.jpg
Reading File: '/Users/james/Desktop/Images/doge2.jpg'
Enter a tag for this target: Doge
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/doge3.jpg
Reading File: '/Users/james/Desktop/Images/doge3.jpg'
Enter a tag for this target: Doge
Added to collection of background images

> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/pepe1.jpg
Reading File: '/Users/james/Desktop/Images/pepe1.jpg'
Enter a tag for this target: Pepe
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/pepe2.png
Reading File: '/Users/james/Desktop/Images/pepe2.png'
Enter a tag for this target: Pepe
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/pepe3.png
Reading File: '/Users/james/Desktop/Images/pepe3.png'
Enter a tag for this target: Pepe
Added to collection of background images

> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/pusheen1.png
Reading File: '/Users/james/Desktop/Images/pusheen1.png'
Enter a tag for this target: Pusheen
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/pusheen2.jpg
Reading File: '/Users/james/Desktop/Images/pusheen2.jpg'

Enter a tag for this target: Pusheen
Added to collection of background images
> Target
Enter full pathname of background image:
Path: /Users/james/Desktop/Images/pusheen3.gif
Reading File: '/Users/james/Desktop/Images/pusheen3.gif'
Enter a tag for this target: Pusheen
Added to collection of background images

Generating Captcha

> Generate
Generating Puzzle:

Picked background1 as background

Picked targets Below:

For Brendan we are using:
target3
target2
target1

For Doge we are using:
target8
target9
target7

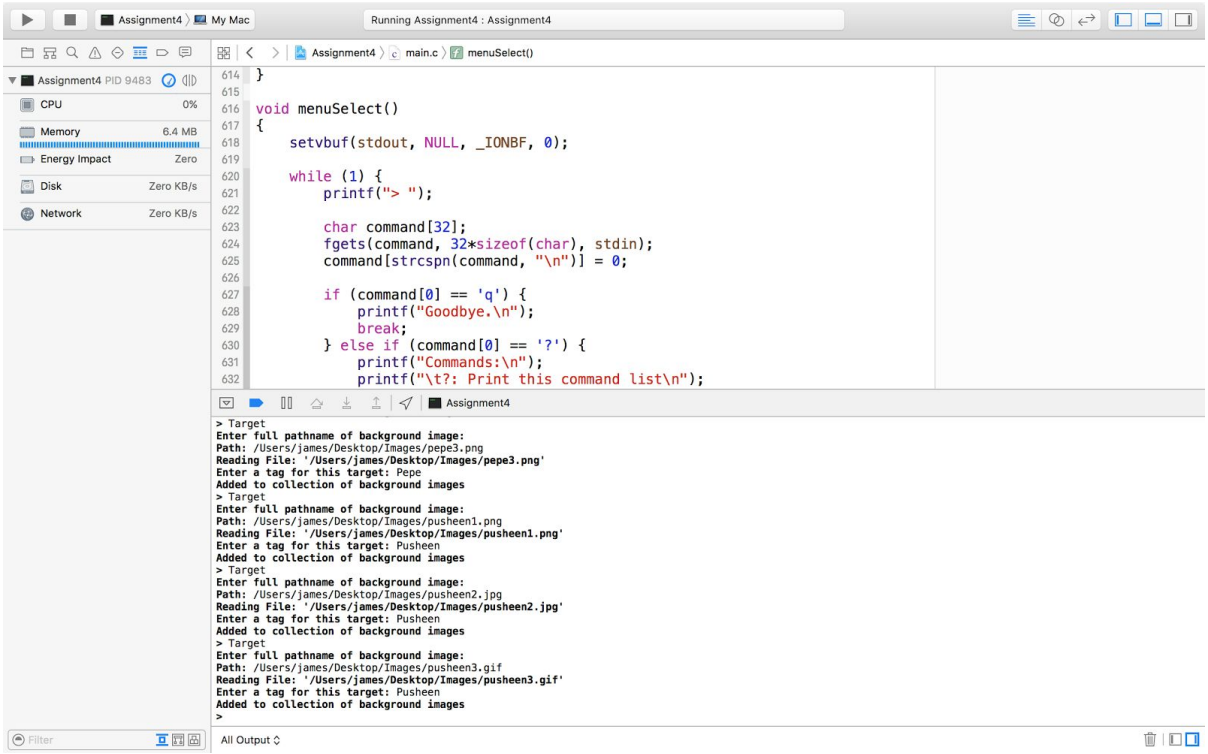
For Pusheen we are using:
target13
target14
target15

For Cat we are using:
target6
target4
target5

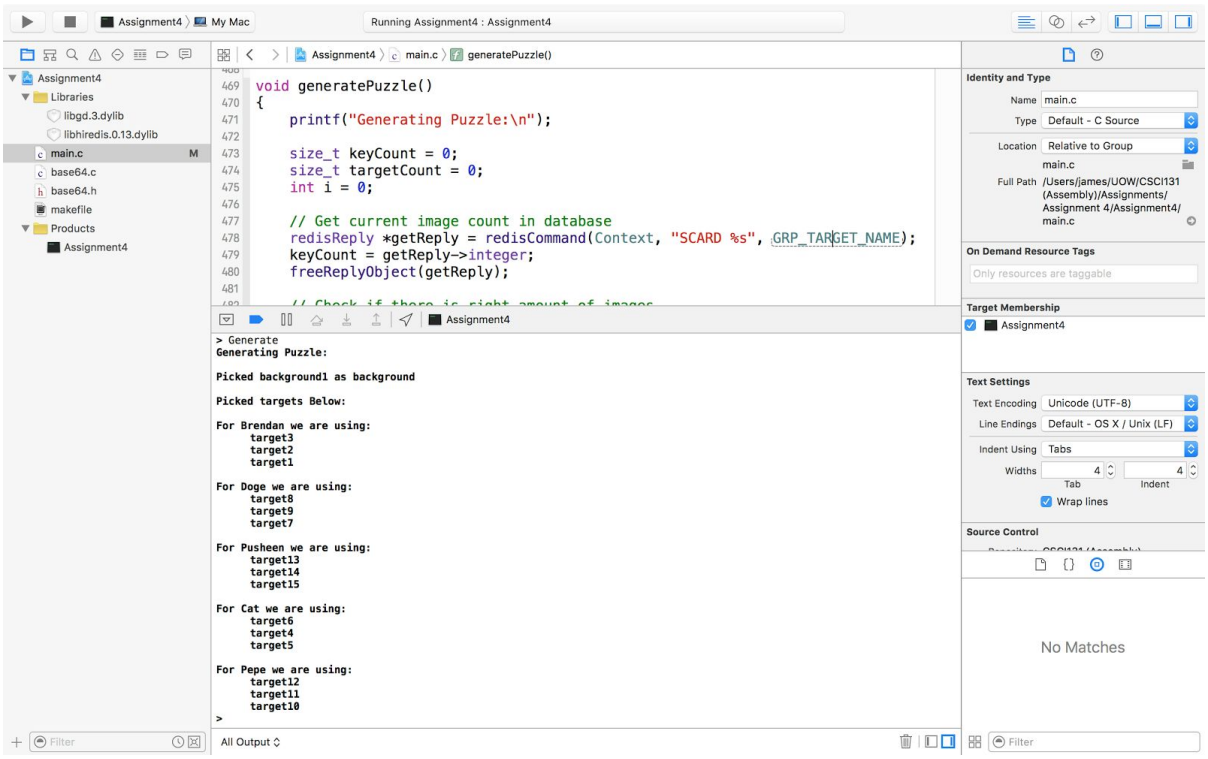
For Pepe we are using:
target12
target11
target10

Screenshots as Evidence

Program Execution
Adding Targets

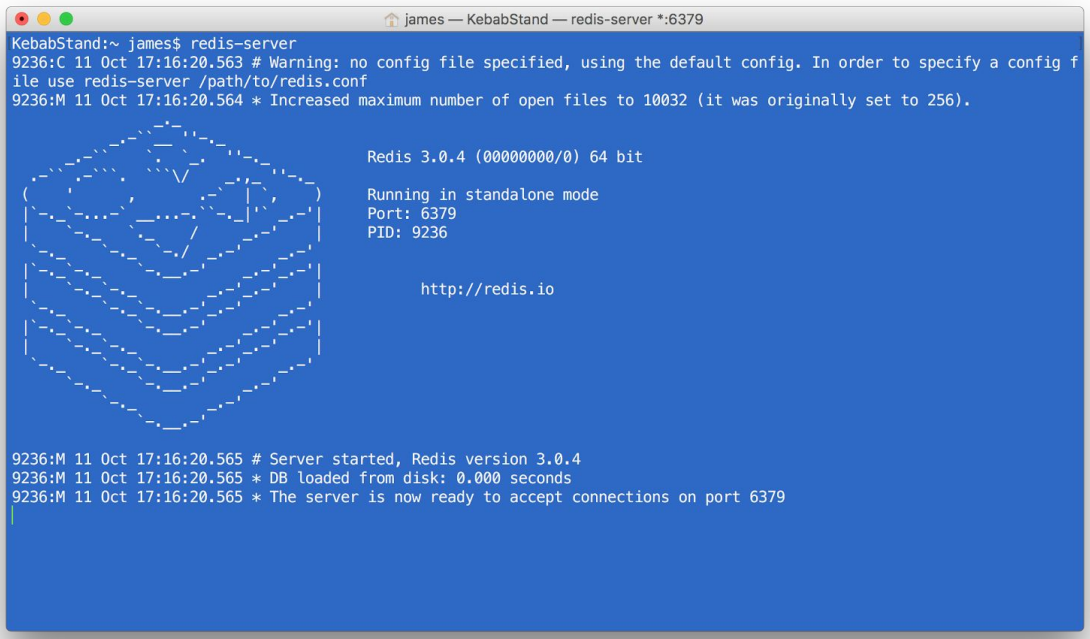


Generating Puzzle

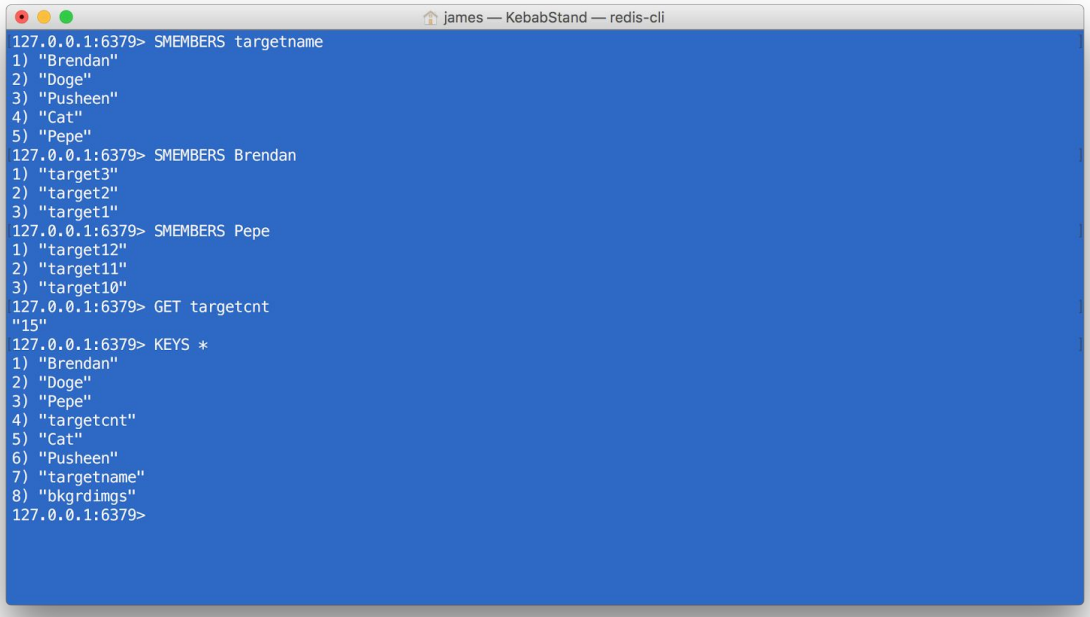


Redis Database

Operation - Starting Server

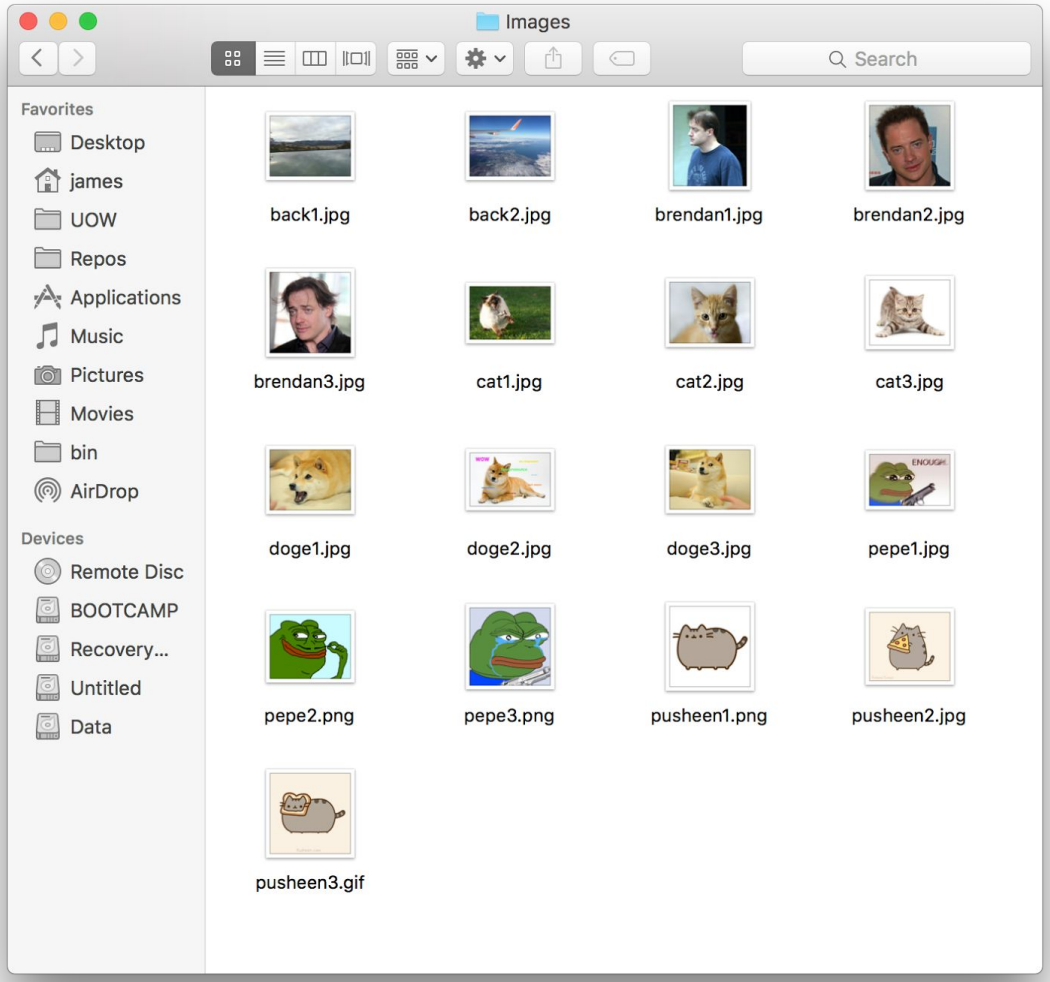


Use - Names and Counters for images

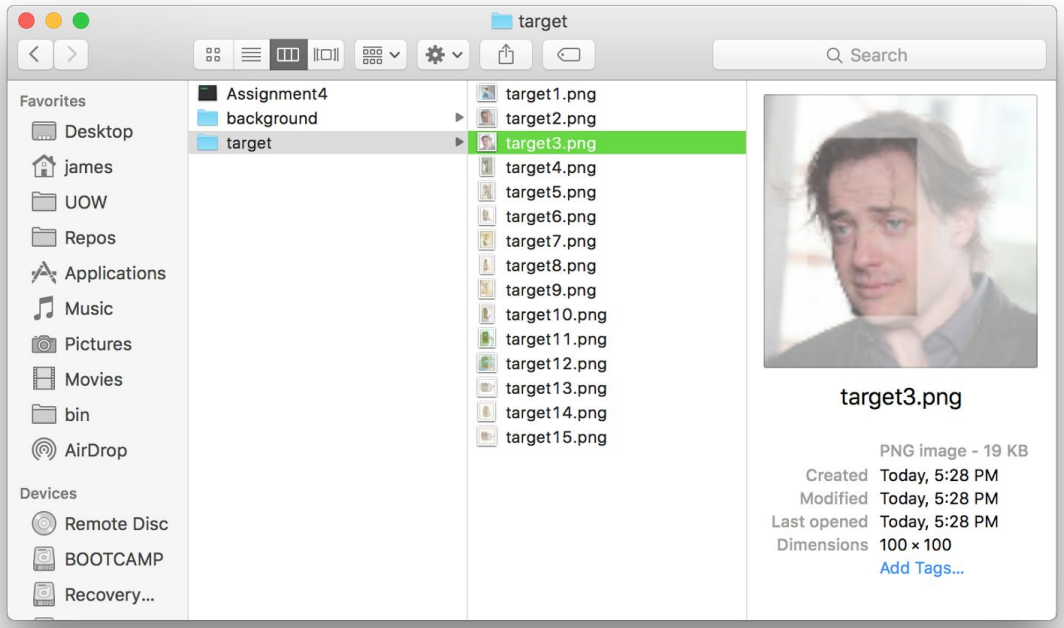


Directories

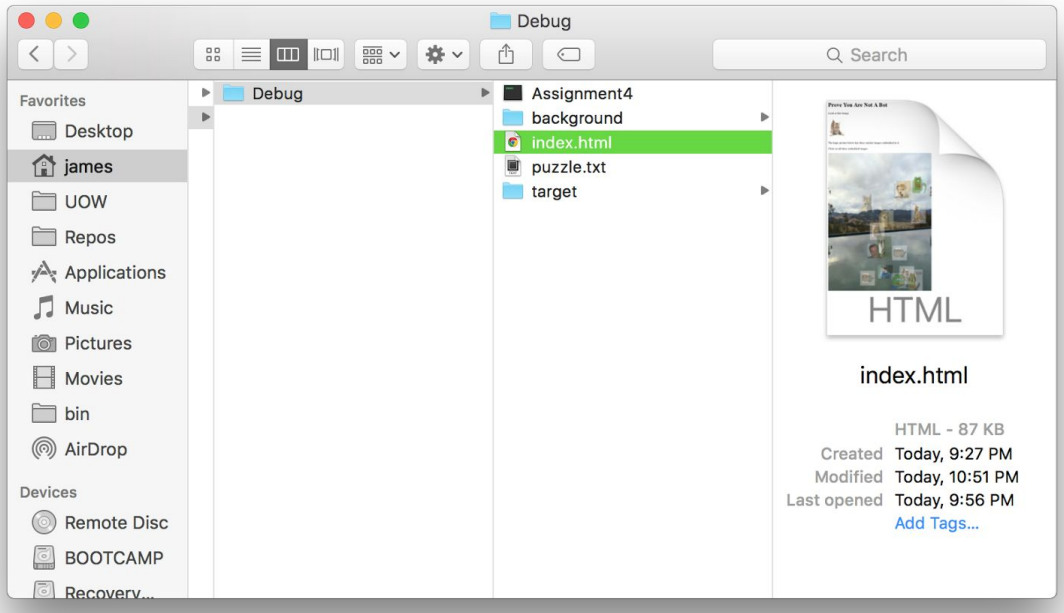
Original Images Folder



Executable Folder
Before Captcha Generation

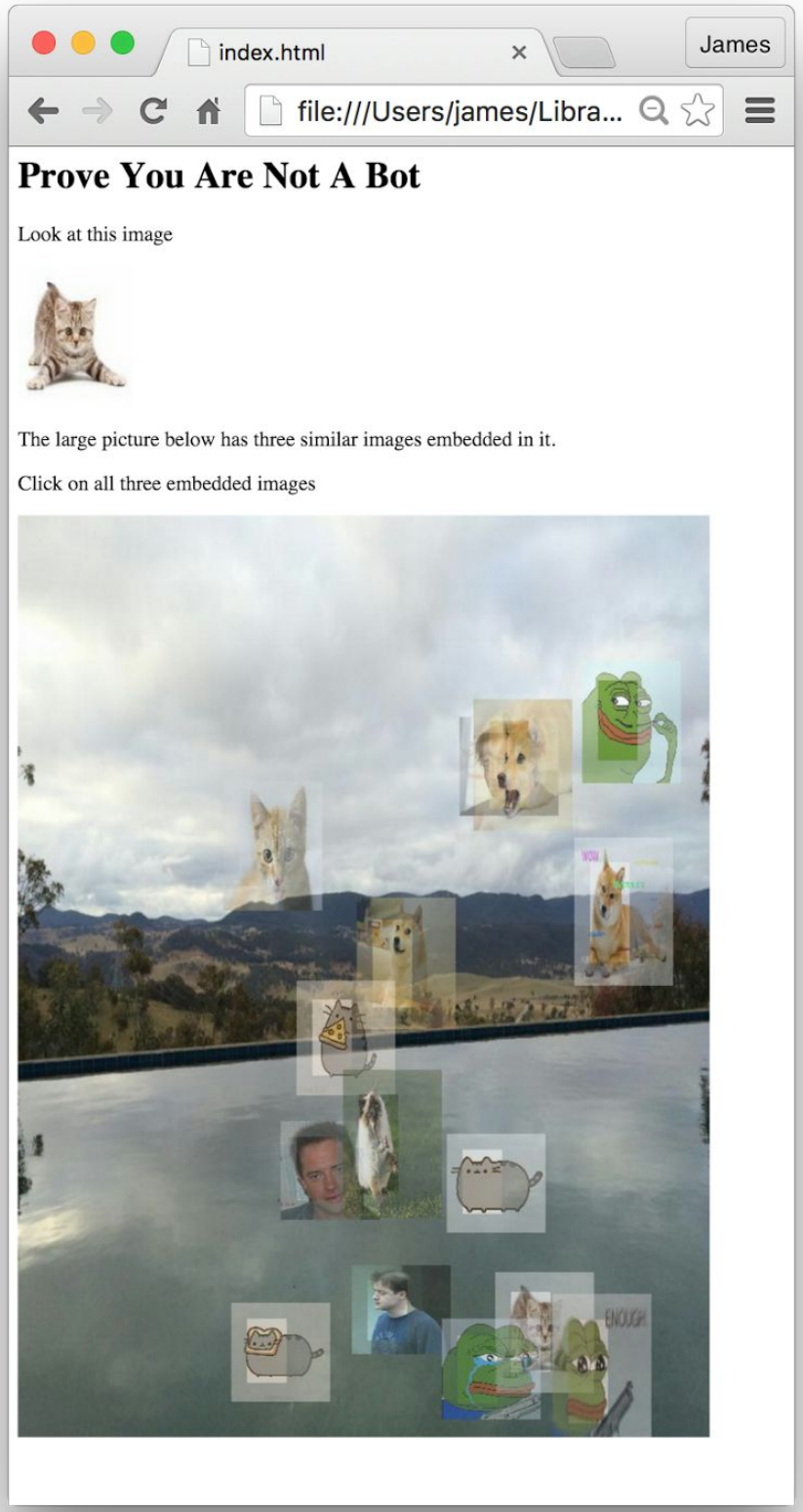


After Captcha Generation

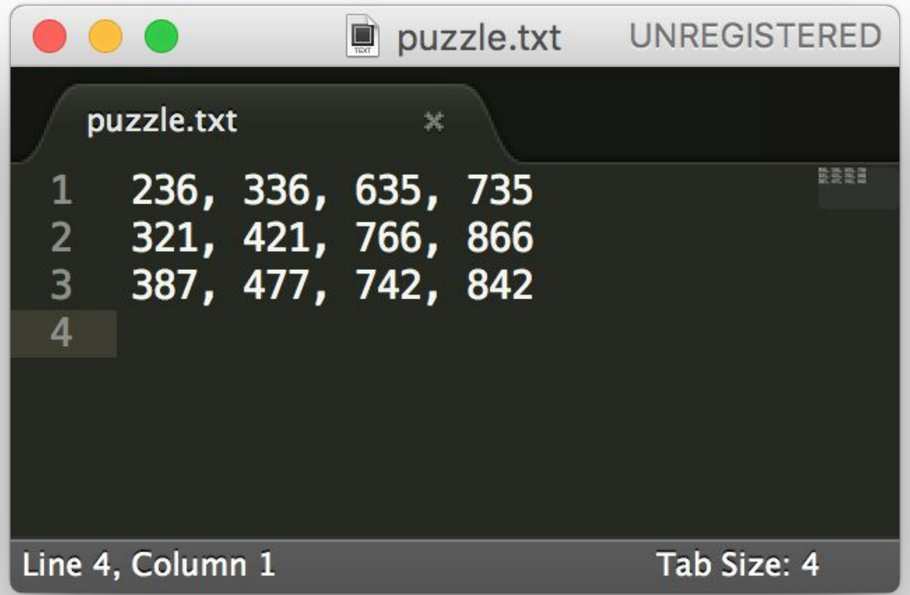


Output Captcha Trail 1

HTML Output File
→ Output of Generate function



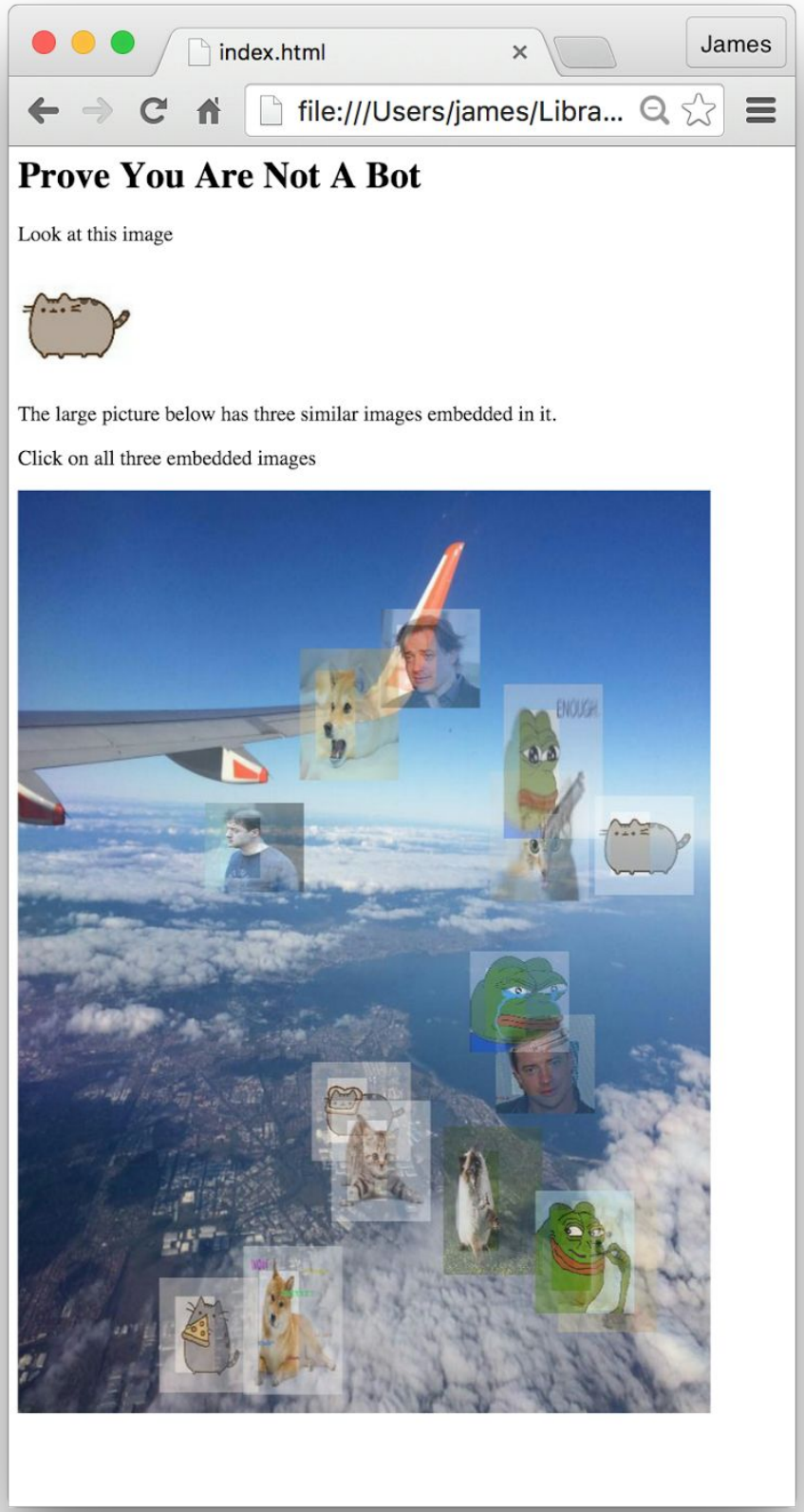
Co-ordinates File
→ Coordinates of 3 correct images for verification (in this case the *Cat* Target)



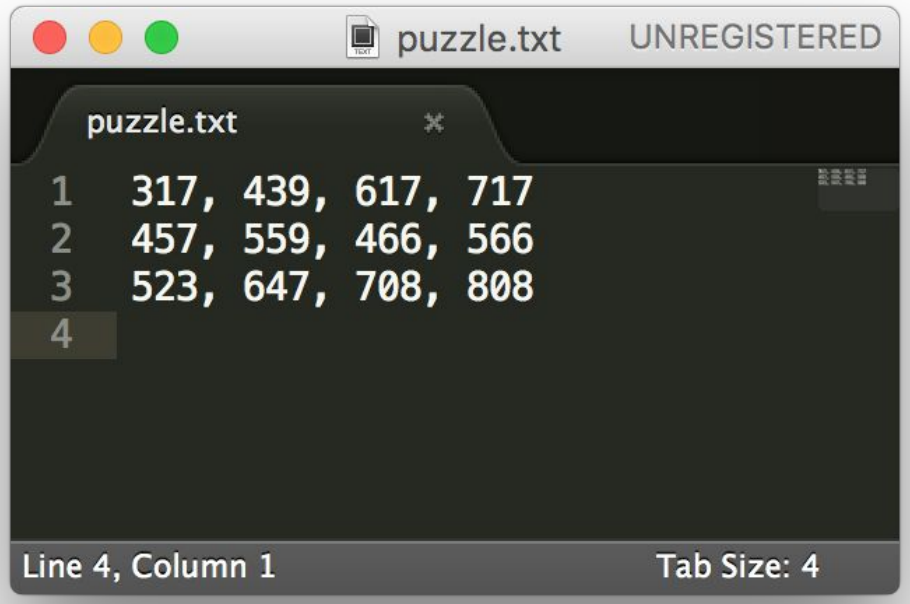
Source File
→ Source behind HTML file to show Base64 usage

Output Captcha Trail 2

HTML Output File
→ Output of Generate function



Co-ordinates File
→ Coordinates of 3 correct images for verification (in this case the *Pusheen* target)



3. Conclusion

As we can see the program runs as expected and provides the appropriate output in creating random captcha images utilising the redis DataBase and GD Libraries.