Second Proposal

First serious, second fun. Titles are as appropriate for the Nexmo Developer blog.

Tutorial Title

Learn the Nexmo SMS APIs by Playing the 1986 Swiss Text Adventure Game

Tutorial Abstract

> *Let's have some fun. We're going to re-create a text Adventure game from 1986. It was designed to secretly run on a mainframe computer's operator console. We'll use our smart phone's text messages to simulate that operator console as we try to survive our "Swiss Adventure". We'll be learning the "ins and outs" of the Nexmo SMS API by sending and receiving text messages from our PHP application and checking message-delivery receipts. We'll learn the 2-Factor Authentication workflow using the Nexmo Verify API.*
>
> *As developers it can be challenging to integrate with any third-party API. We need to understand the sequencing of events in addition to the individual requests and responses. On the client side that's easy. Modern browsers, and tools like the Charles Web Debugging Proxy, can provide a complete picture of client-side traffic including headers and cookies. It can be more difficult to observe server-side interactions. We'll use a web page to display the API interactions (requests and responses) in near-real time as we conduct our Swiss Adventure via Nexmo's SMS API.*

About the App

The original "Swiss Adventure" (1986) was written for the Cray mainframe operator console. This port uses a PHP application as the game engine and the user's smart phone text messaging app as the operator console. Commands and responses are by text message. The accompanying web page displays the Nexmo API requests and responses in real time using JQuery AJAX requests.

We demonstrate two-factor authentication by entering the phone number in a web page and following the Nexmo Verify API workflow. We mask (redact) the verification token when displaying the API requests/responses. The verified phone number will then be used as the adventure game's operator console.

Target Audience

This tutorial is for intermediate and advanced back end developers (in PHP) who wish to learn to implement Nexmo's Verify and/or SMS APIs. The application being developed is a teaching tool rather than having any particular business use.

An interest in retro gaming is helpful but not required.

Prerequisites

Readers should have

- Intermediate knowledge of PHP
- Basic knowledge of MySQL and MySQL table design
- Basic knowledge of using Javascript with JQuery for AJAX requests

Familiarity with the CakePHP framework is helpful but not required.

Learning Objectives

Readers of the tutorial can learn:

- How to use the Nexmo Verify API for two-factor authentication
- How to send a text message using the Nexmo SMS API
- How to receive a text message using the Nexmo SMS API
- How to receive a delivery receipt using the Nexmo SMS API
- The sequencing and request/response content of the above API usage
- How to use web sockets for inbound information from Nexmo

Language and Framework

PHP 7.2 and CakePHP 3 using MySQL 5.6. Javascript and JQuery (versions to be determined) for displaying the API requests/responses in near-real time.

Specific Nexmo/TokBox APIs

- Nexmo SMS (outbound SMS; delivery receipt; inbound SMS)
- Nexmo Verify

Related Reading

- cURL, HTTPS & the Nexmo SMS API Behind the Scenes
- Verify API Overview
- SMS API Overview
- Connect your local development server to the Nexmo API using an ngrok tunnel
- Two-way SMS for customer engagement
- How to Show SMS Notifications in the Browser with Angular, Node.JS, and Ably
- Create Your Own Adventure - provided by the Nexmo community
- Swiss Adventure - the original text adventure, written in Cray I/O Subsystem Assembly Language

External Submissions

- PHP Architect Magazine - for any PHP-centric projects that teach concepts
- freeCodeCamp - any developer-centric articles/tutorials
- Dev.to - any developer-centric articles/tutorials
- codementor - any developer-centric tutorials
- scotch.io - any web development tutorials, thus plausibly anything PHP-centric
- Digital Ocean Community - any developer-centric tutorials
- LinkedIn - anything purporting to be informative
- CakePHP Blog - Anything built as a demo project with the CakePHP framework could be a topic for the FriendsOfCake group or the CakePHP Bakery blog. I can check with the core development team when the time comes.

Three Tweets

The following tweets should include an image; perhaps even just a screen shot of the code.

1. Retro Computing: Learn the @Nexmo SMS API by playing the 1986 text Adventure game that takes you around Switzerland!
2. “You have found the highest rail station in Europe. The Jungfrau, Eiger, and Monch display their deadly beauty.” - learn how @Nexmo APIs work while you play the 1986 Adventure game using text messages to simulate your mainframe operator console.
3. “As you climb through the trap door…” - Let’s learn about text messages the fun way! This tutorial uses a 1986 adventure game to teach how to use the @Nexmo 2-factor verification and SMS APIs. Your phone becomes the operator console.

The following long-form Twitter thread begins with the back story. Persons with a retro gaming interest will pick up on the importance of the back story.

1. Once upon a time, I had an Adventure. That’s me on the left, 9 years earlier, out for a different Adventure. THREAD

West Side

1. In fact I wrote the Adventure… as a game. All you gamers out there will know that the back story is important. That's me seated, far left, on that prior adventure. So what's the story?

Seated

1. Retro gaming means we need to step back in time… in our case, to 1986. The original Nintendo Entertainment System was just being introduced. The Legend of Zelda would appear the following year.
   https://en.wikipedia.org/wiki/History_of_Nintendo#Game_&_Watch_and_Nintendo_Entertainment_System
2. I'd never heard of Nintendo… but I'd played and solved Zork, and the original Colossal Cave Adventure. These text-based games defined the "text Adventure" genre of the era.
   https://en.wikipedia.org/wiki/Zork
3. For me, in 1986, "computing" meant Supercomputing. The Cray Research mainframes were the world's fastest computers. These Cray mainframes become our gaming story.

Cray Mainframe

1. People came to Cray Research to learn how to program their own supercomputer. With so few in existence there were no other options. The months-long training sequence optionally included the operating system internals (yuck).

```
          EJECT                                                                 A0255.933
***********************************************************************       A0255.934
**                                                                            A0255.935
*                                                                             A0255.936
*NAME     MEMAL                                                              A0255.937
*                                                                             A0255.938
*PURPOSE MEMAL WILL ALLOCATE A VARIABLE SIZE AREA OF MEMORY FROM A             A0255.939
*         MEMORY POOL.                                                        A0255.940
*                                                                             A0255.941
*ENTRY    A6 = NUMBER OF THE MEMORY POOL TO ALLOCATE FROM                     A0255.942
*         A7 = NUMBER OF WORDS REQUIRED                                       A0255.943
*                                                                             A0255.944
*EXIT     A6 = STATUS                                                        A0255.945
*                   0 = GOOD RETURN                                          A0255.946
*                   1 = INVALID POOL  NUMBER                                 A0255.947
*                   2 = INVALID NUMBER OF WORDS REQUESTED                    A0255.948
*                   3 = MEMORY NOT AVAILABE                                  A0255.949
*         A7 = ADDRESS OF FIRST USEABLE WORD ALLOCATED--MEANINGFUL            A0255.950
*              ONLY IF A6 = 0                                                 A0255.951
*                                                                             A0255.952
***********************************************************************       A0255.953
MISTAKE1 =          1                                                         A0255.954
MISTAKE2 =          2                                                         A0255.955
MISTAKE3 =          3                                                         A0255.956
MEMAL    =          *                                                         A0255.957
         A5         POOLTBL          BASE ADDRESS OF POOL TABLE               A0255.958
         GET,S5     S6&S7,PTMAX,A5 MAX POOL NUMBER                            A0255.959
         A4         S5                                                        A0255.960
         A0         A4-A6                                                     A0255.961
         JAM        MEMER1           JUMP IF POOL NUMBER GREATER THAN MAX     A0255.962
         A5         A5+A6            ADDRESS OF POOL WORD                     A0255.963
         S0         0,A5                                                      A0255.964
         JSZ        MEMER1                                                    A0255.965
         A0         A7                                                        A0255.966
         JAZ        MEMER2           JUMP IF ZERO WORDS REQUESTED             A0255.967
*                                                                             A0255.968
*        THE TOTAL SIZE TO BE ALLOCATED IS  THE REQUESTED SIZE PLUS           A0255.969
*        THE SIZE OF THE HEADER AND TRAILER                                   A0255.970
*                                                                             A0255.971
         A4         MPHT                                                      A0255.972
         A7         A7+A4            TOTAL SIZE REQUIRED                      A0255.973
*                                                                             A0255.974
*        VERIFY THAT THE REQUESTED SIZE IS NOT LARGER THAN THE POOL           A0255.975
*                                                                             A0255.976
         GET,S1     S6&S7,PTSIZE,A5                                           A0255.977
         A4         S1                                                        A0255.978
         A0         A4-A7                                                     A0255.979
         JAM        MEMER2                                                    A0255.980
*                                                                             A0255.981
         GET,S1     S6&S7,PTBASE,A5                                           A0255.982
         A4         S1               BASE ADDRESS OF MEMORY POOL              A0255.983
*                                                                             A0255.984
*        LOCK OUT INTERRUPTS UNTIL THE SPACE HAS BEEN ALLOCATED               A0255.985
*                                                                             A0255.986
         S1         1                                                         A0255.987
         STRLK,A0   S1                                                        A0255.988
MEMAL10  =          *                                                         A0255.989
         GET,S1     S6&S7,MPID,A4  VALIDATE THE HEADER WORD                   A0255.990
         S2         IDWRD,A6                                                  A0255.991
         S0         S1-S2                                                     A0255.992
         ERRSN                                                                A0255.993
         GET,S0     S6&S7,MPST,A4                                             A0255.994
         JSN        MEMAL50          JUMP IF AREA IN USE                      A0255.995
         GET,S1     S6&S7,MPSIZE,A4  SIZE OF AVAILABLE AREA                   A0255.996
         A1         S1                                                        A0255.997
         A0         A1-A7                                                     A0255.998
         JAN        MEMAL30          REQUEST SIZE UNEQUAL TO AVAILABLE        A0255.999
```

1. Yes, as late as the 1980s, the operating system courses were all in assembly language.
2. In fact there were TWO operating systems to learn. First we learned the Cray mainframe itself, and then we moved on to the mysterious I/O Subsystem, a rack of custom mini-supercomputers with only 128 KBytes of memory. But fast… oh, so fast.

CRAY X-MP/48

1. I was moving back to Cray Research headquarters to teach the operating system internals (yuck). I sat through the entire operating system sequence, since I would soon be teaching part of it.
2. I'd be teaching the mainframe part, so I didn't need to worry about the I/O Subsystem (IOS) part. Yay! But I still had to sit through those two weeks and learn the IOS stuff anyway.
3. The IOS class includes a class project. Each student needs to write an "overlay" that does something, much like "Hello World".

```
*ID TN+101BA,DC=AT
*D AT.4707
OVLTBL    EQUALS    O'720          .Maximum overlay number
*ID TN+101BK,DC=K
*B K.380
          ENTRY     OVLTABLE       .Allow SWISS access
*ID TN+101BC,DC=CALL
*B CALL.260
          OCOME      SWISS,FROM=IOP0+IOP1+IOP2+IOP3
*ID TN+101BO,DC=OVLNUM
*I OVLNUM.103
SWISS     OVNM       (Swiss adventure, root overlay)
SWSDAT    OVNM       (Swiss data)
SWCHECK   OVNM       (Display checkpoint sign)
SWSHOW    OVNM       (Display location text, get new location)
SWSLOC    OVNM       (Read a description entry into Local Memory)
SWSMOV    OVNM       (Display text showing travel)
SWSNEX    OVNM       (Get new valid location)
SWSNO     OVNM       (Display snide remark)
SWSWEL    OVNM       (Print welcome message and instructions)
*MOVEDK SWISS:STATS
*MOVEDK SWSDAT:SWISS
*MOVEDK SWSHOW:SWSDAT
*MOVEDK SWSLOC:SWSHOW
*MOVEDK SWSMOV:SWSLOC
*MOVEDK SWSNEX:SWSMOV
*MOVEDK SWSNO:SWSNEX
*MOVEDK SWSWEL:SWSNO
*MOVEDK SWCHECK:SWISS
```

Overlay

1. That sounds trivial, but it isn't. It's something like writing an extension to the Linux kernel.
2. The student overlay needs to get registered with the rest of the operating system software. (Yes this really is the Adventure back story…)

3. We need to boot the multi-million-dollar mainframe with the modified IOS software. When something doesn't work as expected, the instructor may or may not be able to help.
4. Rebuilding the software might mean waiting until the next day for another try… and we're already at the last couple of class days.
5. Most students completed the assigment as 20–30 lines of code: Display something on the operator console and exit.
6. I, on the other hand, asked around. Had anyone ever created a game program as their assignment?
7. Nope. Never been done. With limited class time and very limited test time, there's good reason that new students never made the attempt, in a new just-learned assembly language.
8. So I wrote an adventure game, inspired by the now-ancient Colossal Cave Adventure program. It's text-based interactive fiction. This was the mid–1980s, after all!
9. It's a tour of Switzerland, with a maze of twisty passages all nearly alike, and a final puzzle involving Arabic words. We had just moved back to Minnesota after living in Saudi Arabia, thus a challenge in Arabic seemed like the right thing to do.
0. The program did have a problem, which I considered a feature at the time.
1. You could not exit the program except by winning (completing) it. One had to solve the whole thing or reboot the Cray mainframe. Oops!
2. Those 1500 lines of I/O Subsystem assembler code are now, so far as I know, the largest surviving example of Cray I/O Subsystem source code. Computing museum curators are actually out there looking for Cray Operating System source code to no avail.
3. Funny thing… five years later I joined the I/O Subsystem software development team. In the software industry, when you put in the effort to achieve something weird, the effort may eventually pay off in an interesting way.
4. That's the back story from 1986. Very few people ever got to play Swiss Adventure, both because it was silly and because it was hidden within the operating system. It had to be played on the Cray supercomputer's physical operator console. Few had physical access.
5. What good does this do us today? I turned the back story into a teaching tool. For you, if you choose to accept the challenge!
6. Operator consoles of the era were entirely text-based. Type something in and the computer types something back. You know… just like text messages work today. Aha!
7. We're going to implement Swiss Adventure using your smart phone's text messaging capability as the operator console.
8. This means that you'll learn how to verify that you control the "operator console" phone number. That is, you'll implement 2-factor authentication in PHP using the @Nexmo Verify API.
9. This also means that you'll learn how to use @Nexmo's SMS API to send and receive text messages to and from your very own "operator console" and check delivery receipts.
0. Perhaps more importantly you'll be observing the SEQUENCING of the API requests and responses. You may or may not survive your Swiss Adventure but you'll sure understand how to use those APIs!
1. "You discover the Lion of Luzern…" our Retro Computing tutorial begins here. https://github.com/ewbarnard/NexmoInsight/blob/master/Proposals/TextAdventure.md
2. Every retro gamer knows about cheat mode… right? You are welcome to trace your way through your Swiss Adventure by studying the original source code. I've placed it online with an MIT Open Source license.

3. It's 1,523 lines of assembler. So the question is… how badly do you want to cheat? Ha! END https://github.com/ewbarnard/InsidePHP/blob/master/APML/swiss.txt

Call for Papers

This tutorial is suitable as both a regular talk (60 minutes, intermediate) and as a hands-on workshop (3 hours, intermediate, need ngrok and LAMP stack or equivalent).

Talk Titles

- Retro Computing: Simulate an Operator Console using Text Messages
- Learn the Nexmo SMS APIs by playing the 1986 Swiss Adventure Game

Talk Description

Let's have some fun. We're going to re-create a text Adventure game from 1986 - but it required an operator console. We'll send and receive text messages to simulate that console. We'll use Nexmo's SMS API for messaging and Verify API for 2-factor authentication.