Homework 1

for WEBLAMP 443 MySQL

by Paul Mitchum, paul@mile23.com

Introduction

This document is compiled from a number of documents that reside here: https://github.com/paul-m/VirtualPersistAPI/tree/master/design

A markdown version of this document is located here: https://github.com/paul-m/VirtualPersistAPI/blob/master/design/Homework1.md

VirtualPersistAPI Project Overview

What Is It?

VirtualPersistAPI (VPA) is a RESTful API for storing arbitrary data.

Users can POST, GET, and DELETE data, keyed on two other pieces of data: A category and a key.

In addition, users can query for the categories which belong to them, and keys which belong to their categories.

Within the API, users are always known by a UUID.

A detailed specification document is here: https://github.com/paul-m/VirtualPersistAPI/blob/master/spec/VirtualPersistAPI.md

What Needs To Be Done?

Since this is a continuation of last semester's project, some work is already done. Most of the data model is complete. The user model exists but only minimally. There is no authentication and only one HTML page, which gives an overall report on existing users and keys.

Forward progress will be broken up into milestones:

Milestone 1

Milestone 1 will deal with the user model and authentication. This implementation will be very simple, to help me learn Symfony's user model.

- Basic authentication: Send a password as `?pw=[password]` in API request.
- User login and report through HTML form.
- Unit and functional tests.

Milestone 2

Milestone 2 involves improving the authentication system.

- Research authentication systems.
- Design authentication system based on the limits imposed by the use case. Probably this: https://github.com/paul-m/VirtualPersistAPI/blob/master/spec/VPA_Security.md
- Unit and functional tests.

Milestone 3

Milestone 3 will be the permissions system.

- Basic designs for permissions system. This might be adding an admin field to the user entity, or a separate table connecting user entities to permission entities.
- Unit and functional tests.

Milestone 4

A back-end is needed for management of users and various reports and log analysis efforts.

- Reports and analysis.
- Unit and functional tests.

Incidental Milestones

It might be the case that one of the following gets done while working on the other milestones:

- Deployment script(s): Pagodabox.com, AWS, etc.

- Performance testing and tweaking, specifically for the database.
- Add more here.

VirtualPersistAPI High Level

VPA runs under a fairly standard LAMP or *AMP stack.

The target for deployment will be Pagodabox and/or OpenShift (chosen because they are free services).

Consumers use the API to query and manage their data on the system.

In the main use case, consumers will be scripts in LSL, on Second Life or OpenSim.

VirtualPersistAPI Project Requirements

VirtualPersistAPI is meant to be a package that can be deployed with relative ease by anyone. Therefore its requirements are typical for a PHP web application.

VirtualPersistAPI is based in the Symfony 2.1 PHP framework, and uses Doctrine DBAL/ORM for the database.

This means that VPA's dependencies are the same as those frameworks' dependencies.

- Symfony: http://symfony.com/doc/2.1/reference/requirements.html
- Doctrine: http://docs.doctrine-project.org/en/2.0.x/reference/introduction.html#requirements

In essence:

- PHP 5.3.3 or greater with JSON and ctype extensions.
- A database supported by PHP's PDO class.

These packages will run on a commodity web server, using LAMP stack, so:

- A Unix-flavored box, deployed in cloud hosting.
- Apache http server.
- MySQL.
- PHP.

VirtualPersistAPI Use Cases

VirtualPersistAPI (VPA) is conceived as persistent data storage for use with virtual worlds like Second Life.

Main use case

The main use case will be a script in Second Life that needs to persist some data.

This script will need to be able to:

- Persist and update/overwrite data (using a POST request)
- Query which data is available (using a GET request specifying category and key lookup)
- Query data (using a GET request)
- Delete data (using a DELETE request)

The amount of data to be stored per record is limited mainly by LSL's request limitations. Currently, the maximum request body size is 2048 bytes by default, but can be set to 4096 or 16384 bytes under some circumstances.

Other use cases

This system could be used by any consumer to persist arbitrary data. It could serve as a data store for non-LSL based systems, using the API.

VirtualPersistAPI Security Concerns

There are two main modes of external access to VPA:

1. Through the API.
2. Administration through web-based pages and forms.

API

Authentication for API-based access will be limited by the in-world capabilities of LSL. While it's theoretically possible to implement complex authentication methods, it's a bit of a practical nightmare.

However, LSL can generate a SHA1 hash, and so that's what we'll depend upon.

See the security documentation here: https://github.com/paul-m/VirtualPersistAPI/blob/master/spec/VPA_Security.md

Web-based

We have to guard against all of the standard web-based exploits, especially SQL injection.

SQL injection: VPA is implemented using Symfony2 and Doctrine. Doctrine is an entity- and database-abstraction layer, which uses PDO. We'll be using Doctrine's entity abstraction layer which parameterizes input to PDO.

Where we're not using the entity abstraction layer, we'll be using the DBAL's parameterized input system, and Doctrine's own query language (DQL). This has the advantage of making our code more portable, as well.

Doctrine security documentation: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/security.html