Homework 2 - Lower-Level Design

Paul Mitchum, paul@mile23.com 2013-04-13

Markdown version of this document here: https://github.com/paul-m/VirtualPersistAPI/blob/master/design/Homework2.md

Client Requirements

Definitions:

- Client: A network resource which is being used to consume or manage VirtualPersistAPI.

There are two main consumers: Clients making http requests to the API, and clients using a web browser to administer or manage the site.

API Request Client

API requests can come from any source over http. The main use case is from within the virtual world of Second Life, using the LSL scripting language. Another important use case is a client for testing the app.

Requirements:

- Able to perform HTTP requests.
- Capable of generating hashes for authentication.

Web Browser Client

Currently, VPA has meager needs in terms of web browser useage. This might change if we allow AJAX requests and/or a sophisticated user interface.

- Able to perform HTTP requests.
- Able to store cookies for session info.
- Allows the user to fill out forms.

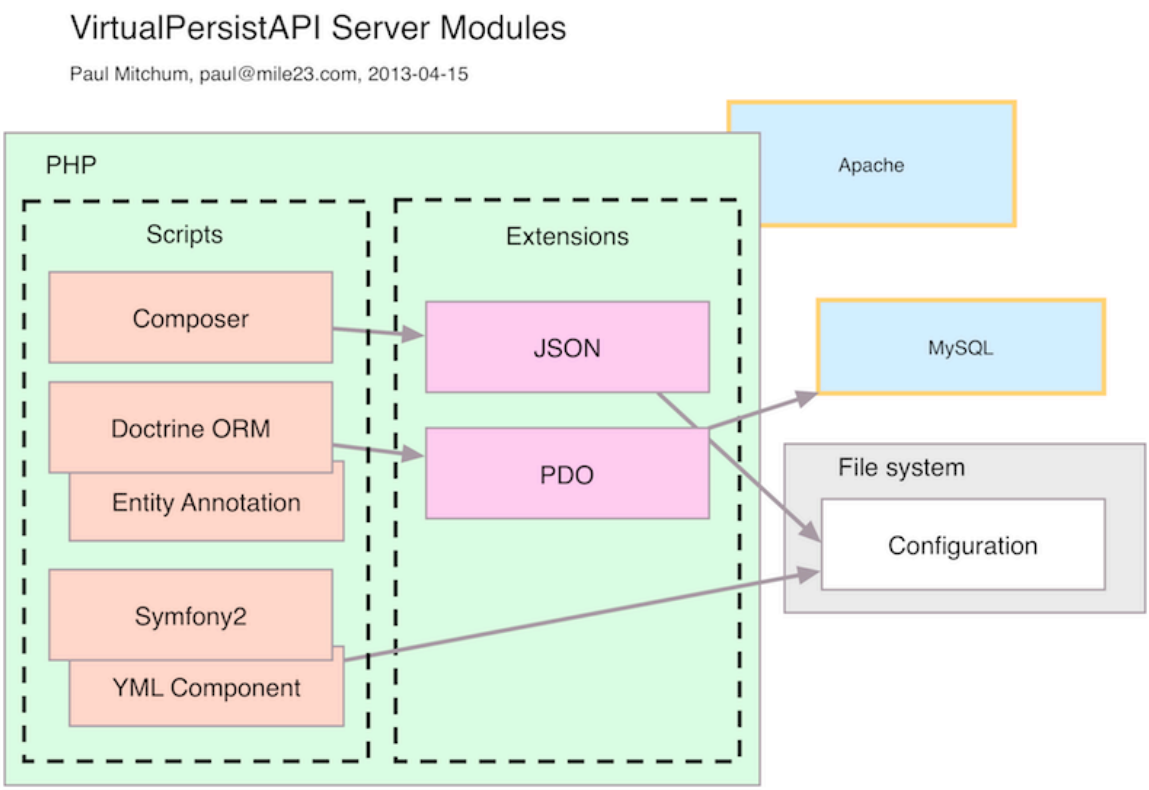
Server Requirements

A high-level requirements document exists here: https://github.com/paul-m/VirtualPersistAPI/blob/master/design/Requirements.md

VirtualPersistAPI is developed using Symfony2 and Doctrine, so it has the requirements of those packages.

Since we're using Composer to install these components and manage their dependencies, our most basic requirements are those of Composer.

Diagram



PHP

- v.5.3.3 or higher, we target 5.4.x.
- Extensions:

    - phar
    - json
    - PDO
- Configuration:
    - `detect_unicode = Off`
    - `allow_url_fopen = On`
    - `apc.enable_cli = Off`

Database

- Compatible with PHP's PDO
- We target MySQL v.5.5.x.

Apache

...is a requirement.

VirtualPersistAPI Entity Relationships

Currently...

There are two entities in this system thus far: User and Record. They are connected by a foreign key in the Record table (owner_id), which refers to the id of the User.

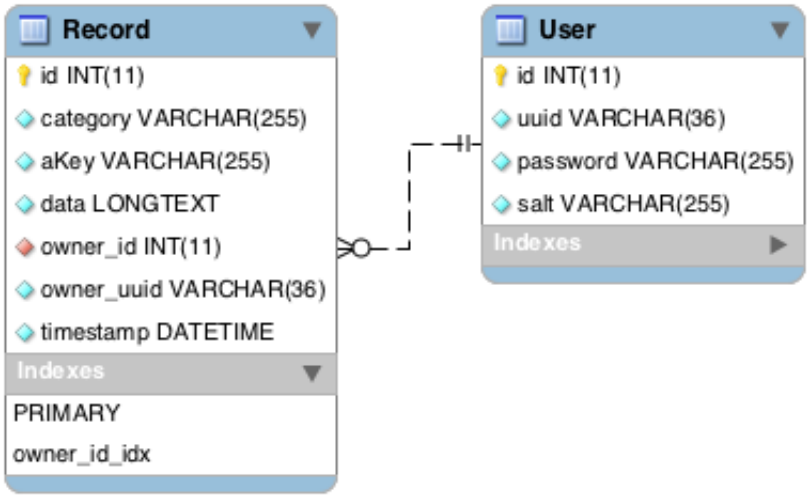
The entities are managed through the Doctrine ORM system, which uses PHP comment annotations to define entities, their tables, and their columns.

The PHP files which use this system are in this directory: https://github.com/paul-m/VirtualPersistAPI/tree/master/symfony/src/VirtualPersistAPI/VirtualPersistBundle/Entity

For instance, the Record entity is simply a class in PHP which has been marked as representing a table in the database. The many-to-one relationship between this entity and User is annotated this way:

```
/**
 * @ORM\Column(type="integer")
 * @ORM\ManyToOne(targetEntity="User")
 * @ORM\JoinColumn(name="owner_id", referencedColumnName="id", nullable=
false)
 */
protected $owner_id;
```

Here is a graphical diagram of the relationship between User and Record, generated with MySQL Workbench:



Here is some MySQL which generates these entities and their relationship:

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES'
;

CREATE SCHEMA IF NOT EXISTS `vpa` DEFAULT CHARACTER SET latin1 ;
USE `vpa` ;

-- -----------------------------------------------------
-- Table `vpa`.`User`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `vpa`.`User` (
  `id` INT(11) NOT NULL AUTO_INCREMENT ,
```

```
  `uuid` VARCHAR(36) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT
NULL ,
  `password` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
 NOT NULL ,
  `salt` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT
 NULL ,
  PRIMARY KEY (`id`) )
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_unicode_ci;

CREATE UNIQUE INDEX `user_uuid` ON `vpa`.`User` (`uuid` ASC) ;


-- -----------------------------------------------------
-- Table `vpa`.`Record`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `vpa`.`Record` (
  `id` INT(11) NOT NULL AUTO_INCREMENT ,
  `category` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
 NOT NULL ,
  `aKey` VARCHAR(255) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT
 NULL ,
  `data` LONGTEXT CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci' NOT NUL
L ,
  `owner_id` INT(11) NOT NULL ,
  `owner_uuid` VARCHAR(36) CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci
' NOT NULL ,
  `timestamp` DATETIME NOT NULL ,
  PRIMARY KEY (`id`) ,
  CONSTRAINT `owner_id`
    FOREIGN KEY (`owner_id` )
    REFERENCES `vpa`.`User` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_unicode_ci;

CREATE INDEX `owner_id_idx` ON `vpa`.`Record` (`owner_id` ASC) ;
```

```
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

What's Next?

In the future, we'll likely have a Roles table and a join table for it. This will be connected to the User table through the join table, so that Roles can stay normalized, and Users can have many Roles. This aspect of the design isn't finalized so maybe not.