VirtualPersistAPI

Final project for WEBLAMP 443
by Paul Mitchum

What Is VPA?

VirtualPersistAPI is a RESTful data storage API.

POST, GET, DELETE to URI:
/endpoint/uuid/category/key

```
/api/6d286553-59ae-409a-887d-ee75df67b834/someCategory/aKey
```

UUID represents the user.

Why is VPA?

VPA exists to allow people using virtual worlds to store data outside the virtual world system using http requests.

VPA can be general-purpose, but is designed with Second Life as a specific use-case.

Second Life avatars are identified by UUID.

Live demo?

VPA Architecture

Symfony2

Doctrine2

- ORM: Object Relational Mapper
- DBAL: DB Abstraction Layer

PHP in a LAMP stack.

SQLite for testing.

VPA Project Requirements:

Easy to support

Easily deployable

Class Project Requirements:

- Transactions
- Debuggging
- Metrics
- Triggers
- Stored Routines
  - Functions
  - Procedures
- Cursors
- Views

Transactions in Doctrine

Three types of transaction management in Doctrine:

Implicit: Transactions are managed by ORM.

Explicit: Force isolation through DBAL.

EZ Explicit: $em->transactional(\Closure) shorthand.

Transactions (cont.)

In Doctrine, use DBAL to set isolation level for a number of transactions.

```
$entityManager
  ->getConnection()
  ->setTransactionIsolation(
  Connection::TRANSACTION_SERIALIZABLE
  );
```

Transactions (cont.)

Doctrine ORM can also lock at the entity level, for read or write:

```
EntityManager#find(
    $className,
    $id,
    LockMode::PESSIMISTIC_READ)
```

Imposes SELECT .. FOR UPDATE

Debugging

Symfony2 provides a profiler and exception stack trace.

Response can be modified to include debug info in header. Ideally factored into Response class.

```
$reqDebug = $request->query->get('debug', 0);
if ($reqDebug) {
    $response->headers->set(
        'X-VPA-Debug', 'Some debuggy info.', TRUE);
}
```

Debugging (cont.)

Symfony2 also provides a profiler. Using a profiler token you can retrieve profile info for a request. The token is revealed in an X-Debug-Token header.

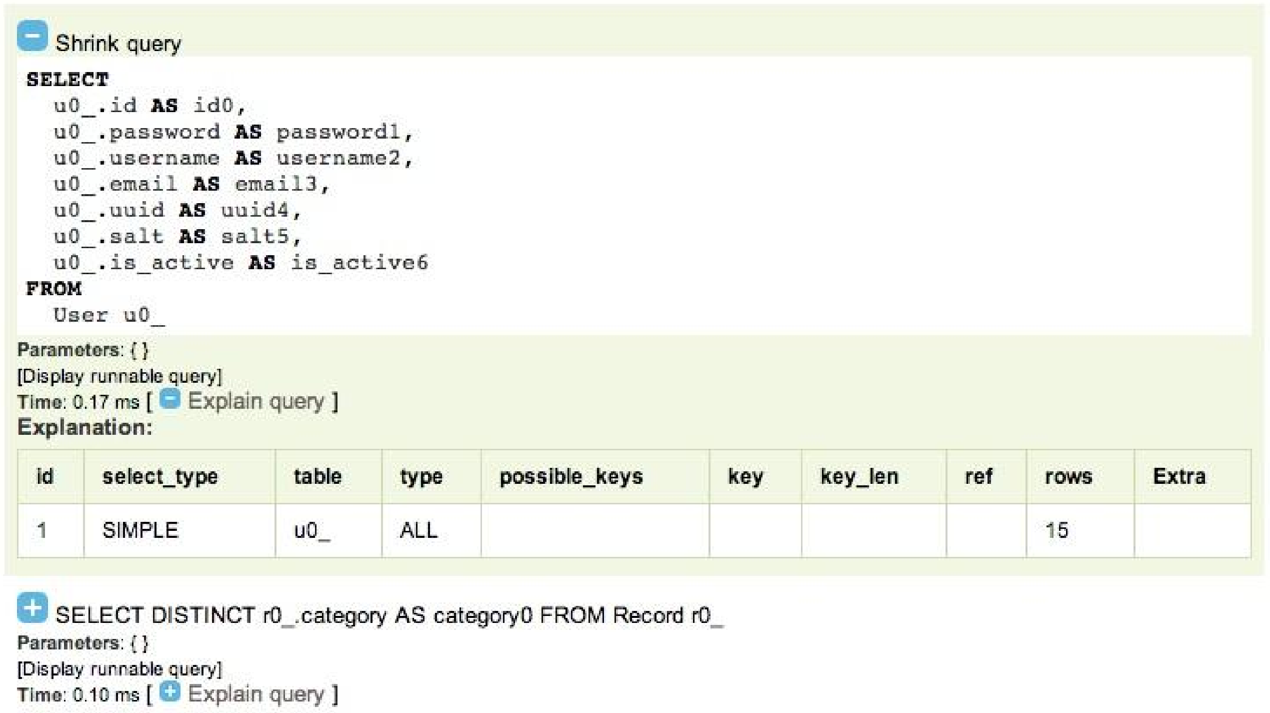
```
X-Debug-Token: de3200
```

This can be assembled into a URL:

```
http://example.com/_profiler/de3200
```

Debugging (cont.)

The profiler gives you a great deal of information, including a profile of all the Doctrine-related SQL queries.



Shrink query

```
SELECT
  u0_.id AS id0,
  u0_.password AS password1,
  u0_.username AS username2,
  u0_.email AS email3,
  u0_.uuid AS uuid4,
  u0_.salt AS salt5,
  u0_.is_active AS is_active6
FROM
  User u0_
```

Parameters: { }
[Display runnable query]
Time: 0.17 ms [Explain query]
Explanation:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	u0_	ALL					15	

SELECT DISTINCT r0_.category AS category0 FROM Record r0_
Parameters: { }
[Display runnable query]
Time: 0.10 ms [Explain query]

Views

Doctrine2 doesn't support views.
Doctrine1 did.

Relatively easy workaround:
use Doctrine\ORM\Query\ResultSetMapping;

Inform Doctrine of which result columns refer to entities it manages.

Views (cont.)

```
$rsm = new ResultSetMapping;
$rsm->addEntityResult(
  'VirtualPersistBundle:Record', 'r');
$rsm->addFieldResult('r', 'id', 'id');
...
```

Views (cont.)

Views downsides:
 * Deployment is a hassle.
 * My use case only uses the view as a prefabricated JOIN, so not worth the pain.
 * Not natively supported by Doctrine2.
 * Entity mapping only works for one entity type.

Triggers

Design decision:

Have a trigger delete content related to a user when the user account is deleted.

Triggers (cont.)

Cascading delete in Doctrine2 annotation:

```
/**
 * @ORM\ManyToOne(targetEntity="User")
 * @ORM\JoinColumn(name="owner",
        referencedColumnName="id",
        nullable=false,
        onDelete="CASCADE")
 */
protected $owner;
```

Doesn't work in SQLite, because Doctrine drivers phail at foreign key constraints.