

Zend_Auth Component Proposal - Darby Felton

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Auth Component Proposal

Proposed Component Name	Zend_Auth
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Auth
Proposers	Darby Felton
Revision	0.3.1 - 28 November 2006: Updated content post-approval (wiki revision: 21)

Table of Contents

- 1. Overview
- 2. References
- 3. Component Requirements, Constraints, and Acceptance Criteria
- 4. Dependencies on Other Framework Components
- 5. Theory of Operation
- Definitions
- Operation Details
- 6. Milestones / Tasks
- 7. Class Index
- 8. Use Cases
 - Website Login with E-mail Address and Password
 - Fallback Authentication with Security Question and Answer
 - Securing Sensitive Operations
 - Arbitrary Credentials
 - Credentials that Include the Identity
- 9. Class Skeletons

1. Overview



Vote Now for an Authentication Adapter!

In order to get an idea of which concrete authentication adapters are most needed, we have a ballot ready to accept your vote on which authentication adapter you would most like to see included with the framework. [Vote Now!](#)

Zend_Auth provides an API for generalized authentication and includes concrete authentication implementations.

Zend_Auth is concerned only with **authentication** and not **authorization**. Authentication is loosely defined as determining whether an entity actually is what it purports to be, based on some criteria. Authorization, the process of deciding whether to allow an entity access to, or to perform operations upon, other entities is out of scope with respect to this proposal. It is acknowledged that both authentication and authorization are integral parts of an application that are typically used together. We address authentication and authorization with separate framework components, however, with consideration for maximum flexibility for developers to integrate the concepts in accordance with their varied requirements. For more information about authorization with the Zend Framework, please see [Zend_Acl](#).

2. References

- [Phly Documentation: Class: Phly_Auth](#)
- [MediaWiki: AuthPlugin Class Reference](#)
- [PEAR::Auth](#)
- [Solar_Auth](#)

3. Component Requirements, Constraints, and Acceptance Criteria

- Provide an API to which arbitrary authentication systems can adhere
- Provide common concrete authentication mechanisms
- Encapsulate and make available data obtained during authentication with token interface
 - Identity and credentials (e.g., username, e-mail, one-time use keys)
 - Date and time
 - Other results from authentication back-end
- Facilitate authentication persistence across requests
- This component is only concerned with performing authentication; other operations, including management of authentication back end data stores, are out of scope.

4. Dependencies on Other Framework Components

- `Zend_Exception`
- `Zend_Session` (optional, on by default)

5. Theory of Operation

Definitions

For the purposes of this proposal,

- An **identity** is a piece of information used to represent the identity of the requester. For example, an authentication scheme might use a username or e-mail address for the purposes of identification, or perhaps the identity is an integer that maps to a database table.
- A **credential** is a piece of information used to authenticate the requester against the reported identity. For example, a password is often used as a credential in web applications. Similarly, security questions and answers (e.g., What was your first pet's name?) may be used as authentication credentials. In many cases, credentials include identifying information. For example, a fingerprint might be used alone as an authentication credential that indicates identity inherently.
- An **authentication token** refers to, represents, or encapsulates the results of an authentication attempt. For example, if a user logs in with a username and password, an application might use an object to refer to this authentication attempt, and this object would be an authentication token. An authentication token need not necessarily include the authentication credentials presented by the authentication attempt. The token would very likely contain the authenticated identity, and it may contain other information that is important to the application, such as the date and time of login, the number of failed login attempts, and other information from the authentication back-end.

Operation Details

Since authentication methods vary widely depending on the specific needs of an application, `Zend_Auth` imposes no particular back end authentication mechanism, though it is intended that it provides a set of concrete authentication adapters for the most common use cases.

We have an abstract class, `Zend_Auth_Adapter`, from which concrete authentication adapter classes inherit.

`Zend_Auth_Adapter` defines an `authenticate()` method that should be overridden by child classes, but this is not enforced by PHP because it is not designated as an abstract method. This, in turn, is because abstract method signatures must match their implemented

counterparts (in PHP 5.1.4), and we cannot know what the magic number of parameters would be for an `authenticate()` method, since actual authentication implementations are expected to vary widely. Thus, we resort to throwing an exception if the `authenticate()` method is called on an object inheriting from the abstract class that does not override `authenticate()`.

The `authenticate()` method of authentication adapter classes is intended to return an instance of a class that implements the `Zend_Authentication_Token_Interface`. Again, we're not enforcing the return type, but this intent is documented in the docblock for `Zend_Auth_Adapter::authenticate()`.

For some authentication backends and in certain circumstances (perhaps even in the most common use case), it may make sense to simply call a single static method upon the adapter class, providing authentication credentials, and returning authentication results as a token. The `staticAuthenticate()` method of the adapter class would be used when the credentials are known and the application is concerned only with the results of the authentication for a particular request.

`Zend_Auth_Digest` is included as an example concrete authentication adapter class (credit to [Gavin](#)).

6. Milestones / Tasks

1. Finish proposal revisions for review, based on feedback and collaboration - DONE
2. Arrive on approvable proposal through iterative review and updates - DONE
3. Implement approved design with inline API documentation - IN PROGRESS
4. Drive development with unit tests
5. Write manual documentation

7. Class Index

Common

- `Zend_Auth`
- `Zend_Auth_Exception`
- `Zend_Auth_Adapter` (abstract)
- `Zend_Auth_Adapter_Exception`
- `Zend_Auth_Token_Interface` (interface)

[Digest authentication Adapter](#)

- `Zend_Auth_Digest_Adapter`
- `Zend_Auth_Digest_Token`
- `Zend_Auth_Digest_Exception`

8. Use Cases

Website Login with E-mail Address and Password

Perhaps the most common use case for web applications is password-based authentication. In order to authenticate with a password, the user purports an identity, in this case, using an e-mail address to which the user has access. The user supplies the e-mail address to establish the identity against which the system challenges the user to authenticate using a password that only the person having access to the e-mail address is supposed to know.

Fallback Authentication with Security Question and Answer

It is often the case that users forget their passwords. Some applications may allow sending an e-mail to the identity that the user claims to be, so that the password may be reset to something the user will hopefully remember in the future. If the user is not who she claims to be, then the person that truly is the purported identity would receive the mailing and would simply opt not to change the password.

Other applications, however, may need to provide an alternative means of authentication, such as a security question and answer. To illustrate, suppose we have a user who forgets her password. If she had entered in her profile a security question and answer, then the application can simply ask the question, and the user would provide the correct answer in order to authenticate successfully.

"What was the name of your first pet," "on what street were you born," and "what was your high school mascot" are all examples of security questions that may be asked by an application. Though it may be important for application security not to let authentication hinge upon questions to which answers may be guessable or discoverable by entities other than the identity holder, these decisions are for application developers to

make and are outside of the scope of this proposal.

Securing Sensitive Operations

In certain situations it makes sense to require authentication credentials to be presented, regardless of the presence of an authentication token representing a previously successful authentication attempt.

For example, a user might log into a website with an e-mail address and password. The user clicks around various areas in the website, and her actions are performed with the identity which with she logged in, since an authentication token exists in her session and is recognized by the application. For extra security on sensitive operations, however, the website demands that the user enter her password again before being allowed to perform sensitive operations such as editing her profile or changing her password or e-mail address.

Arbitrary Credentials

In many instances, a simple username and password scheme is sufficiently secure for the purposes of the application. Other applications, however, may require more than a trivial set of credentials in order to consider a requester genuinely authenticated.

Consider the case of an application for credit or a loan, where the applicant must provide a series of information that is checked against existing records. There may be complicated sets of rules that must be fulfilled in order to completely authenticate a person.

An applicant would no doubt be required to provide a set of data that must be correct, such as her social security number (SSN), current address and phone number, and current bank information.

There may be other criteria, however, that provide some flexibility when the user cannot provide certain credentials required for authentication. For example, the applicant may have to answer 3 correct questions about her credit history, though some of the information may not be immediately available to the applicant. In this situation, the applicant may choose which of perhaps 10 questions to answer based on the information she does have at hand.

Credentials that Include the Identity

Authentication is often performed by first purporting an identity and then presenting credentials that are checked against the purported identity. But there are cases where separation of the identity from credentials may not be practical. Biometric and security card scanners are a couple of examples in which the credentials may inherently represent the identity of the requester, and the concepts of identity and credentials are not mutually exclusive.

9. Class Skeletons

```
class Zend_Auth
{
    /**
     * Authentication adapter
     *
     * @var Zend_Auth_Adapter
     */
    protected $_adapter;

    /**
     * Sets the authentication adapter
     *
     * @param Zend_Auth_Adapter $adapter
     * @return void
     */
    public function __construct(Zend_Auth_Adapter $adapter)
    {
        $this->_adapter = $adapter;
    }

    /**
     * Authenticates against the attached adapter
     *
     * If and only if $useSession is true, then the authentication token is saved to
     * the PHP session.
     *
     * All other parameters are passed along to the adapter's authenticate() method.
     */
}
```

```

*
* @param boolean $useSession
* @return Zend_Auth-Token_Interface
*/
public function authenticate($useSession = true)
{
    $args = func_get_args();
    $args = array_slice($args, 1);
    $token = call_user_func_array(array($this->_adapter, __METHOD__), $args);

    /**
     * @todo persist token in session if $useSession === true
     */

    return $token;
}
}

class Zend_Auth_Exception extends Zend_Exception
{}

abstract class Zend_Auth_Adapter
{
    /**
     * Extending classes may implement this method, accepting authentication
     * credentials as parameters, and returning the authentication results
     *
     * @throws Zend_Auth_Adapter_Exception
     * @return Zend_Auth-Token_Interface
     */
    public static function staticAuthenticate()
    {
        throw Zend::exception('Zend_Auth_Adapter_Exception',
            __METHOD__ . '() must be implemented by a concrete adapter class');
    }

    /**
     * Extending classes should implement this method, accepting authentication
     * credentials as parameters, and returning the authentication results
     *
     * @throws Zend_Auth_Adapter_Exception
     * @return Zend_Auth-Token_Interface
     */
    public function authenticate()
    {
        throw Zend::exception('Zend_Auth_Adapter_Exception',
            __METHOD__ . '() must be implemented by a concrete adapter class');
    }
}

class Zend_Auth_Adapter_Exception extends Zend_Auth_Exception
{}

interface Zend_Auth-Token_Interface
{
    /**
     * Returns whether the authentication token is currently valid (i.e., whether it
     * represents a successful authentication attempt)
     *
     * @return boolean
     */
    public function isValid();

    /**
     * Returns a message about why the authentication token is not valid

```

```

    * or null if the authentication token is valid
    *
    * @return string|null
    */
public function getMessage();

/**
 * Returns the identity represented by the authentication token
 *
 * @return mixed
 */
public function getIdentity();

/**
 * Sets the identity represented by the authentication token
 *
 * @param mixed $identity
 */
public function setIdentity($identity);
}

class Zend_Auth_Digest_Adapter extends Zend_Auth_Adapter
{
    /**
     * Creates a new digest authentication object against the $filename provided
     *
     * @param string $filename
     * @throws Zend_Auth_Digest_Exception
     */
    public function __construct($filename)
    {}

    /**
     * Authenticates against the given parameters
     *
     * @param string $filename
     * @param string $username
     * @param string $password
     * @param string $realm
     * @return Zend_Auth_Digest-Token
     */
    public static function staticAuthenticate($filename, $realm, $username, $password)
    {}

    /**
     * Authenticates the realm, username and password given
     *
     * @param string $realm
     * @param string $username
     * @param string $password
     * @throws Zend_Auth_Digest_Exception
     * @return Zend_Auth_Digest-Token
     */
    public function authenticate($realm, $username, $password)
    {}
}

class Zend_Auth_Digest-Token implements Zend_Auth-Token_Interface
{
    /**
     * Defined by Zend_Auth-Token_Interface
     *
     * @return boolean
     */
    public function isValid()
    {}
}

```

```
/**
 * Defined by Zend_Auth-Token_Interface
 *
 * @return string|null
 */
public function getMessage()
{}

/**
 * Defined by Zend_Auth-Token_Interface
 *
 * @return unknown_type
 */
public function getIdentity()
{}

/**
 * Defined by Zend_Auth-Token_Interface
 *
 * @param unknown_type $identity
 */
public function setIdentity($identity)
{}
}
```

```
class Zend_Auth_Digest_Exception extends Zend_Auth_Adapter_Exception
{
```

```
]]</ac:plain-text-body></ac:macro>
]]</ac:plain-text-body></ac:macro>
```