Session Handler

The framework contains some SESSION handlers as well. To use them, just create a new instance of the certain class once. The plugins will register a new session_set_save_handler (http://php.net/manual/en/function.session-set-save-handler.php) which syncs the frameworks SESSION (quick-reference#cookie,-get,-post,-request,-session,-files,-server,-env) hive keys to the corresponding new session handler class.

Engines

Cache

Namespace: `\`
File location: `lib/session.php`

The Session class provides a lightweight Cache (cache)-based session handler.

Make sure you have enabled the cache (quick-reference#cache) to make this work. Usage:

```
// just create an object
new Session();

$f3->set('SESSION.test',123);
echo $f3->get('SESSION.test');
```

The constructor accepts a few optional arguments:

```
new Session( [ callable $onsuspect [, string $key [, Cache $cache ]]] )
```

The `$onsuspect` argument is there to override the default handling of suspicious sessions: when an IP or agent change is detected during the session, the session is destroyed and a 403 error is thrown.

To override this behaviour, you can pass your own suspect session handler to the constructor:

```
new Session(function(Session $session,$id){
  // Session $id is suspect, do something
  // you can read $session->agent(), $session->ip(), $session->csrf() and $session->s
tamp()
  // and/or return FALSE to trigger the default 403 error
});
```

The `$key` argument is there to store the CSRF (session#csrf) token into a hive variable:

```
new Session(NULL,'CSRF');
echo $f3->CSRF; // token here
```

The `$cache` argument makes it possible to store the session data in a dedicated Cache instance, rather than the default one:

```
$cache=Cache::instance(); // Default cache (the one defined by the CACHE variable)
$sessionCache=new Cache('folder=var/sessions/'); // Session cache
new Session(NULL,NULL,$sessionCache);
```

SQL

This class provides a SQL-based session handler.

Namespace: `\DB\SQL`
File location: `lib/db/sql/session.php`

Assuming that you have a working SQL DB (sql) object in the `DB` hive key, use:

```
$db = $f3->get('DB');
// just create an object
new \DB\SQL\Session($db);

$f3->set('SESSION.test',123);
echo $f3->get('SESSION.test');
```

It will automatically create the required table, if the `session` table does not exist. The table name can be controlled by the 2nd `$table` parameter of the constructor. A 3rd parameter is used to `$force` the table creation, if it's not existing.

Mongo

This class provides a Mongo-based session handler.

Namespace: `\DB\Mongo`
File location: `lib/db/mongo/session.php`

Assuming that you have a working Mongo DB (mongo) object in the `DB` hive key, use:

```
$db = $f3->get('DB');
// just create an object
new \DB\Mongo\Session($db);

$f3->set('SESSION.test',123);
echo $f3->get('SESSION.test');
```

The table name can be controlled by the 2nd `$table` parameter of the constructor.

Jig

This class provides a JIG-based session handler.

Namespace: `\DB\Jig`
File location: `lib/db/jig/session.php`

Assuming that you have a working Jig DB (jig) object in the `DB` hive key, use:

```
$db = $f3->get('DB');
// just create an object
new \DB\JIG\Session($db);

$f3->set('SESSION.test',123);
echo $f3->get('SESSION.test');
```

The table name can be controlled by the 2nd `$table` parameter of the constructor.

Methods

agent

Return HTTP user agent

```
string|FALSE agent( )
```

This method returns the browser's user-agent that was used by the client when the current session was created.

csrf

Return anti-CSRF token

```
string|FALSE csrf( )
```

This method returns a CSRF token which is bound to the current active Session and the current request. Use this token to protect your application from CSRF attacks. The constructor of the Session classes already performs some checks to prevent Session hijacking, but it is recommended that you implement some extra validation to shield your app from URL-Spoofing and CSRF (https://en.wikipedia.org/wiki/Cross-site_request_forgery) attacks.

Here's how to use it:

1) First call the `csrf()` method to get the current session token and store it somewhere:

```
$sess=new DB\SQL\Session($db);
$f3->CSRF=$sess->csrf();
```

NB: as an alternative, you can instantiate the Session class with the 5th parameter set to a hive key name, which will hold the CSRF token. E.g:

```
new DB\SQL\Session($db,'sessions',TRUE,NULL,'CSRF');// now $f3->CSRF holds the token
```

2) Save that token to session:

```
$f3->copy('CSRF','SESSION.csrf');// the variable name is up to you
```

3) Add that token to your form:

```
<input type="hidden" name="token" value="{{ @CSRF }}">
```

4) On form submission, compare the received token with the value stored in session:

```
$token = $f3->get('POST.token');
$csrf = $f3->get('SESSION.csrf');
if (empty($token) || empty($csrf) || $token!==$csrf) {
  // CSRF attack detected
}
```

The complete example looks like this:

```
$f3->DB=new DB\SQL('mysql:host=127.0.0.1;port=3306;dbname=test;','user','p4ssw0rd');

$f3->route('GET|POST /test-csrf',function($f3,$params){

    new DB\SQL\Session($f3->DB,'sessions',TRUE,NULL,'CSRF');
    // or:
    // $sess=new DB\SQL\Session($f3->DB);
    // $f3->CSRF=$sess->csrf();

    if ($f3->VERB=='POST') {
        $token = $f3->get('POST.token');
        $csrf = $f3->get('SESSION.csrf');
        if (empty($token) || empty($csrf) || $token!==$csrf) {
            echo 'CSRF attack!';
        else
            echo 'Your name is '.$f3->get('POST.name');
        die();
    }

    $f3->copy('CSRF','SESSION.csrf');

    echo '<form action="" method="post">'.
        '<input type="text" name="name" value="" placeholder="Your name"/>'.
        '<input type="hidden" name="token" value="'.$f3->CSRF.'"/>'.
        '<button type="submit">Submit</button></form>';

});
```

ip

Return IP address

```
string|FALSE ip( )
```

This method returns the IP address that was used by the client when the current session was created.

stamp

Return Unix timestamp

```
string|FALSE stamp( )
```

This method returns the "last updated at" unix timestamp of the current session.

Events

onsuspect

Custom callback for suspect sessions

```
// Log suspicious sessions and destroy them.
new \DB\SQL\Session($db,'sessions',TRUE,function($session){
  // Suspect session
  $logger = new \Log('logs/session.log');
  $f3=\Base::instance();
  if (($ip=$session->ip())!=$f3->get('IP'))
    $logger->write('user changed IP:'.$ip);
  else
    $logger->write('user changed browser/device:'.$f3->get('AGENT'));

  // The default behaviour destroys the suspicious session.
  return false;
});
```

The default behaviour destroys the session and throws a `HTTP 403` error when a suspect session is detected (wrong IP-Address or User-Agent). To extend or overwrite that behaviour, you can use the fourth constructor parameter to define a callback function. If the callback returns `FALSE` the default behaviour gets also executed.

Notice: The framework doesn't verify the CSRF token automatically, you have to do it manually (session#csrf).