```php
<?php

if (!defined('ROUTE')) $route  = include '../Routage.php';

class getListe extends Action { //======================================== objec
t user
  static public $time;
  static public $userID;

  public    function __construct($route) {
    parent::__construct($route);
  }

  protected function isParametersValid(){

    self::isvalid_UserIdentity();
    self::isValid_Time();

    return $this;
  }
  protected function askBdD() {

    $Sql = new Sql();
    $Sql->select()
        ->from  ('sessions')
        ->orderBy('end ASC');

    self::db_Time($Sql);
    self::db_UserIdentity($Sql);

    $datas = DB::query($Sql)->fetchAll(PDO::FETCH_ASSOC);

    Error::showIfFalse($datas);

    self::db_Format($datas);
  }

  static private function isvalid_UserIdentity(){

    if (self::isParameterDefined('userIdentity')) {

      //Lecture du UserID
      $user = json_decode(self::get(
        'DB/users',
        'get',
        array(
          'userIdentity' => self::$datas['userIdentity'],
          'field'        => 'id')));

      //Traitement des erreurs
      if (is_null($user) || !is_null($user->errors))
        Error::show(self::$datas['userIdentity']);
      else
        self::$userID = $user->datas;

    //Par defaut tous
    } else
      self::$userID = 'all';
  }
  static private function isValid_Time(){
    Error::showIfValueParameterNotAutorized(
      'time',
      array('past', 'futur','all'),
      Error::OPTIONAL);
  }
```

```
  static private function db_Time($Sql) {
    switch (self::$time) {
      case 'past':
        $Sql->where ('start <= "' . strtotime('now') . '"');
        break;
      case 'futur':
        $Sql->where ('end >= "' . strtotime('now') . '"');
        break;
    }
  }
  static private function db_UserIdentity($Sql) {
    if (self::$userID != 'all')
      Error::showNotFinish('userIdentity');
  }
  static private function db_Format($datas) {
    if (count($datas) > 0) {
      foreach($datas as $no => $data){
        $datas[$no]['start'] = new DateTime("@" . $data['start']);
        $datas[$no]['end']   = new DateTime("@" . $data['end']);
      }
    }
    self::$datas = $datas;
  }

  static protected function initClass() {
    Error::initClass('{
      "messages"    : {
        "askBdD"               : [ "Requete Impossible [_data_]" ],
        "isvalid_UserIdentity" : [ "Identite inconnue [userIdentity = _data_]"
],
        "isValid_Time"         : [ "Parametre invalide [time = _data_]" ]
      }
    }');
    self::setRequestMode(self::GET);
  }
}

class InfoAPI {
  public function __construct(){
    echo
<<<EOT
<pre>
sessions

* getListe

-----------------------------------
getListe([userIdentity:string, time:string])
  Retourne la liste des sessions
  Format JSON : tableau de string : [id, start, end]

Parametres optionnels
- userIdentity : limitation a un membre (identite:string)
- time : [futur/past] : limitation au futur ou au passe

En general:
-----------
Retourne un dictionnaire (cle/aleur) JSON avec :
- errors : null ou string = message de l'erreur recontree
- datas  : null ou donnees retournees par la fonction

exemples :
  {"errors":null,"datas":[1,2,3]}
  {"errors":"Impossible de calculer", datas:null}
```

```
</pre>
EOT;
  }
}

$route->go('getListe');
```