

```
<?php
/*
 * Auteur      : Hoarau Nicolas
 * Projet      : WE GO
 * Page        : backend.php
 * Date        : 26.05.2020
 * Description  : Fichier qui contient toutes les fonctionsdu site.
 * Version     : 1.0
 */

/* Import PHPMailer classes into the global namespace
These must be at the top of your script, not inside a function*/

use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;

// Load Composer's autoloader
require dirname(dirname(__DIR__)) . '/vendor/autoload.php';
require_once dirname(__DIR__) . '/Controller/DatabaseController.php';
require_once dirname(dirname(__DIR__)) . '/config/config.php';

/**
 * @date 26.05.20
 * @author Hoarau Nicolas
 * @brief Fonction qui vérifie si l'utilisateur est connecté
 * @return boolean
 * @version 1.0
 */
function IsLogged(): bool
{
    return array_key_exists('loggedIn', $_SESSION) && $_SESSION['loggedIn'];
}

/**
 * @author Hoarau Nicolas
 * @date 27.05.2020
 * @brief Fonction qui récupère le salt de l'utilisateur
 * @param array $args
 * @return string
 * @version 1.0
 */
function GetSalt(array $args): ?string
{
    // initialise à null les colonnes du tableau vide/inexistante
    $args += [
        'userEmail' => null,
        'userNickname' => null
    ];

    extract($args); // extrait les données du tableau avec comme nom de varvariable son
    nom de colonne

    $clauseField = "";
```

```
54 $clauseValue = "";
55
56 if ($userEmail !== null) {
57     $clauseField = "email";
58     $clauseValue = $userEmail;
59 } elseif ($userNickname) {
60     $clauseField = "pseudo";
61     $clauseValue = $userNickname;
62 }
63
64 $query = <<<EX
65     SELECT salt
66     FROM user
67     WHERE `{ $clauseField }` = :clauseValue;
68     EX;
69
70 try {
71     $requestSalt = DatabaseController::prepare($query);
72     $requestSalt->bindParam(':clauseValue', $clauseValue, PDO::PARAM_STR);
73     $requestSalt->execute();
74
75     $result = $requestSalt->fetch(PDO::FETCH_ASSOC);
76
77     return $result !== false ? $result['salt'] : null;
78 } catch (PDOException $e) {
79     return null;
80 }
81 }
82
83 /**
84  * @author Hoarau Nicolas
85  * @date 26.05.2020
86  * @brief Fonction qui créer un nouvel utilisateur dans la base de données
87  * @param string $nickname
88  * @param string $lastname
89  * @param string $firstname
90  * @param string $email
91  * @param string $phoneNumber
92  * @param string $password
93  * @return boolean
94  * @version 1.0
95  *     Inscription sans salt
96  * @version 1.1
97  *     Inscription avec salt
98  */
99 function Register(string $nickname, string $firstname, string $email, string
    $password, string $lastname = null, string $phoneNumber = null): bool
100 {
101     $query = "";
102     $salt = hash('sha256', microtime());
103     $userPassword = hash('sha256', $password . $salt);
104
105     if ($lastname != null && $phoneNumber != null) {
106         $query = <<<EX
107             INSERT INTO user (pseudo, prenom, nom, email, password, telephone, salt)
```

```
108     VALUES (:nickname, :firstname, :lastname, :email, :password, :phoneNumber,
:salt);
109     EX;
110 } elseif ($phoneNumber != null) {
111     $query = <<<EX
112     INSERT INTO user (pseudo, prenom, nom, email, password, telephone, salt)
113     VALUES (:nickname, :firstname, :email, :password, :phoneNumber, :Salt);
114     EX;
115 } elseif ($lastname != null) {
116     $query = <<<EX
117     INSERT INTO user (pseudo, prenom, nom, email, password, salt)
118     VALUES (:nickname, :firstname, :lastname, :email, :password, :salt);
119     EX;
120 } else {
121     $query = <<<EX
122     INSERT INTO user (pseudo, prenom, email, password, salt)
123     VALUES (:nickname, :firstname, :email, :password, :salt);
124     EX;
125 }
126
127 try {
128     DatabaseController::beginTransaction();
129
130     $requestRegister = DatabaseController::prepare($query);
131     $requestRegister->bindParam(':nickname', $nickname, PDO::PARAM_STR, 50);
132     $requestRegister->bindParam(':firstname', $firstname, PDO::PARAM_STR, 50);
133
134     if ($lastname != null)
135         $requestRegister->bindParam(':lastname', $lastname, PDO::PARAM_STR, 50);
136
137     $requestRegister->bindParam(':email', $email, PDO::PARAM_STR, 100);
138     $requestRegister->bindParam(':password', $userPassword, PDO::PARAM_STR, 100);
139
140     if ($phoneNumber != null)
141         $requestRegister->bindParam(':phoneNumber', $phoneNumber, PDO::PARAM_STR, 50);
142
143     $requestRegister->bindParam(':salt', $salt, PDO::PARAM_STR, 64);
144
145     $requestRegister->execute();
146
147     DatabaseController::commit();
148     return true;
149 } catch (PDOException $e) {
150     DatabaseController::rollBack();
151     return false;
152 }
153 }
154
155 /**
156  * @author Hoarau Nicolas
157  * @date 26.05.2020
158  * @brief Fonction qui connecte l'utilisateur
159  * @param array les données de connexion de l'utilisateur
160  * @return array|false
161  * @version 1.0
```

```
162 *      Inscription sans salt
163 * @version 1.1
164 *      Inscription avec salt
165 */
166 function Login(array $args)
167 {
168     // initialise à null les colonnes du tableau vide/inexistante
169     $args += [
170         'userEmail' => null,
171         'userNickname' => null,
172         'userPwd' => null,
173     ];
174
175     extract($args); // extrait les données du tableau avec comme nom de variable son
    nom de colonne
176
177     $loginField = "";
178     $authenticator = "";
179     $salt = "";
180
181     if ($userEmail !== null) {
182         $salt = GetSalt(['userEmail' => $userEmail]);
183         $authenticator = $userEmail;
184         $loginField = "email";
185     } elseif ($userNickname !== null) {
186         $salt = GetSalt(['userNickname' => $userNickname]);
187         $authenticator = $userNickname;
188         $loginField = "pseudo";
189     } else {
190         return false;
191     }
192
193     if ($userPwd == null)
194         return false;
195
196     $pwd = hash('sha256', $userPwd . $salt);
197
198     $query = <<<EX
199     SELECT idUser, pseudo, prenom, nom, email, telephone
200     FROM user
201     WHERE `{ $loginField }` = :wayToConnectValue
202     AND password = :pwd;
203     EX;
204
205     try {
206         $requestLogin = DatabaseController::prepare($query);
207         $requestLogin->bindParam(':wayToConnectValue', $authenticator, PDO::PARAM_STR);
208         $requestLogin->bindParam(':pwd', $pwd, PDO::PARAM_STR);
209         $requestLogin->execute();
210
211         $result = $requestLogin->fetch(PDO::FETCH_ASSOC);
212
213         return $result !== false & $result > 0 ? $result : false;
214     } catch (PDOException $e) {
215         return false;
216     }
```

```
216 }
217 }
218
219 /**
220  * @author Hoarau Nicolas
221  * @date 26.05.2020
222  * @brief Fonction qui vérifie si le paramètre donnée(email, nickname) est déjà
    utilisé ou non
223  * @param array $args
224  * @return boolean|null
225  * @version 1.0
226  */
227 function IsTaken(array $args): ?bool
228 {
229     $args += [
230         'userEmail' => null,
231         'userNickname' => null,
232         'idUser' => null
233     ];
234
235     extract($args);
236
237     $authenticator = "";
238     $field = "";
239     if ($userEmail != null) {
240         $authenticator = $userEmail;
241         $field = "email";
242     } elseif ($userNickname != null) {
243         $authenticator = $userNickname;
244         $field = "pseudo";
245     }
246
247     $queryWithoutIdUser = <<<EX
248         SELECT `{ $field }`
249         FROM user
250         WHERE `{ $field }` = :authenticator;
251     EX;
252
253     $queryWithIdUser = <<<EX
254         SELECT `{ $field }`
255         FROM user
256         WHERE `{ $field }` = :authenticator
257         AND idUser <> :idUser;
258     EX;
259
260     $query = ($idUser != null) ? $queryWithIdUser : $queryWithoutIdUser;
261
262     try {
263         $requestIsUsed = DatabaseController::prepare($query);
264         $requestIsUsed->bindParam(':authenticator', $authenticator, PDO::PARAM_STR);
265
266         if ($idUser != null)
267             $requestIsUsed->bindParam(':idUser', $idUser, PDO::PARAM_INT);
268
269     }
```

```
270     $requestIsUsed->execute();
271     $result = $requestIsUsed->fetch(PDO::FETCH_ASSOC);
272
273     return $result !== false ? true : false;
274 } catch (PDOException $e) {
275     return null;
276 }
277 }
278
279 /**
280  * @author Hoarau Nicolas
281  * @date 27.05.2020
282  * @brief Fonction qui crée un événement
283  * @param string $eventName
284  * @param string $eventDescription
285  * @param string $place
286  * @param Date $beginningDate
287  * @param Date $endDate
288  * @param integer $nbMaxGuest
289  * @param FILES $img
290  * @param integer $idUser
291  * @return boolean
292  * @version 1.0
293  */
294 function CreateEvent(string $eventName, string $eventDescription, string $place,
    $beginningDate, $endDate, int $nbMaxGuest, $img, int $type, int $idUser, array
    $guestlist = null): bool
295 {
296     $creationDate = date('Y-m-d H:i');
297     $beginningDate = date("Y-m-d H:i", strtotime($beginningDate));
298     $endDate = date("Y-m-d H:i", strtotime($endDate));
299
300     $query = <<<EX
301     INSERT INTO evenement (nom, descriptif, dateDebut, dateFin, dateCreation, lieu,
    prive, nbMaxParticipant, urlImage, idOrganisateur)
302     VALUES (:name, :description, :beginningDate, :endDate, :creationDate, :place,
    :type, :nbMaxGuest, :urlImg, :idUser);
303     EX;
304
305     try {
306         DatabaseController::beginTransaction();
307
308         $requestCreateEvent = DatabaseController::prepare($query);
309         $requestCreateEvent->bindParam(':name', $eventName, PDO::PARAM_STR, 45);
310         $requestCreateEvent->bindParam(':description', $eventDescription, PDO::PARAM_STR,
    255);
311         $requestCreateEvent->bindParam(':beginningDate', $beginningDate);
312         $requestCreateEvent->bindParam(':endDate', $endDate);
313         $requestCreateEvent->bindParam(':creationDate', $creationDate);
314         $requestCreateEvent->bindParam(':place', $place, PDO::PARAM_STR, 255);
315         $requestCreateEvent->bindParam(':type', $type, PDO::PARAM_INT, 1);
316         $requestCreateEvent->bindParam(':nbMaxGuest', $nbMaxGuest, PDO::PARAM_INT, 11);
317         $requestCreateEvent->bindParam(':urlImg', $img, PDO::PARAM_STR, 255);
318         $requestCreateEvent->bindParam(':idUser', $idUser, PDO::PARAM_INT, 11);
319
```

```
320     $requestCreateEvent->execute();
321
322     if ($guestlist != null) {
323         $lastInsertId = DatabaseController::getInstance()->lastInsertId();
324
325         for ($i = 0; $i < count($guestlist); $i++) {
326             if ($idUser != $guestlist[$i]) {
327                 if (CreateInvite($guestlist[$i], $lastInsertId) == false) {
328                     DatabaseController::rollBack();
329                 }
330             }
331         }
332
333         DatabaseController::commit();
334         return true;
335     }
336
337     DatabaseController::commit();
338     return true;
339 } catch (PDOException $e) {
340     DatabaseController::rollBack();
341     return false;
342 }
343 }
344
345 /**
346  * @author Hoarau Nicolas
347  * @date 27.05.2020
348  * @brief Fonction qui crée une invitation pour un utilisateur à un événement
349  * @param integer $idGuest
350  * @param integer $idEvent
351  * @return boolean
352  * @version 1.0
353  */
354 function CreateInvite(int $idGuest, int $idEvent): bool
355 {
356     $query = <<<EX
357         INSERT INTO invites (idUser, idEvenement)
358         VALUES (:idGuest, :idEvent);
359     EX;
360
361     try {
362         DatabaseController::beginTransaction();
363
364         $requestCreateInvite = DatabaseController::prepare($query);
365         $requestCreateInvite->bindParam(':idGuest', $idGuest, PDO::PARAM_INT, 11);
366         $requestCreateInvite->bindParam(':idEvent', $idEvent, PDO::PARAM_INT, 11);
367         $requestCreateInvite->execute();
368
369         DatabaseController::commit();
370
371         return true;
372     } catch (PDOException $e) {
373         DatabaseController::rollBack();
374         return false;
375     }
376 }
```

```
375 }
376 }
377
378 /**
379  * @author Hoarau Nicolas
380  * @date 05.06.2020
381  * @brief Fonction qui supprime l'invitation d'un utilisateur à un événement
382  * @param integer $idGuest
383  * @param integer $idEvent
384  * @return boolean
385  * @version 1.0
386  */
387 function DeleteInvite(int $idGuest, int $idEvent): bool
388 {
389     $query = <<<EX
390     DELETE FROM invites WHERE idUser = :idUser AND idEvenement = :idEvent;
391     EX;
392
393     try {
394         DatabaseController::beginTransaction();
395
396         $requestDeleteInvite = DatabaseController::prepare($query);
397         $requestDeleteInvite->bindParam(':idUser', $idGuest, PDO::PARAM_INT, 11);
398         $requestDeleteInvite->bindParam(':idEvent', $idEvent, PDO::PARAM_INT, 11);
399         $requestDeleteInvite->execute();
400
401         DatabaseController::commit();
402
403         return true;
404     } catch (PDOException $e) {
405         DatabaseController::rollBack();
406         return false;
407     }
408 }
409
410 /**
411  * @author Hoarau Nicolas
412  * @date 02.06.2020
413  * @brief Fonction qui vérifie si l'événement existe
414  * @param integer $idEvent
415  * @return boolean
416  * @version 1.0
417  */
418 function EventExist(int $idEvent): bool
419 {
420     $query = <<<EX
421     SELECT COUNT(idEvenement) AS exist
422     FROM evenement
423     WHERE idEvenement = :idEvent
424     EX;
425
426     try {
427         $requestExist = DatabaseController::prepare($query);
428         $requestExist->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
429         $requestExist->execute();
```



```
430
431     $result = $requestExist->fetch(PDO::FETCH_ASSOC);
432
433     return $result['exist'] == 0 ? false : true;
434 } catch (PDOException $e) {
435     return false;
436 }
437 }
438
439 /**
440  * @author Hoarau Nicolas
441  * @date 02.06.2020
442  * @brief Fonction qui vérifie si l'utilisateur participa à l'événement
443  * @param integer $idEvent
444  * @param integer $idUser
445  * @return boolean
446  * @version 1.0
447  */
448 function IsParticipating(int $idEvent, int $idUser): bool
449 {
450     $query = <<<EX
451     SELECT COUNT(*) AS participating
452     FROM inscriptions
453     WHERE idUser = :idUser
454     AND idEvenement = :idEvent
455     EX;
456
457     try {
458         $requestIsParticipating = DatabaseController::prepare($query);
459         $requestIsParticipating->bindParam(':idUser', $idUser, PDO::PARAM_INT);
460         $requestIsParticipating->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
461         $requestIsParticipating->execute();
462
463         $result = $requestIsParticipating->fetch(PDO::FETCH_ASSOC);
464
465         return $result['participating'] == 0 ? false : true;
466     } catch (PDOException $e) {
467         return false;
468     }
469 }
470
471 /**
472  * @author Hoarau Nicolas
473  * @date 02.06.2020
474  * @brief Fonction qui vérifie si l'événement est privé
475  * @param integer $idEvent
476  * @return boolean
477  * @version 1.0
478  */
479 function IsPrivate(int $idEvent): bool
480 {
481     $query = <<<EX
482     SELECT prive
483     FROM evenement
484     WHERE idEvenement = :idEvent;
```

```
485     EX;
486
487     try {
488         $requestIsPrivate = DatabaseController::prepare($query);
489         $requestIsPrivate->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
490         $requestIsPrivate->execute();
491
492         $result = $requestIsPrivate->fetch(PDO::FETCH_ASSOC);
493
494         return $result['prive'] == 0 ? false : true;
495     } catch (PDOException $e) {
496         return false;
497     }
498 }
499
500 /**
501  * @author Hoarau Nicolas
502  * @date 02.06.2020
503  * @brief Fonction qui vérifie si l'utilisateur est invité à l'événement donnée
504  * @param integer $idEvent
505  * @param integer $idUser
506  * @return boolean
507  * @version 1.0
508  */
509 function IsInvited(int $idEvent, int $idUser): bool
510 {
511     $query = <<<EX
512     SELECT COUNT(*) AS isInvited
513     FROM invites
514     WHERE idUser = :idUser
515     AND idEvenement = :idEvent;
516     EX;
517
518     try {
519         $requestIsInvited = DatabaseController::prepare($query);
520         $requestIsInvited->bindParam(':idUser', $idUser, PDO::PARAM_INT);
521         $requestIsInvited->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
522         $requestIsInvited->execute();
523         $result = $requestIsInvited->fetch(PDO::FETCH_ASSOC);
524
525         return $result['isInvited'] == 1 ? true : false;
526     } catch (PDOException $e) {
527         return false;
528     }
529 }
530
531 /**
532  * @author Hoarau Nicolas
533  * @date 02.06.2020
534  * @brief Fonction qui vérifie si l'utilisateur est le créateur de l'événement donnée
535  * @param integer $idEvent
536  * @param integer $idUser
537  * @return boolean
538  * @version 1.0
539  */
```

```
540 function IsCreator(int $idEvent, int $idUser): bool
541 {
542     $query = <<<EX
543     SELECT COUNT(*) AS isCreator
544     FROM evenement
545     WHERE idOrganisateur = :idUser
546     AND idEvenement = :idEvent;
547     EX;
548
549     try {
550         $requestIsCreator = DatabaseController::prepare($query);
551         $requestIsCreator->bindParam(':idUser', $idUser, PDO::PARAM_INT);
552         $requestIsCreator->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
553         $requestIsCreator->execute();
554
555         $result = $requestIsCreator->fetch(PDO::FETCH_ASSOC);
556
557         return $result['isCreator'] == 1 ? true : false;
558     } catch (PDOException $e) {
559         return false;
560     }
561 }
562
563 /**
564  * @author Hoarau Nicolas
565  * @date 02.06.2020
566  * @brief Fonction qui récupère les information de l'événement
567  * @param integer $idEvent
568  * @param integer $idUser
569  * @return array|false
570  * @version 1.0
571  */
572 function GetEventData(int $idEvent, int $idUser = null)
573 {
574     $queryLogged = <<<EX
575     SELECT e.idEvenement, e.nom, e.descriptif, e.dateDebut, e.dateFin, e.lieu,
576     e.nbMaxParticipant, e.prive, e.urlImage, u.pseudo, (SELECT COUNT(*) FROM inscriptions
577     WHERE idUser = :idUser AND idEvenement = :idEvent) AS participating, (SELECT
578     COUNT(idUser) FROM inscriptions WHERE idevenement = :idEvent) AS nbGuest
579     FROM evenement AS e
580     JOIN user AS u ON e.idOrganisateur = u.idUser
581     WHERE e.idEvenement = :idEvent;
582     EX;
583
584     $queryNotLogged = <<<EX
585     SELECT e.idEvenement, e.nom, e.descriptif, e.dateDebut, e.dateFin, e.lieu,
586     e.nbMaxParticipant, e.prive, e.urlImage, u.pseudo, (SELECT COUNT(idUser) FROM
587     inscriptions WHERE idEvenement = :idEvent) AS nbGuest
588     FROM evenement AS e
589     JOIN user AS u ON e.idOrganisateur = u.idUser
590     WHERE e.idEvenement = :idEvent;
591     EX;
592
593     $query = $idUser !== null ? $queryLogged : $queryNotLogged;
```

```
590 try {
591     $requestGetEventData = DatabaseController::prepare($query);
592
593     if ($idUser !== null)
594         $requestGetEventData->bindParam(':idUser', $idUser, PDO::PARAM_INT);
595
596     $requestGetEventData->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
597     $requestGetEventData->execute();
598
599     $result = $requestGetEventData->fetch(PDO::FETCH_ASSOC);
600
601     return $result !== false ? $result : null;
602 } catch (PDOException $e) {
603     return false;
604 }
605 }
606
607 /**
608  * @author Hoarau Nicolas
609  * @date 02.06.2020
610  * @brief Fonction qui envoie un mail à un destinataire précit
611  * @param string $recipientMail
612  * @param string $message
613  * @return bool
614  * @version 1.0
615  */
616 function SendMail(string $recipientMail, string $message, string $subject): bool
617 {
618     // Instantiation and passing `true` enables exceptions
619     $mail = new PHPMailer(true);
620
621     try {
622         //Enable SMTP debugging.
623         $mail->SMTPDebug = 0;
624         //Set PHPMailer to use SMTP.
625         $mail->isSMTP();
626
627         //Set SMTP host name
628         $mail->Host = MAIL_HOST;
629
630         //Set this to true if SMTP host requires authentication to send email
631         $mail->SMTPAuth = true;
632
633         //Provide username and password
634         $mail->Username = MAIL_USERNAME;
635         $mail->Password = MAIL_PASSWORD;
636
637         //If SMTP requires TLS encryption then set it
638         $mail->SMTPSecure = "tls";
639
640         //Set TCP port to connect to
641         $mail->Port = MAIL_PORT;
642         $mail->From = MAIL_USERNAME;
643         $mail->FromName = "WEGO Service";
644     }
```

```
645     $mail->addAddress($recipientMail);
646
647     $mail->isHTML(true);
648
649     $mail->Subject = $subject;
650     $mail->Body = $message;
651
652     if ($mail->send())
653         return true;
654     else
655         return false;
656 } catch (Exception $e) {
657     return false;
658 }
659 }
660
661 /**
662  * @author Hoarau Nicolas
663  * @date 03.06.2020
664  * @brief Récupère tous les événements publics pas encore passé
665  * @return array|null
666  * @version 1.0
667  */
668 function GetAllFutureEvent(): ?array
669 {
670     $dateMoreOneHour = date('Y-m-d H:i:s', strtotime(date('Y-m-d H:i:s') . '+1hours'));
671
672     $query = <<<EX
673     SELECT idEvenement, nom, dateDebut
674     FROM evenement
675     WHERE TIMESTAMP(dateDebut) >= :dateTime
676     ORDER BY dateDebut DESC
677     EX;
678
679     try {
680         $requestGetAllFutureEvent = DatabaseController::prepare($query);
681         $requestGetAllFutureEvent->bindParam(':dateTime', $dateMoreOneHour);
682         $requestGetAllFutureEvent->execute();
683         $result = $requestGetAllFutureEvent->fetchAll(PDO::FETCH_ASSOC);
684         return $result;
685     } catch (PDOException $e) {
686         return null;
687     }
688 }
689
690 /**
691  * @author Hoarau Nicolas
692  * @date 03.06.2020
693  * @brief Fonction qui récupère le nombre de participant à un événement
694  * @param integer $idEvent
695  * @return integer|null
696  * @version 1.0
697  */
698 function GetEventNbGuest(int $idEvent): ?int
699 {
```

```
700 $query = <<<EX
701 SELECT COUNT(idUser) AS nbGuest
702 FROM inscriptions
703 WHERE idEvenement = :idEvent
704 EX;
705
706 try {
707     $requestGetEventNbGuest = DatabaseController::prepare($query);
708     $requestGetEventNbGuest->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
709     $requestGetEventNbGuest->execute();
710
711     $result = $requestGetEventNbGuest->fetch(PDO::FETCH_ASSOC);
712
713     return $result !== false ? $result['nbGuest'] : 0;
714 } catch (PDOException $e) {
715     return null;
716 }
717 }
718
719 /**
720  * @author Hoarau Nicolas
721  * @date 03.06.2020
722  * @brief Fonction qui récupère les addresses mail des participant à l'événement donr
723  * @param integer $idEvent
724  * @return array|null
725  * @version 1.0
726  */
727 function GetEventUsersMail(int $idEvent): ?array
728 {
729     $query = <<<EX
730     SELECT u.email
731     FROM inscriptions AS i
732     JOIN user AS u on i.idUser = u.idUser
733     WHERE idEvenement = :idEvent;
734     EX;
735
736     try {
737         $requestGetEventUsersMail = DatabaseController::prepare($query);
738         $requestGetEventUsersMail->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
739         $requestGetEventUsersMail->execute();
740
741         $result = $requestGetEventUsersMail->fetchAll(PDO::FETCH_ASSOC);
742
743         return $result;
744     } catch (PDOException $e) {
745         return null;
746     }
747 }
748
749 /**
750  * @author Hoarau Nicolas
751  * @date 04.06.2020
752  * @brief Fonction qui modifie un événement donné
753  * @param string $eventName
754  * @param string $eventDescription
```

```
755 * @param string $place
756 * @param Date $beginningDate
757 * @param Date $endDate
758 * @param integer $nbMaxGuest
759 * @param integer $idUser
760 * @param integer $idEvent
761 * @return boolean
762 * @version 1.0
763 */
764 function EditEvent(string $eventName, string $eventDescription, string $place,
    $beginningDate, $endDate, int $nbMaxGuest, $img = null, int $idEvent): bool
765 {
766     $query = <<<EX
767     UPDATE evenement
768     SET nom = :eventName, descriptif = :eventDescription, lieu = :place, dateDebut =
:beginningDate, dateFin = :endDate, nbMaxParticipant = :nbMaxGuest
769     EX;
770
771     if ($img != null)
772         $query .= ", urlImage = :urlImg";
773
774     $query .= " WHERE idEvenement = :idEvent";
775
776     try {
777         DatabaseController::beginTransaction();
778         $requestEditEvent = DatabaseController::prepare($query);
779         $requestEditEvent->bindParam(':eventName', $eventName, PDO::PARAM_STR);
780         $requestEditEvent->bindParam(':eventDescription', $eventDescription,
PDO::PARAM_STR);
781         $requestEditEvent->bindParam(':place', $place, PDO::PARAM_STR);
782         $requestEditEvent->bindParam(':beginningDate', $beginningDate);
783         $requestEditEvent->bindParam(':endDate', $endDate);
784         $requestEditEvent->bindParam(':nbMaxGuest', $nbMaxGuest, PDO::PARAM_INT);
785
786         if ($img != null)
787             $requestEditEvent->bindParam(':urlImg', $img, PDO::PARAM_STR, 255);
788
789         $requestEditEvent->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
790         $requestEditEvent->execute();
791         DatabaseController::commit();
792         return true;
793     } catch (PDOException $e) {
794         DatabaseController::rollBack();
795         return false;
796     }
797 }
798
799 /**
800 * @author Hoarau Nicolas
801 * @date 04.06.2020
802 * @brief Fonction qui récupère l'id des événements d'un utilisateur
803 * @param integer $idUser
804 * @return array|int|null
805 * @version 1.0
806 */
```

```
807 function GetUsersEvent(int $idUser)
808 {
809     $query = <<<EX
810     SELECT idEvenement FROM evenement WHERE idOrganisateur = :idUser;
811     EX;
812
813     try {
814         $requestGetUsersEvents = DatabaseController::prepare($query);
815         $requestGetUsersEvents->bindParam(':idUser', $idUser, PDO::PARAM_INT);
816         $requestGetUsersEvents->execute();
817
818         $result = $requestGetUsersEvents->fetchAll(PDO::FETCH_ASSOC);
819         return $result == false ? 0 : $result;
820     } catch (PDOException $e) {
821         return null;
822     }
823 }
824
825 /**
826  * @author Hoarau Nicolas
827  * @date 05.06.2020
828  * @brief Fonction qui compte le nombre de participant aux événement de l'utilisateur
829  * @param array $usersEvent
830  * @return integer
831  * @version 1.0
832  */
833 function CountUsersGuest(array $usersEvent): int
834 {
835     $result = 0;
836
837     foreach ($usersEvent as $event) {
838         $result += GetEventNbGuest($event['idEvenement']);
839
840         if ($result > 0)
841             break;
842     }
843
844     return $result;
845 }
846
847 /**
848  * @author Hoarau Nicolas
849  * @date 05.06.2020
850  * @brief Fonction qui récupère les événements auxquels un utilisateur participe
851  * @param integer $idUser
852  * @return array|null
853  * @version 1.0
854  */
855 function GetEventsParticipating(int $idUser): ?array
856 {
857     $query = <<<EX
858     SELECT idEvenement FROM inscriptions WHERE idUser = :idUser;
859     EX;
860
861     try {
```



```
862     $requestGetEventsParticipating = DatabaseController::prepare($query);
863     $requestGetEventsParticipating->bindParam(':idUser', $idUser, PDO::PARAM_INT);
864     $requestGetEventsParticipating->execute();
865
866     $result = $requestGetEventsParticipating->fetchAll(PDO::FETCH_ASSOC);
867
868     return $result;
869 } catch (PDOException $e) {
870     return null;
871 }
872 }
873
874 /**
875  * @author Hoarau Nicolas
876  * @date 05.06.2020
877  * @brief Fonction qui supprime la participation d'un utilisateur à un événement
878  * @param integer $idUser
879  * @param integer $idEvent
880  * @return boolean
881  * @version 1.0
882  */
883 function DeleteParticipation(int $idUser, int $idEvent): bool
884 {
885     $query = <<<EX
886     DELETE FROM inscriptions WHERE idEvenement = :idEvent AND idUser = :idUser;
887     EX;
888
889     try {
890         DatabaseController::beginTransaction();
891
892         $requestDeleteParticipation = DatabaseController::prepare($query);
893         $requestDeleteParticipation->bindParam(':idUser', $idUser, PDO::PARAM_INT);
894         $requestDeleteParticipation->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
895         $requestDeleteParticipation->execute();
896
897         DatabaseController::commit();
898
899         return true;
900     } catch (PDOException $e) {
901         DatabaseController::rollBack();
902         return false;
903     }
904 }
905
906 /**
907  * @author Hoarau Nicolas
908  * @date 05.06.2020
909  * @brief Fonction qui supprime l'invitation d'un utilisateur à un événement
910  * @param integer $idUser
911  * @param integer $idEvent
912  * @return boolean
913  * @version 1.0
914  */
915 function DeleteInvitation(int $idUser, int $idEvent): bool
916 {
```

```
917 $query = <<<EX
918 DELETE FROM invites WHERE idEvenement = :idUser AND idUser = :idUser;
919 EX;
920
921 try {
922     DatabaseController::beginTransaction();
923
924     $requestDeleteDeleteInvitation = DatabaseController::prepare($query);
925     $requestDeleteDeleteInvitation->bindParam(':idUser', $idUser, PDO::PARAM_INT);
926     $requestDeleteDeleteInvitation->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
927     $requestDeleteDeleteInvitation->execute();
928
929     DatabaseController::commit();
930
931     return true;
932 } catch (PDOException $e) {
933     DatabaseController::rollBack();
934     return false;
935 }
936 }
937
938 /**
939  * @author Hoarau Nicolas
940  * @date 05.06.2020
941  * @brief Fonction qui récupère les événements auxquels un utilisateur est invités
942  * @param integer $idUser
943  * @return array|null
944  * @version 1.0
945  */
946 function GetEventsInvited(int $idUser): ?array
947 {
948     $query = <<<EX
949     SELECT idEvenement FROM invites WHERE idUser = :idUser;
950     EX;
951
952     try {
953         $requestGetEventsParticipating = DatabaseController::prepare($query);
954         $requestGetEventsParticipating->bindParam(':idUser', $idUser, PDO::PARAM_INT);
955         $requestGetEventsParticipating->execute();
956
957         $result = $requestGetEventsParticipating->fetchAll(PDO::FETCH_ASSOC);
958
959         return $result;
960     } catch (PDOException $e) {
961         return null;
962     }
963 }
964
965 /**
966  * @author Hoarau Nicolas
967  * @date 05.06.2020
968  * @brief Fonction qui modifie un utilisateur donné
969  * @param string $nickname
970  * @param string $firstname
971  * @param string $email
```

```
972 * @param string $password
973 * @param string $lastname
974 * @param string $phoneNumber
975 * @param integer $idUser
976 * @param string $oldEmail
977 * @return boolean
978 * @version 1.0
979 */
980 function EditProfile(string $nickname, string $firstname, string $email, string
    $password = null, string $lastname = null, string $phoneNumber = null, int $idUser,
    string $oldEmail): bool
981 {
982     $salt = "";
983     $userPassword = null;
984
985     $query = <<<EX
986     UPDATE user
987     SET pseudo = :nickname, prenom = :firstname, email = :email
988     EX;
989
990     if ($lastname != null || $lastname != "")
991         $query .= ", nom = :lastname";
992
993     if ($phoneNumber != null || $phoneNumber != "")
994         $query .= ", telephone = :phoneNumber";
995
996     if ($password != null || $password != "") {
997         $salt = GetSalt(['userEmail' => $oldEmail]);
998         $userPassword = hash('sha256', $password . $salt);
999         $query .= ", password = :password";
1000     }
1001
1002     $query .= " WHERE idUser = :idUser;";
1003
1004     try {
1005         DatabaseController::beginTransaction();
1006
1007         $requestEditEvent = DatabaseController::prepare($query);
1008         $requestEditEvent->bindParam(':nickname', $nickname, PDO::PARAM_STR);
1009         $requestEditEvent->bindParam(':firstname', $firstname, PDO::PARAM_STR);
1010         $requestEditEvent->bindParam(':email', $email, PDO::PARAM_STR);
1011
1012         if ($lastname != null)
1013             $requestEditEvent->bindParam(':lastname', $lastname, PDO::PARAM_STR);
1014
1015         if ($phoneNumber != null)
1016             $requestEditEvent->bindParam(':phoneNumber', $phoneNumber, PDO::PARAM_STR);
1017
1018         if ($userPassword != null)
1019             $requestEditEvent->bindParam(':password', $userPassword, PDO::PARAM_STR);
1020
1021         $requestEditEvent->bindParam(':idUser', $idUser, PDO::PARAM_INT);
1022         $requestEditEvent->execute();
1023         DatabaseController::commit();
1024         return true;
    }
```

```
1025 } catch (PDOException $e) {
1026     DatabaseController::rollBack();
1027     return false;
1028 }
1029 }
1030
1031 function GetInvited(int $idEvent): ?array
1032 {
1033     $query = <<<EX
1034     SELECT u.pseudo, i.idUser
1035     FROM invites AS i
1036     JOIN user AS u ON i.idUser = u.idUser
1037     WHERE i.idEvenement = :idEvent
1038     EX;
1039
1040     try {
1041         $requestGetInvited = DatabaseController::prepare($query);
1042         $requestGetInvited->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
1043         $requestGetInvited->execute();
1044         $result = $requestGetInvited->fetchAll(PDO::FETCH_ASSOC);
1045
1046         return $result;
1047     } catch (PDOException $e) {
1048         return null;
1049     }
1050 }
1051
1052 /**
1053  * @author Hoarau Nicolas
1054  * @date 08.06.2020
1055  * @brief Fonction qui supprime un événement données
1056  * @param integer $idEvent
1057  * @return boolean
1058  */
1059 function DeleteEvent(int $idEvent): bool
1060 {
1061     $query = <<<EX
1062     DELETE FROM evenement WHERE idEvenement = :idEvent;
1063     EX;
1064
1065     try {
1066         DatabaseController::beginTransaction();
1067         $requestDeleteEvent = DatabaseController::prepare($query);
1068         $requestDeleteEvent->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
1069         $requestDeleteEvent->execute();
1070         DatabaseController::commit();
1071         return true;
1072     } catch (PDOException $e) {
1073         DatabaseController::rollBack();
1074         return false;
1075     }
1076 }
1077
```

