```php
<?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 vararіable son
nom de colonne

  $clauseField = "";
```

```php
  $clauseValue = "";

  if ($userEmail !== null) {
    $clauseField = "email";
    $clauseValue = $userEmail;
  } elseif ($userNickname) {
    $clauseField = "pseudo";
    $clauseValue = $userNickname;
  }

  $query = <<<EX
    SELECT salt
    FROM user
    WHERE `{$clauseField}` = :clauseValue;
    EX;

  try {
    $requestSalt = DatabaseController::prepare($query);
    $requestSalt->bindParam(':clauseValue', $clauseValue, PDO::PARAM_STR);
    $requestSalt->execute();

    $result = $requestSalt->fetch(PDO::FETCH_ASSOC);

    return $result !== false ? $result['salt'] : null;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 26.05.2020
 * @brief Fonction qui créer un nouvel utilisateur dans la base de données
 * @param string $nickname
 * @param string $lastname
 * @param string $firstname
 * @param string $email
 * @param string $phoneNumber
 * @param string $password
 * @return boolean
 * @version 1.0
 *    Inscription sans salt
 * @version 1.1
 *    Inscription avec salt
 */
function Register(string $nickname, string $firstname, string $email, string $password, string $lastname = null, string $phoneNumber = null): bool
{
  $query = "";
  $salt = hash('sha256', microtime());
  $userPassword = hash('sha256', $password . $salt);

  if ($lastname != null && $phoneNumber != null) {
    $query = <<<EX
      INSERT INTO user (pseudo, prenom, nom, email, password, telephone, salt)
```

```php
      VALUES (:nickname, :firstname, :lastname, :email, :password, :phoneNumber, :salt);
      EX;
  } elseif ($phoneNumber != null) {
    $query = <<<EX
      INSERT INTO user (pseudo, prenom, nom, email, password, telephone, salt)
      VALUES (:nickname, :firstname, :email, :password, :phoneNumber, :Salt);
      EX;
  } elseif ($lastname != null) {
    $query = <<<EX
      INSERT INTO user (pseudo, prenom, nom, email, password, salt)
      VALUES (:nickname, :firstname, :lastname, :email, :password, :salt);
      EX;
  } else {
    $query = <<<EX
      INSERT INTO user (pseudo, prenom, email, password, salt)
      VALUES (:nickname, :firstname, :email, :password, :salt);
      EX;
  }

  try {
    DatabaseController::beginTransaction();

    $requestRegister = DatabaseController::prepare($query);
    $requestRegister->bindParam(':nickname', $nickname, PDO::PARAM_STR, 50);
    $requestRegister->bindParam(':firstname', $firstname, PDO::PARAM_STR, 50);

    if ($lastname != null)
      $requestRegister->bindParam(':lastname', $lastname, PDO::PARAM_STR, 50);

    $requestRegister->bindParam(':email', $email, PDO::PARAM_STR, 100);
    $requestRegister->bindParam(':password', $userPassword, PDO::PARAM_STR, 100);

    if ($phoneNumber != null)
      $requestRegister->bindParam(':phoneNumber', $phoneNumber, PDO::PARAM_STR, 50);

    $requestRegister->bindParam(':salt', $salt, PDO::PARAM_STR, 64);

    $requestRegister->execute();

    DatabaseController::commit();
    return true;
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 26.05.2020
 * @brief Fonction qui connecte l'utilisateur
 * @param array les données de conexion de l'utilisateur
 * @return array|false
 * @version 1.0
```

```php
 *    Inscription sans salt
 * @version 1.1
 *    Inscription avec salt
 */
function Login(array $args)
{
  // initialise à null les colonnes du tableau vide/inexistante
  $args += [
    'userEmail' => null,
    'userNickname' => null,
    'userPwd' => null,
  ];

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

  $loginField = "";
  $authenticator = "";
  $salt = "";

  if ($userEmail !== null) {
    $salt = GetSalt(['userEmail' => $userEmail]);
    $authenticator = $userEmail;
    $loginField = "email";
  } elseif ($userNickname !== null) {
    $salt = GetSalt(['userNickname' => $userNickname]);
    $authenticator = $userNickname;
    $loginField = "pseudo";
  } else {
    return false;
  }

  if ($userPwd == null)
    return false;

  $pwd = hash('sha256', $userPwd . $salt);

  $query = <<<EX
    SELECT idUser, pseudo, prenom, nom, email, telephone
     FROM user
     WHERE `{$loginField}` = :wayToConnectValue
     AND password = :pwd;
    EX;

  try {
    $requestLogin = DatabaseController::prepare($query);
    $requestLogin->bindParam(':wayToConnectValue', $authenticator, PDO::PARAM_STR);
    $requestLogin->bindParam(':pwd', $pwd, PDO::PARAM_STR);
    $requestLogin->execute();

    $result = $requestLogin->fetch(PDO::FETCH_ASSOC);

    return $result !== false > 0 ?  $result : false;
  } catch (PDOException $e) {
    return false;
```

```php
  }
}

/**
 * @author Hoarau Nicolas
 * @date 26.05.2020
 * @brief Fonction qui vérifie si le paramètre donnée(email, nickname) est déjà utilisé ou non
 * @param array $args
 * @return boolean|null
 * @version 1.0
 */
function IsTaken(array $args): ?bool
{
  $args += [
    'userEmail' => null,
    'userNickname' => null,
    'idUser' => null
  ];

  extract($args);

  $authenticator = "";
  $field = "";
  if ($userEmail != null) {
    $authenticator = $userEmail;
    $field = "email";
  } elseif ($userNickname != null) {
    $authenticator = $userNickname;
    $field = "pseudo";
  }

  $queryWithoutIdUser = <<<EX
    SELECT `{$field}`
    FROM user
    WHERE `{$field}` = :authenticator;
    EX;

  $queryWithIdUser = <<<EX
    SELECT `{$field}`
    FROM user
    WHERE `{$field}` = :authenticator
    AND idUser <> :idUser;
    EX;


  $query = ($idUser != null) ? $queryWithIdUser : $queryWithoutIdUser;

  try {
    $requestIsUsed = DatabaseController::prepare($query);
    $requestIsUsed->bindParam(':authenticator', $authenticator, PDO::PARAM_STR);

    if ($idUser != null)
      $requestIsUsed->bindParam(':idUser', $idUser, PDO::PARAM_INT);

```

```php
    $requestIsUsed->execute();
    $result = $requestIsUsed->fetch(PDO::FETCH_ASSOC);

    return $result !== false ? true : false;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 27.05.2020
 * @brief Fonction qui créée un événement
 * @param string $eventName
 * @param string $eventDescription
 * @param string $place
 * @param Date $beginningDate
 * @param Date $endDate
 * @param integer $nbMaxGuest
 * @param FILES $img
 * @param integer $idUser
 * @return boolean
 * @version 1.0
 */
function CreateEvent(string $eventName, string $eventDescription, string $place, $beginningDate, $endDate, int $nbMaxGuest, $img, int $type, int $idUser, array $guestlist = null): bool
{
  $creationDate = date('Y-m-d H:i');
  $beginingDate = date("Y-m-d H:i", strtotime($beginningDate));
  $endDate = date("Y-m-d H:i", strtotime($endDate));

  $query = <<<EX
    INSERT INTO evenement (nom, descriptif, dateDebut, dateFin, dateCreation, lieu, prive, nbMaxParticipant, urlImage, idOrganisateur)
    VALUES (:name, :description, :beginningDate, :endDate, :creationDate, :place, :type, :nbMaxGuest, :urlImg ,:idUser);
    EX;

  try {
    DatabaseController::beginTransaction();

    $requestCreateEvent = DatabaseController::prepare($query);
    $requestCreateEvent->bindParam(':name', $eventName, PDO::PARAM_STR, 45);
    $requestCreateEvent->bindParam(':description', $eventDescription, PDO::PARAM_STR, 255);
    $requestCreateEvent->bindParam(':beginningDate', $beginingDate);
    $requestCreateEvent->bindParam(':endDate', $endDate);
    $requestCreateEvent->bindParam(':creationDate', $creationDate);
    $requestCreateEvent->bindParam(':place', $place, PDO::PARAM_STR, 255);
    $requestCreateEvent->bindParam(':type', $type, PDO::PARAM_INT, 1);
    $requestCreateEvent->bindParam(':nbMaxGuest', $nbMaxGuest, PDO::PARAM_INT, 11);
    $requestCreateEvent->bindParam(':urlImg', $img, PDO::PARAM_STR, 255);
    $requestCreateEvent->bindParam(':idUser', $idUser, PDO::PARAM_INT, 11);

```

```php
    $requestCreateEvent->execute();

    if ($guestlist != null) {
      $lastInsertId = DatabaseController::getInstance()->lastInsertId();

      for ($i = 0; $i < count($guestlist); $i++) {
        if ($idUser != $guestlist[$i]) {
          if (CreateInvite($guestlist[$i], $lastInsertId) == false) {
            DatabaseController::rollBack();
          }
        }
      }

      DatabaseController::commit();
      return true;
    }

    DatabaseController::commit();
    return true;
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 27.05.2020
 * @brief Fonction qui créée une invitation pour un utilisateur à un événement
 * @param integer $idGuest
 * @param integer $idEvent
 * @return boolean
 * @version 1.0
 */
function CreateInvite(int $idGuest, int $idEvent): bool
{
  $query = <<<EX
    INSERT INTO invites (idUser, idEvenement)
    VALUES (:idGuest, :idEvent);
    EX;

  try {
    DatabaseController::beginTransaction();

    $requestCreateInvite = DatabaseController::prepare($query);
    $requestCreateInvite->bindParam(':idGuest', $idGuest, PDO::PARAM_INT, 11);
    $requestCreateInvite->bindParam(':idEvent', $idEvent, PDO::PARAM_INT, 11);
    $requestCreateInvite->execute();

    DatabaseController::commit();

    return true;
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
```

```php
  }
}

/**
 * @author Hoarau Nicolas
 * @date 05.06.2020
 * @brief Fonction qui supprime l'invitation d'un utilisateur à un événement
 * @param integer $idGuest
 * @param integer $idEvent
 * @return boolean
 * @version 1.0
 */
function DeleteInvite(int $idGuest, int $idEvent): bool
{
  $query = <<<EX
  DELETE FROM invites WHERE idUser = :idUser AND idEvenement = :idEvent;
  EX;

  try {
    DatabaseController::beginTransaction();

    $requestDeleteInvite = DatabaseController::prepare($query);
    $requestDeleteInvite->bindParam(':idGuest', $idGuest, PDO::PARAM_INT, 11);
    $requestDeleteInvite->bindParam(':idEvent', $idEvent, PDO::PARAM_INT, 11);
    $requestDeleteInvite->execute();

    DatabaseController::commit();

    return true;
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 02.06.2020
 * @brief Fonction qui vérifie si l'événement existe
 * @param integer $idEvent
 * @return boolean
 * @version 1.0
 */
function EventExist(int $idEvent): bool
{
  $query = <<<EX
  SELECT COUNT(idEvenement) AS exist
  FROM evenement
  WHERE idEvenement = :idEvent
  EX;

  try {
    $requestExist = DatabaseController::prepare($query);
    $requestExist->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestExist->execute();
```

```php

    $result = $requestExist->fetch(PDO::FETCH_ASSOC);

    return $result['exist'] == 0 ? false : true;
  } catch (PDOException $e) {
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 02.06.2020
 * @brief Fonction qui vérifie si l'utilisateur participaê à l'événement
 * @param integer $idEvent
 * @param integer $idUser
 * @return boolean
 * @version 1.0
 */
function IsParticipating(int $idEvent, int $idUser): bool
{
  $query = <<<EX
  SELECT COUNT(*) AS participating
  FROM inscriptions
  WHERE idUser = :idUser
  AND idEvenement = :idEvent
  EX;

  try {
    $requestIsParticipating = DatabaseController::prepare($query);
    $requestIsParticipating->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestIsParticipating->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestIsParticipating->execute();

    $result = $requestIsParticipating->fetch(PDO::FETCH_ASSOC);

    return $result['participating'] == 0 ? false : true;
  } catch (PDOException $e) {
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 02.06.2020
 * @brief Fonction qui vérifie si l'événement est privé
 * @param integer $idEvent
 * @return boolean
 * @version 1.0
 */
function IsPrivate(int $idEvent): bool
{
  $query = <<<EX
  SELECT prive
  FROM evenement
  WHERE idEvenement = :idEvent;
```

```php
  EX;

  try {
    $requestIsPrivate = DatabaseController::prepare($query);
    $requestIsPrivate->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestIsPrivate->execute();

    $result = $requestIsPrivate->fetch(PDO::FETCH_ASSOC);

    return $result['prive'] == 0 ? false : true;
  } catch (PDOException $e) {
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 02.06.2020
 * @brief Fonction qui vérifie si l'utilisateur est invité à l'événement donnée
 * @param integer $idEvent
 * @param integer $idUser
 * @return boolean
 * @version 1.0
 */
function IsInvited(int $idEvent, int $idUser): bool
{
  $query = <<<EX
  SELECT COUNT(*) AS isInvited
  FROM invites
  WHERE idUser = :idUser
  AND idEvenement = :idEvent;
  EX;

  try {
    $requestIsInvited = DatabaseController::prepare($query);
    $requestIsInvited->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestIsInvited->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestIsInvited->execute();
    $result = $requestIsInvited->fetch(PDO::FETCH_ASSOC);

    return $result['isInvited'] == 1 ? true : false;
  } catch (PDOException $e) {
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 02.06.2020
 * @brief Fonction qui vérifie si l'utilisateur est le créateur de l'événement donnée
 * @param integer $idEvent
 * @param integer $idUser
 * @return boolean
 * @version 1.0
 */
```

```php
function IsCreator(int $idEvent, int $idUser): bool
{
  $query = <<<EX
  SELECT COUNT(*) AS isCreator
  FROM evenement
  WHERE idOrganisateur = :idUser
  AND idEvenement = :idEvent;
  EX;

  try {
    $requestIsCreator = DatabaseController::prepare($query);
    $requestIsCreator->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestIsCreator->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestIsCreator->execute();

    $result = $requestIsCreator->fetch(PDO::FETCH_ASSOC);

    return $result['isCreator'] == 1 ? true : false;
  } catch (PDOException $e) {
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 02.06.2020
 * @brief Fonction qui récupère les information de l'événement
 * @param integer $idEvent
 * @param integer $idUser
 * @return array|false
 * @version 1.0
 */
function GetEventData(int $idEvent, int $idUser = null)
{
  $queryLogged = <<<EX
  SELECT e.idEvenement, e.nom, e.descriptif, e.dateDebut, e.dateFin, e.lieu,
e.nbMaxParticipant, e.prive, e.urlImage, u.pseudo, (SELECT COUNT(*) FROM inscriptions
WHERE idUser = :idUser AND idEvenement = :idEvent) AS participating, (SELECT
COUNT(idUser) FROM inscriptions WHERE idevenement = :idEvent) AS nbGuest
  FROM evenement AS e
  JOIN user AS u ON e.idOrganisateur = u.idUser
  WHERE e.idEvenement = :idEvent;
  EX;

  $queryNotLogged = <<<EX
  SELECT e.idEvenement, e.nom, e.descriptif, e.dateDebut, e.dateFin, e.lieu,
e.nbMaxParticipant, e.prive, e.urlImage, u.pseudo, (SELECT COUNT(idUser) FROM
inscriptions WHERE idEvenement = :idEvent) AS nbGuest
  FROM evenement AS e
  JOIN user AS u ON e.idOrganisateur = u.idUser
  WHERE e.idEvenement = :idEvent;
  EX;

  $query = $idUser !== null ? $queryLogged : $queryNotLogged;

```

```php
  try {
    $requestGetEventData = DatabaseController::prepare($query);

    if ($idUser !== null)
      $requestGetEventData->bindParam(':idUser', $idUser, PDO::PARAM_INT);

    $requestGetEventData->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestGetEventData->execute();

    $result = $requestGetEventData->fetch(PDO::FETCH_ASSOC);

    return $result !== false ? $result : null;
  } catch (PDOException $e) {
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 02.06.2020
 * @brief Fonction qui envoie un mail à un destinataire précit
 * @param string $recipientMail
 * @param string $message
 * @return bool
 * @version 1.0
 */
function SendMail(string $recipientMail, string $message, string $subject): bool
{
  // Instantiation and passing `true` enables exceptions
  $mail = new PHPMailer(true);

  try {
    //Enable SMTP debugging.
    $mail->SMTPDebug = 0;
    //Set PHPMailer to use SMTP.
    $mail->isSMTP();

    //Set SMTP host name
    $mail->Host = MAIL_HOST;

    //Set this to true if SMTP host requires authentication to send email
    $mail->SMTPAuth = true;

    //Provide username and password
    $mail->Username = MAIL_USERNAME;
    $mail->Password = MAIL_PASSWORD;

    //If SMTP requires TLS encryption then set it
    $mail->SMTPSecure = "tls";

    //Set TCP port to connect to
    $mail->Port = MAIL_PORT;
    $mail->From = MAIL_USERNAME;
    $mail->FromName = "WEGO Service";

```

```php
    $mail->addAddress($recipientMail);

    $mail->isHTML(true);

    $mail->Subject = $subject;
    $mail->Body =  $message;

    if ($mail->send())
      return true;
    else
      return false;
  } catch (Exception $e) {
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 03.06.2020
 * @brief Récupère tous les événements publiques pas encore passé
 * @return array|null
 * @version 1.0
 */
function GetAllFutureEvent(): ?array
{
  $dateMoreOneHour = date('Y-m-d H:i:s', strtotime(date('Y-m-d H:i:s') . '+1hours'));

  $query = <<<EX
  SELECT idEvenement, nom, dateDebut
  FROM evenement
  WHERE TIMESTAMP(dateDebut) >= :dateTime
  ORDER BY dateDebut DESC
  EX;

  try {
    $requestGetAllFutureEvent = DatabaseController::prepare($query);
    $requestGetAllFutureEvent->bindParam(':dateTime', $dateMoreOneHour);
    $requestGetAllFutureEvent->execute();
    $result = $requestGetAllFutureEvent->fetchAll(PDO::FETCH_ASSOC);
    return $result;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 03.06.2020
 * @brief Fonction qui récupère le nombre de participant à un événement
 * @param integer $idEvent
 * @return integer|null
 * @version 1.0
 */
function GetEventNbGuest(int $idEvent): ?int
{
```

```php
  $query = <<<EX
  SELECT COUNT(idUser) AS nbGuest
  FROM inscriptions
  WHERE idEvenement = :idEvent
  EX;

  try {
    $requestGetEventNbGuest = DatabaseController::prepare($query);
    $requestGetEventNbGuest->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestGetEventNbGuest->execute();

    $result = $requestGetEventNbGuest->fetch(PDO::FETCH_ASSOC);

    return $result !== false ? $result['nbGuest'] : 0;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 03.06.2020
 * @brief Fonction qui récupère les addresses mail des participant à l'événement donr
 * @param integer $idEvent
 * @return array|null
 * @version 1.0
 */
function GetEventUsersMail(int $idEvent): ?array
{
  $query = <<<EX
  SELECT u.email
  FROM inscriptions AS i
  JOIN user AS u on i.idUser = u.idUser
  WHERE idEvenement = :idEvent;
  EX;

  try {
    $requestGetEventUsersMail = DatabaseController::prepare($query);
    $requestGetEventUsersMail->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestGetEventUsersMail->execute();

    $result = $requestGetEventUsersMail->fetchAll(PDO::FETCH_ASSOC);

    return $result;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 04.06.2020
 * @brief Fonction qui modifie un événement donné
 * @param string $eventName
 * @param string $eventDescription
```

```php
 * @param string $place
 * @param Date $beginningDate
 * @param Date $endDate
 * @param integer $nbMaxGuest
 * @param integer $idUser
 * @param integer $idEvent
 * @return boolean
 * @version 1.0
 */
function EditEvent(string $eventName, string $eventDescription, string $place, $beginningDate, $endDate, int $nbMaxGuest, $img = null, int $idEvent): bool
{
  $query = <<<EX
  UPDATE evenement
  SET nom = :eventName, descriptif = :eventDescription, lieu = :place, dateDebut = :beginningDate, dateFin = :endDate, nbMaxParticipant = :nbMaxGuest
  EX;

  if ($img != null)
    $query .= ", urlImage = :urlImg";

  $query .= " WHERE idEvenement = :idEvent";

  try {
    DatabaseController::beginTransaction();
    $requestEditEvent = DatabaseController::prepare($query);
    $requestEditEvent->bindParam(':eventName', $eventName, PDO::PARAM_STR);
    $requestEditEvent->bindParam(':eventDescription', $eventDescription, PDO::PARAM_STR);
    $requestEditEvent->bindParam(':place', $place, PDO::PARAM_STR);
    $requestEditEvent->bindParam(':beginningDate', $beginningDate);
    $requestEditEvent->bindParam(':endDate', $endDate);
    $requestEditEvent->bindParam(':nbMaxGuest', $nbMaxGuest, PDO::PARAM_INT);

    if ($img != null)
      $requestEditEvent->bindParam(':urlImg', $img, PDO::PARAM_STR, 255);

    $requestEditEvent->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestEditEvent->execute();
    DatabaseController::commit();
    return true;
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 04.06.2020
 * @brief Fonction qui récupère l'id des événements d'un utilisateur
 * @param integer $idUser
 * @return array|int|null
 * @version 1.0
 */
```

```php
function GetUsersEvent(int $idUser)
{
  $query = <<<EX
  SELECT idEvenement FROM evenement WHERE idOrganisateur = :idUser;
  EX;

  try {
    $requestGetUsersEvents = DatabaseController::prepare($query);
    $requestGetUsersEvents->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestGetUsersEvents->execute();

    $result = $requestGetUsersEvents->fetchAll(PDO::FETCH_ASSOC);
    return $result == false ? 0 : $result;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 05.06.2020
 * @brief Fonction qui compte le nombre de participant aux événement de l'utilisateur
 * @param array $usersEvent
 * @return integer
 * @version 1.0
 */
function CountUsersGuest(array $usersEvent): int
{
  $result = 0;

  foreach ($usersEvent as $event) {
    $result += GetEventNbGuest($event['idEvenement']);

    if ($result > 0)
      break;
  }

  return $result;
}

/**
 * @author Hoarau Nicolas
 * @date 05.06.2020
 * @brief Fonction qui récupère les événements auxquels un utilisateur participe
 * @param integer $idUser
 * @return array|null
 * @version 1.0
 */
function GetEventsParticipating(int $idUser): ?array
{
  $query = <<<EX
  SELECT idEvenement FROM inscriptions WHERE idUser = :idUser;
  EX;

  try {
```

```php
    $requestGetEventsParticipating = DatabaseController::prepare($query);
    $requestGetEventsParticipating->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestGetEventsParticipating->execute();

    $result = $requestGetEventsParticipating->fetchAll(PDO::FETCH_ASSOC);

    return $result;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 05.06.2020
 * @brief Fonction qui supprime la participation d'un utilisateur à un événement
 * @param integer $idUser
 * @param integer $idEvent
 * @return boolean
 * @version 1.0
 */
function DeleteParticipation(int $idUser, int $idEvent): bool
{
  $query = <<<EX
  DELETE FROM inscriptions WHERE idEvenement = :idEvent AND idUser = :idUser;
  EX;

  try {
    DatabaseController::beginTransaction();

    $requestDeleteParticipation = DatabaseController::prepare($query);
    $requestDeleteParticipation->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestDeleteParticipation->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestDeleteParticipation->execute();

    DatabaseController::commit();

    return true;
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 05.06.2020
 * @brief Fonction qui supprime l'invitation d'un utilisateur à un événement
 * @param integer $idUser
 * @param integer $idEvent
 * @return boolean
 * @version 1.0
 */
function DeleteInvitation(int $idUser, int $idEvent): bool
{
```

```php
  $query = <<<EX
  DELETE FROM invites WHERE idEvenement = :idEvent AND idUser = :idUser;
  EX;

  try {
    DatabaseController::beginTransaction();

    $requestDeleteDeleteInvitation = DatabaseController::prepare($query);
    $requestDeleteDeleteInvitation->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestDeleteDeleteInvitation->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestDeleteDeleteInvitation->execute();

    DatabaseController::commit();

    return true;
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 05.06.2020
 * @brief Fonction qui récupère les événements auxquels un utilisateur est invités
 * @param integer $idUser
 * @return array|null
 * @version 1.0
 */
function GetEventsInvited(int $idUser): ?array
{
  $query = <<<EX
  SELECT idEvenement FROM invites WHERE idUser = :idUser;
  EX;

  try {
    $requestGetEventsParticipating = DatabaseController::prepare($query);
    $requestGetEventsParticipating->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestGetEventsParticipating->execute();

    $result = $requestGetEventsParticipating->fetchAll(PDO::FETCH_ASSOC);

    return $result;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 05.06.2020
 * @brief Fonction qui modifie un utilisateur donné
 * @param string $nickname
 * @param string $firstname
 * @param string $email
```

```
 * @param string $password
 * @param string $lastname
 * @param string $phoneNumber
 * @param integer $idUser
 * @param string $oldEmail
 * @return boolean
 * @version 1.0
 */
function EditProfile(string $nickname, string $firstname, string $email, string $password = null, string $lastname = null, string $phoneNumber = null, int $idUser, string $oldEmail): bool
{
  $salt = "";
  $userPassword = null;

  $query = <<<EX
  UPDATE user
  SET pseudo = :nickname, prenom = :firstname, email = :email
  EX;

  if ($lastname != null || $lastname != "")
    $query .= ", nom = :lastname";

  if ($phoneNumber != null || $phoneNumber != "")
    $query .= ", telephone = :phoneNumber";

  if ($password != null || $password != "") {
    $salt = GetSalt(['userEmail' => $oldEmail]);
    $userPassword = hash('sha256', $password . $salt);
    $query .= ", password = :password";
  }

  $query .= " WHERE idUser = :idUser;";

  try {
    DatabaseController::beginTransaction();

    $requestEditEvent = DatabaseController::prepare($query);
    $requestEditEvent->bindParam(':nickname', $nickname, PDO::PARAM_STR);
    $requestEditEvent->bindParam(':firstname', $firstname, PDO::PARAM_STR);
    $requestEditEvent->bindParam(':email', $email, PDO::PARAM_STR);

    if ($lastname != null)
      $requestEditEvent->bindParam(':lastname', $lastname, PDO::PARAM_STR);

    if ($phoneNumber != null)
      $requestEditEvent->bindParam(':phoneNumber', $phoneNumber, PDO::PARAM_STR);

    if ($userPassword != null)
      $requestEditEvent->bindParam(':password', $userPassword, PDO::PARAM_STR);

    $requestEditEvent->bindParam(':idUser', $idUser, PDO::PARAM_INT);
    $requestEditEvent->execute();
    DatabaseController::commit();
    return true;
```

```php
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
  }
}

function GetInvited(int $idEvent): ?array
{
  $query = <<<EX
  SELECT u.pseudo, i.idUser
  FROM invites AS i
  JOIN user AS u ON i.idUser = u.idUser
  WHERE i.idEvenement = :idEvent
  EX;

  try {
    $requestGetInvited = DatabaseController::prepare($query);
    $requestGetInvited->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestGetInvited->execute();
    $result = $requestGetInvited->fetchAll(PDO::FETCH_ASSOC);

    return $result;
  } catch (PDOException $e) {
    return null;
  }
}

/**
 * @author Hoarau Nicolas
 * @date 08.06.2020
 * @brief Fonction qui supprime un événement données
 * @param integer $idEvent
 * @return boolean
 */
function DeleteEvent(int $idEvent): bool
{
  $query = <<<EX
  DELETE FROM evenement WHERE idEvenement = :idEvent;
  EX;

  try {
    DatabaseController::beginTransaction();
    $requestDeleteEvent = DatabaseController::prepare($query);
    $requestDeleteEvent->bindParam(':idEvent', $idEvent, PDO::PARAM_INT);
    $requestDeleteEvent->execute();
    DatabaseController::commit();
    return true;
  } catch (PDOException $e) {
    DatabaseController::rollBack();
    return false;
  }
}
```