FACULTY OF ORGANISATIONAL SCIENCE, UNIVERSITY OF BELGRADE

FACULTY OF MANAGEMENT SCIENCE AND INFORMATICS, UNIVERSITY OF ŽILINA

FACULTY OF ORGANIZATION AND INFORMATICS, UNIVERSITY OF ZAGREB

Ivana Crnov
Karla Hlebec
Andreas Leja
Lukáš Panuška

Application for managing user accounts

ERASMUS+ TEACH4EDU4 PROJECT

Joint Creative Clasroom, 2023.

FACULTY OF ORGANISATIONAL SCIENCE, UNIVERSITY OF BELGRADE

FACULTY OF MANAGEMENT SCIENCE AND INFORMATICS, UNIVERSITY OF ŽILINA

FACULTY OF ORGANIZATION AND INFORMATICS, UNIVERSITY OF ZAGREB

Ivana Crnov, Faculty of organization and informatics

Karla Hlebec, Faculty of organization and informatics

Andreas Leja, Faculty of organization and informatics

Lukáš Panuška, Faculty of management science and informatics

APPLICATION FOR MANAGING USER ACCOUNTS

ERASMUS+ TEACH4EDU4 PROJECT

Mentors:

Assoc. Prof. Markus Schatten, Ph.D.
Srđa Bjeladinović, Ph.D.
doc. Ing. Michal Kvet Ph.D.

Joint Creative Clasroom, 2023.

Content

1. Description of the application domain 1
2. ERA model 3
   2.1. Tables and connections 5
3. Database 9
4. Implementation 10
   4.1. Database implementation 10
   4.2. Application implementation 12
      4.2.1. Administrator point of view 13
      4.2.2. User point of view 16
5. Conclusion 18
6. References 19

Figure table 20

1. Description of the application domain

This application is mainly for managing user accounts, which means that there will be two main roles of application – admin and user. Admin is the role responsible for managing the role of user. Because there will be a table “Role” in database ERA model and database itself, it is important to highlight that in this context *role* is used only to explain main personas who can use the application.

In the process of making the project, several technologies were used. Firstly, for ERA model we used *Toad Data Modeler*. After that, for the database we used *PostgreSQL* database system. Considering the application topic, to make data creation easier, we made data generator in programming language *Python*. The project also includes making web graphical interface which we developed by using *php*, *css* and *javascript* programming languages.

When planning the making of this kind of application, it was first supposed to be decided for what the application would be used for. It is not the same if we are managing users in hospitals or in banks. We decided to make a Blog because we wanted to show how simple the application could look from the outside (user perspective) while from the inside (developer perspective) it is more complicated than many often think.

After defining the main function of our application, it was time to make ERA model. ERA (**E**ntity-**R**elationship-**A**ttribute) model represent a visual aid in showing the relationship between different entities. In making the ERA model of this applications entities are Users, Post, Logs, Account_status, Report_log, Report_code, City, Country, Role, Permission, Role_permission, Users_roles and Users_permission. After making all the entities and filling them with attributes, they need to be connected. For connecting entities, we use different types of relationships. Relationship connects tables where one references to the other via foreign key [2]. Three types of relationship are used in databases [3]:

- One-to-one (1:1)→ with this relationship it is showed that every primary key relates to zero or one record in the table with which it is connected
- One-to-many (1:N) → with this relationship it is showed that only one record relates to zero, one or many records in the table with which it is connected
- Many-to-many (N:M) → with this relationship it is showed that one, zero or many record relate to zero, one or many records in the table with which it is connected

In the relationship, many-to-many it is necessary to have third table (known as associate or linking table) because relational database cannot directly accommodate this kind of relationship. In our model, we are using every relationship except one-to-one relationship.

ERA model will be shown below in the continuation of the work when we will describe the appearance of the model in detail.

Many different types of data can be used in databases and the most common are string, integer, date… Every type can have some kind of constraints. In our application, we used are integer (mostly for generating primary key – it must be unique and also foreign key), varchar with size restrictions, timestamp for date and time, Boolean (data with two possible options – true/false), text for descriptions of codes and bytea for storage of pictures as raw binary strings.

Due to the need for data, a data generator was created in *Python* programming language. It works by importing statements in Python such as csv modules available online (most common blog titles, list of countries by ISO 3166-1 alpha-3 codes[1] list of most popular male and female names, most popular surnames), string module, hashlib module, random module and functions for time and date. Based on the restrictions, the program generated specific number of data which is used to fill the database tables.

2. ERA model

For easier understanding of database, it was needed to first make the model to explain all the data that will be present in the application. It was chosen to make an ERA (Entity-Relationship-Attribute) model.

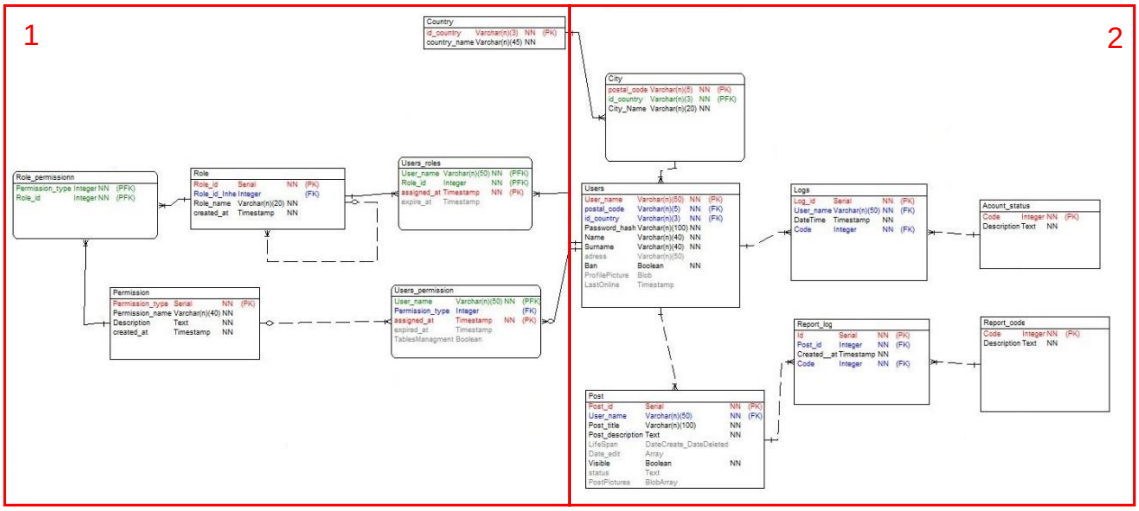


Figure 1. ERA model

On the picture (Figure 1.), is the whole ERA model of this application, but on pictures below, the model is separated in two parts for better visibility.

We have 13 tables that are connected among each other via relationship. The connection between the tables and the tables themselves will be explained in more detail below.

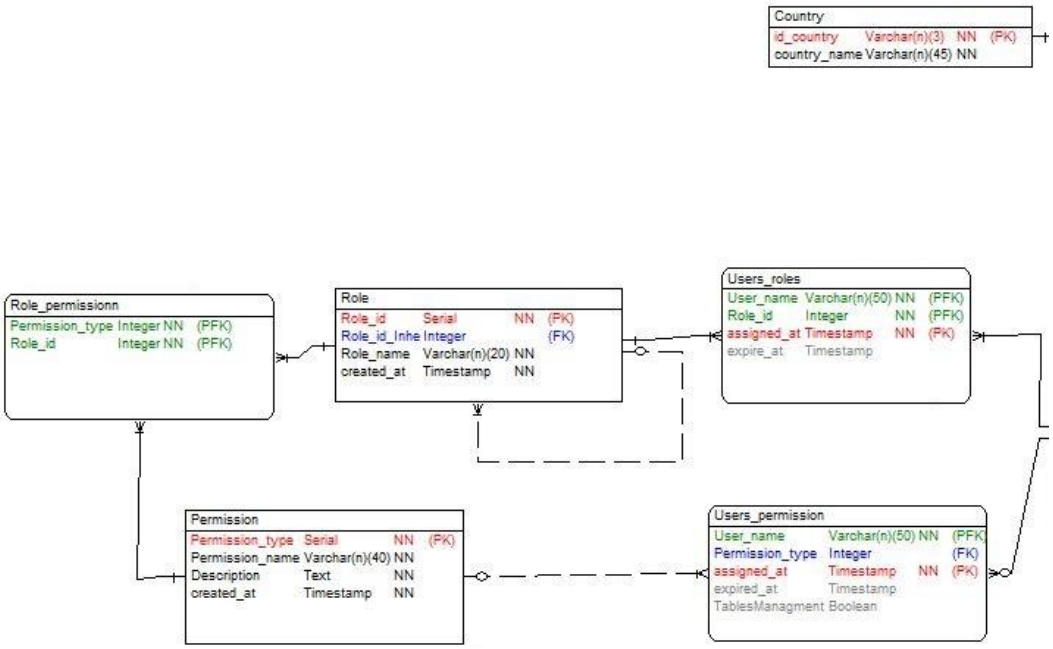


Figure 2. ERA model pt. 1 (tag 1 on Figure 1.)

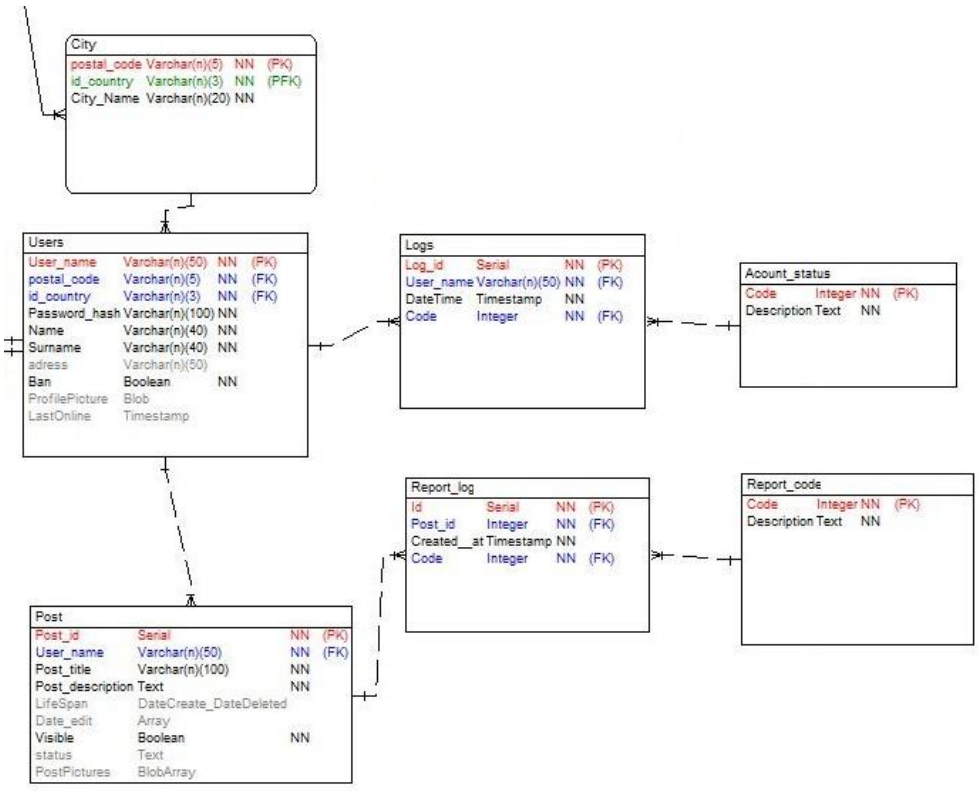


Figure 3. ERA model pt.2 (tag 2 on Figure 1.)

2.1. Tables and connections

- USERS

  Table *Users* is the most important table of the model. It is the center of the application and that is visible by the relations that goes from that table. Data included in the table *Users* are *User_name*, which is also primary code of that table, it is unique and it is mandatory for this table. User also has *postal_code* and *id_country* so it is visible where user from, this data is also mandatory. Of course, every user has *Password_hash* and that means that because of the security reasons user password is hashed with SHA256 function. Aside from *User_name*, every user has to give its *Name* and *Surname*. The information that is not obligatory are *address*, *ProfilePicture* and when was the user *LastOnline*.

  Due to the needs of location (*postal_code* and *id_country*), the table user is connected to the table *City* via connection N:1 which means that one city can be connected with more users, but one user can input only one city.

  When the user is connected to our application, every log that he has is remembered. That is why we also have table *Logs* with which the user is connected via 1:N relation. It means that one user can have more logs, while one log can be connected to only one user.

  Because our application is actually Blog, every user can write *Posts* in it and that gives us one more table. Users and Posts are connected via 1:N relation, so one user can have more blogs, but one blog is connected to only one user as an author.

  Every user has assigned role (table *User_roles*) and can have some kind of permissions (table *User_permission*). Those are the last two tables connected directly to the table *User*. For the roles it is important that one user can have more roles, and one role can be connected to more users (1:N relation). For the permission, there is a case that it is not mandatory that user has some special permission so it is said that one user can have 0 or many permissions and also one permission can be connected to one or more users (1:N relation).

- CITY

  Table *City* is directly connected to table *Users* and to table *Country*. The connection with table *Users* is already explained. The connection between

City and *Country* is 1:N which means that one country can have more cities, but one city can be in only one country. Information that are collected in the table *City* are *postal_code* which is also the primary key of the table, *id_country*, the foreign key and last is the *City_Name*. All of the data from the table *City* is mandatory.

- COUNTRY

  Table *Country* is simple table connected directly only with table *City* with relation as explained before. This table has only mandatory data, one of which is the primary key (*id_country*) and the other is *country_name*. For primary key we used ISO 3166-1 alpha-3 codes. Those are the codes that are defined by the International Organization for Standardization and are used throughout the world [1].

- LOGS

  Because this is the application for managing user accounts, it is necessary to collect data from users, such as *Logs*. This table is connected directly to user, as mentioned before, but it is also connected to another table and that is *Account_status*. Table Logs collect all the logs that user had (date and time when the log was made), and also for admin to not to be confused, it is important to have *User_name* (foreign key for table *Users*) to see which log belongs to which person. Every log has its own *Code* that represents foreign key from table *Account_status* so the meaning of it will be explained in more detail in that table.

- ACCOUNT_STATUS

  Account_status keeps *Code* for all the log status that user can have along with the *Description* of each. It is connected directly to *Logs* table with 1:N connection (explained in Logs). The account can have 10 status:

  - Log in
  - Log out
  - Log in failed
  - Post created
  - Post deleted
  - Post reported
  - Role assigned to the user
  - Role removed from the user
  - Permission assigned to the user
  - Permission removed from the user

- POST

  In the table *Post* it is visible what information should or can a post have so based on that, like every post, it has *Post_title* and *Post description*. Every post can be deleted and edited, as well as hidden or shown so accordingly, we have time variables (*Life span* and *Date_edit*) that remember when any of those changes occurred. For the post, user can add *PostPictures* – a picture that follows the description or the title of the blog. Every post can be reported and then it should be allowed by the admin. Because of that, it is possible for the post to have status which will describe to user what is happening with its blog. Table *Post* is directly connected with table *Users* (through the *User_name*) as already explained before, but also it has direct connection to table *Report_log* which is to be expected given the reporting capability mentioned above. The connection between *Post* and *Report_log* is 1:N which means that one post can have more reports, and one report can be given to more posts.

- REPORT_LOG

  Report_log is not only connected to *Post*, but also to *Report_code* table. This other connection is N:1 which means that one log can have more report codes, and one report code can be assigned to more logs. *Report_log* is connected to *Post* via *Post_id* and to *Report_log* via *Code*. Beside that, the table also contains time when the report was made (*Created_at*).

- REPORT_CODE

  This table is directly connected only to *Report_log* table. It contains the *Code* which is unique and *Description* of the code for every report reason. Every post can be reported and this is important part of every application because it gives users the permission to participate in managing other users posts.

- PERMISSION

  Every user has different permissions. They are determined by the users role, but they can also be assigned to an individual user regardless of the role. Type of the permission is its primary key (*Permission_type*), but it also has the name (*Permission_name*) and the *Description* so it would be more understandable who is permission for and what does it represents. Permissions are not all created at once so if the admin realizes that new permissions should be made he will create them afterwards. That is why every permission has time when it was created (*created_at*) Permissions are connected via table *Role_permission* to the table *Role* with 1:N relation. Also

it is connected to the table *Users* via table *Users_permission* and in this case, the relation is zero-to-many which means that user can have more permissions outside of assigned *Role*.

- ROLE

  The function of the table *Role* is to show what roles can user have. Role has one unary relation and with that we showed the hierarchy present between users. It also shows inheritance (Role_id_Inhe). It means that user has one role and associated attributes, moderator has its role and its attributes along with all the users attributes, and last the admin has its attributes but he inherited all of the attributes from roles below him (user<moderator<admin). To conclude, user with most privileges in the application is admin, while user with least privileges is regular unregistered/unlogged user. Table *Role* is connected to table *Permission* via table *Role_permission* with connection 1:N (one role has more permissions and one permission is given to one role). It is also connected to the table *Users* through connection to intermediate table *Users_roles* via connection 1:N (one user can have multiple roles and one role can be given to multiple users). Roles can be added subsequently so for each role we can see when it is created.

- USERS_ROLES

  This table represents an intermediary between table *Users* and table *Role*. In this table it is visible that role is determined by time because it has beginning (*assigned_at*) and an end (*expire_at*). It is not necessary that the role expires.

- ROLE_PERMISSION

  As it is already mentioned, this table is mediator between tables *Role* and *Role permission*.

- USER_PERMISSION

  This table is connection between *Permission* and *Users*. It contains information relevant for when user has permission outside of its role. It is known which user (User_name) has which permission (Permission_type) and when was that permission given to him (assigned_at). Also the permission can be temporary so it is known when will it expire (expire_at).

3. Database

The made database is generalized-relational. That it is generalized we can support with the facts that we use inheritance (explained in the ERA model for the table *Role*). Also we use arrays and custom data type. Custom data type is made for lifespan of the post. Type is lifespan and it is made from two existing types – date of creation and date of deleting (two timestamps). It is also relational database and it is visible in the tables because every row has its own primary key, tables are connected via primary and foreign keys [4].

Database system used in the project is PostgreSQL. It runs on FOI server so everyone who knows credentials can access that server. It was made based on ERA model explained in the previous chapter. Database is in the 3rd normal form and it is visible by the fact that there are no transitive dependencies. This means that there are no attributes that are not primary keys that are dependent on each other.

All of the data have constrains if we are looking at the type of data (if the data should be numbers, we cannot input text). Some of the data in the database also has size restriction so the user should be careful when entering information, such as *Post_title* not to be bigger than 100 characters. Besides those regular constraints and rules, there are no other for our database.

To protect the data in our database, we are using SHA256 encryption to protect the passwords connected to the users. Not all users have all permissions and they can have different role so to many users access to sensitive data is not allowed. Thanks to the logs, admin can see user activities and that is also part that protects other users and our database. Last thing that we did to protect our data is logging into blog. Because of that not every web user can access our data.

As mentioned before, we have logs that track users activity on the application. There are also information about reports of the users blogs. This allows us to track data needed for security and statistic purposes.

4. Implementation

GitHub repository of the application is available on https://github.com/AILab-FOI/JCC-ADBSTeam2

4.1. Database implementation

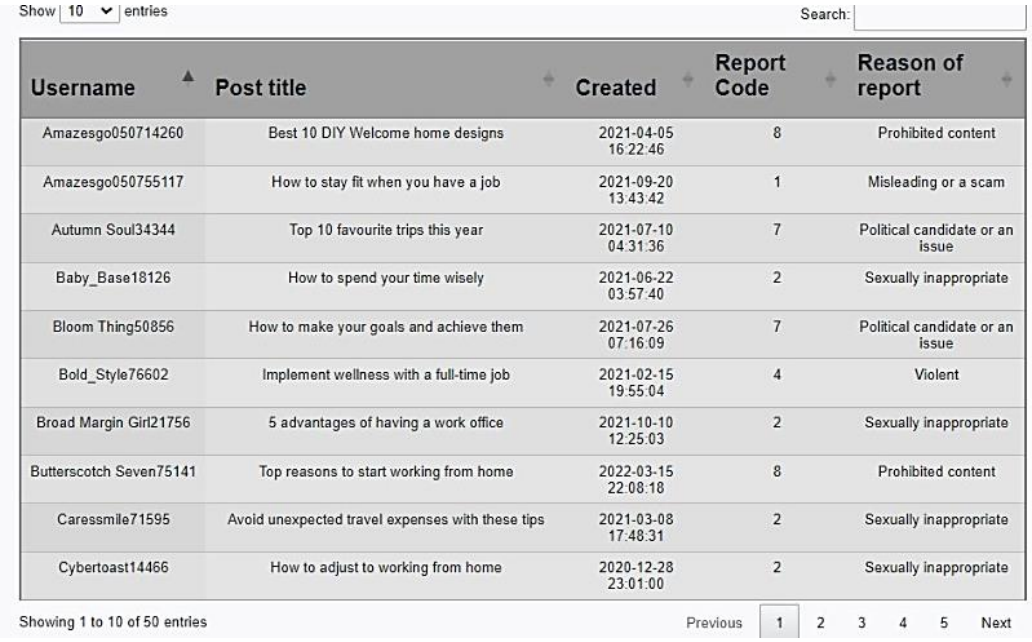
To use the database in our application, we first had to connect it to the server and than connect the server to php so we can use the data from database (in GitHub connect.php). To connect the database we first assigned values to needed variables for connection. Because of security reasons, values in the following code are renamed. After assigning values, we created new empty class and an object of that same class. The values from the beginning are now added to the properties of the created object. After that we use *pgconnect* method to connect a PostgreSQL database to our file.

```php
<?php
$servername = "servername";
$username = "username";
$password = "password";
$dbname = "dbname";

class connectionParams {}
$param = new connectionParams;

$param->host = "servername";
$param->port = 12345;
$param->dbname = "dbname";
$param->user = "username";
$param->password = "password";

$hostString = "";

foreach ($param as $key => $value) {
    $hostString = $hostString . $key . "=" . $value . " ";
}

$conn = pg_connect("host=servername port=12345 dbname=dbname
user=username password=password");
?>
```

The file is imported in every other *.php* file with the following command:

```php
    require('connect.php');
```

The following picture (Figure 4.) is the result of the successfully connected database to the application. It is the table where it is visible that some users have reports. There are report codes and reasons why the posts are reported. Same codes and same reasons are available in the table *Report_code* of our database (Figure 5.).

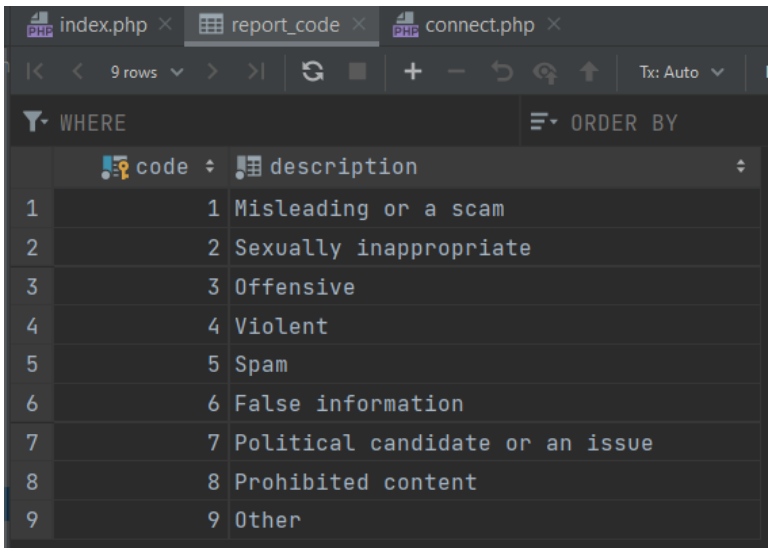


Show 10 entries Search:

Username	Post title	Created	Report Code	Reason of report
Amazesgo050714260	Best 10 DIY Welcome home designs	2021-04-05 16:22:46	8	Prohibited content
Amazesgo050755117	How to stay fit when you have a job	2021-09-20 13:43:42	1	Misleading or a scam
Autumn Soul34344	Top 10 favourite trips this year	2021-07-10 04:31:36	7	Political candidate or an issue
Baby_Base18126	How to spend your time wisely	2021-06-22 03:57:40	2	Sexually inappropriate
Bloom Thing50856	How to make your goals and achieve them	2021-07-26 07:16:09	7	Political candidate or an issue
Bold_Style76602	Implement wellness with a full-time job	2021-02-15 19:55:04	4	Violent
Broad Margin Girl21756	5 advantages of having a work office	2021-10-10 12:25:03	2	Sexually inappropriate
Butterscotch Seven75141	Top reasons to start working from home	2022-03-15 22:08:18	8	Prohibited content
Caressmile71595	Avoid unexpected travel expenses with these tips	2021-03-08 17:48:31	2	Sexually inappropriate
Cybertoast14466	How to adjust to working from home	2020-12-28 23:01:00	2	Sexually inappropriate

Showing 1 to 10 of 50 entries Previous 1 2 3 4 5 Next

Figure 4. Application for managing users - Reports



index.php | report_code | connect.php

9 rows Tx: Auto

WHERE ORDER BY

	code	description
1	1	Misleading or a scam
2	2	Sexually inappropriate
3	3	Offensive
4	4	Violent
5	5	Spam
6	6	False information
7	7	Political candidate or an issue
8	8	Prohibited content
9	9	Other

Figure 5. Application for managing users - Database table Reports

4.2. Application implementation

This application for managing users we can split in two parts:

1. Administrator point of view
2. User point of view

Every part of the application is using the same data from database but the difference is in the permissions and roles. User cannot see some parts, which the administrator can.

The application is made from the *.php* file listed on the picture below (Figure 6.):

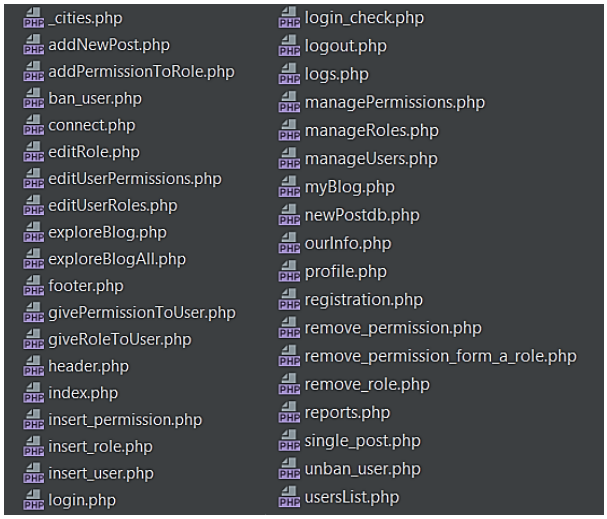


Figure 6. List of .php files

Some php files have associated *.css* file listed in the picture below (Figure 7.):

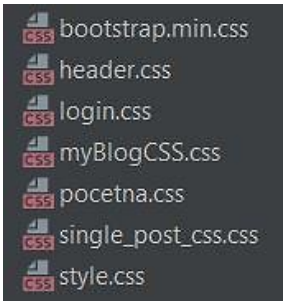


Figure 7. Application .css files

All the links and pages of the application are put in the header:

```
<ul>
        <li><a href="exploreBlogAll.php">Explore all blogs</a></li>
        <li><a href="exploreBlog.php">Explore random blogs</a></li>
        <?php if (isset($_SESSION['username'])){ ?>
            <li><a href="myBlog.php">My blog</a></li>
        <?php } if(isset($_SESSION['username']) && $_SESSION['username']
== "Racciel" || $_SESSION['username'] == "admin") {// this has to be
changed later to include the query data ?>
        <li><a href="manageUsers.php">Manage users</a></li>
        <li><a href="manageRoles.php">Manage roles</a></li>
        <li><a href="managePermissions.php">Manage permissions</a></li>
        <li><a href="usersList.php">Users list</a></li>
        <li><a href="logs.php">Logs</a></li>
        <li><a href="reports.php">Reports</a></li>
            <?php } ?>
        <li><a href="ourInfo.php">Some sort of info</a></li>
    </ul>
```

4.2.1. Administrator point of view

Administrator can access to all the parts of application as it is shown in the picture below (Figure 8.).

Figure 8. Administrator header

Besides that, the main part of his role is to manage users so it is seen in that he has many functions based on that – manage users, manage roles, manage permissions, users list etc.

Considering administrator main job is managing the user, page of that part is shown on picture below (Figure 9.):

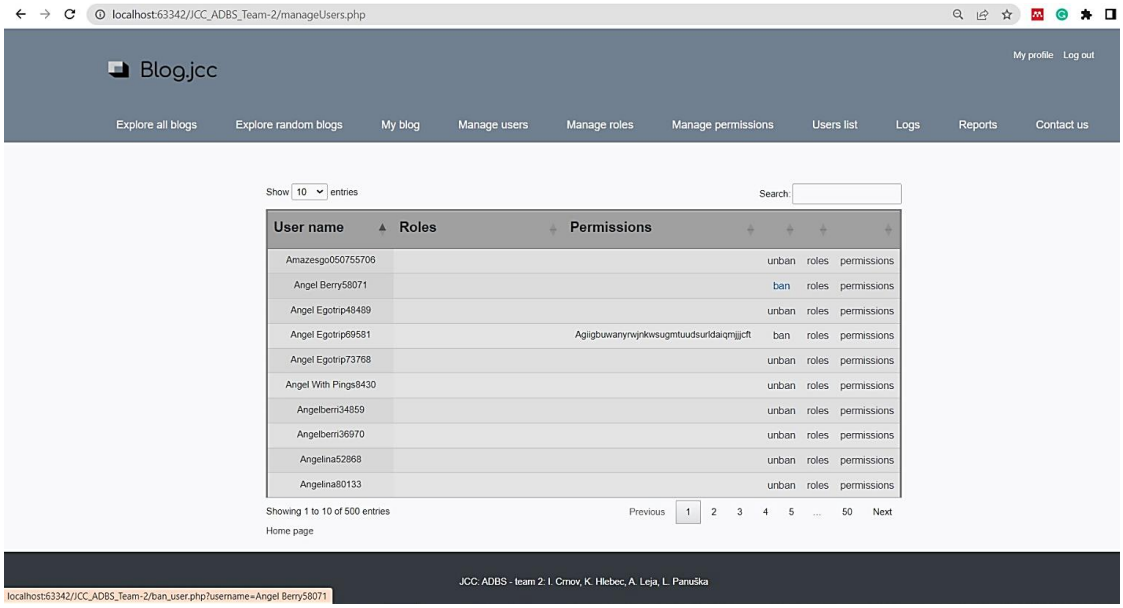


Figure 9. Manage user - ban

Part of the managing is showed in the following code. It is visible that in the part of managing user, admin has view of names of the users, which roles does every of the showed user has and also what permission it has.

```
<?php
$result = pg_query($conn, "SELECT user_name, ban FROM users;");

if($result) {
    echo('<table id="table_id" class="table josjedna display">');
    echo('<thead>
            <tr>
                <th scope="col"><label>User name</label></th>
                <th scope="col"><label>Roles</label></th>
                <th scope="col"><label>Permissions</label></th>
                <th></th><th></th><th></th>
            </tr>
        </thead>');
    echo('<tbody>');
    while ($row = pg_fetch_row($result)) {
        $user = $row[0];
        echo "<tr>
            <td>$row[0]</td>
            <td>";
        $subresult = pg_query($conn,
        "SELECT Users_roles.*, role.role_name
        FROM Users_roles
        JOIN role
        ON role.role_id = users_roles.role_id
        WHERE Users_roles.user_name = '$user';");
        if($subresult) {
            while($row2 = pg_fetch_row($subresult)) {
                echo $row2[4]." ";
            }
        }
```

```
        echo "</td><td>";

        $subresult = pg_query($conn,
        "SELECT Users_permission.*, Permission.permission_name
        FROM Users_permission
        JOIN Permission
        ON permission.permission type = users permission.permission type
        WHERE Users_permission.user_name = '$user';");
        if($subresult) {
            while($row2 = pg_fetch_row($subresult)) {
                echo $row2[5]." ";
            }
        }

        echo"</td>
            <td>";

        if($row[1] == 'f'){
            echo "<a href='ban_user.php?username=$row[0]'>ban</a>";
        }
        else {
            echo "<a href='unban_user.php?username=$row[0]'>unban</a>";
        }
        echo"</td>
            <td><a
href='editUserRoles.php?username=$row[0]'>roles</a></td>
            <td><a
href='editUserPermissions.php?username=$row[0]'>permissions</a></td>
            </tr>";
    }
    echo('</tbody>');
    echo('</table>');
}
echo('<a href="index.php">Home page</a>');
?>
```

The represented code shows possibility of admin to ban the user. It is visible that database is used because of the queries used in this part of code. The main part of banning the user is actually in the file *ban_users.php* where it is visible that if the ban is set to true it will redirect admin to page that is developed through file *manageUsers.php*.

```
<?php
    require('connect.php');
    $username = $_GET['username'];

    $result = pg_query($conn, "UPDATE Users SET ban = TRUE WHERE
user_name = '$username';");

    if($result) {
        header('Location: manageUsers.php');
    }
    else {
        header('Location: ' . $_SERVER["HTTP_REFERER"] );
        exit;
    }
```

4.2.2. User point of view

User does not have access to everything but only to the parts that are shown on the header in the picture below (Figure 10.).

Figure 10. Users header

Users main function is adding new posts on its blog. On the picture bellow (Figure 11.), it is visible how it looks in the application when user wants to add post.

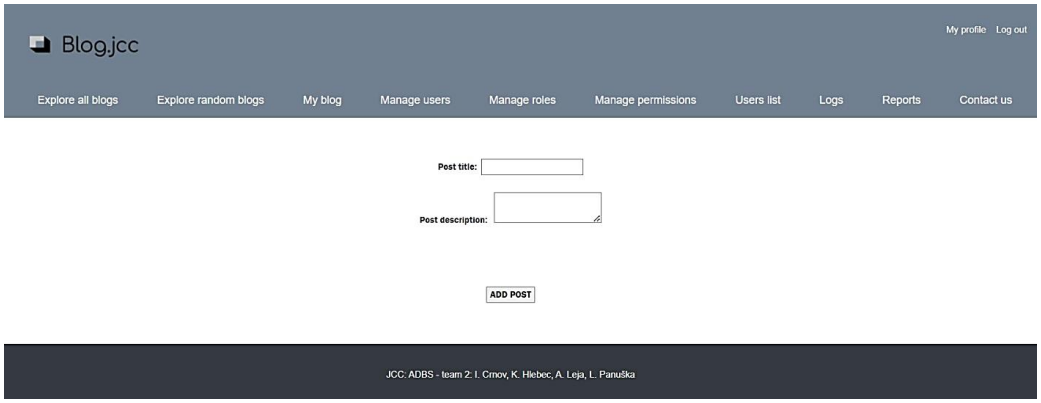


Figure 11. User point of view - adding post

How adding a post looks from the developer point of view is visible in the following part of code:

```
<div id="addNewPost" class="login">
    <form id="addnew" method="POST" action="newPostdb.php.php">
    <br><br>
        <label>Post title: </label> <input type="text"
name="post_title" required><br><br>
        <label>Post description:   </label><textarea
name="description"></textarea><br><br>


        </select><br><br>
        <br><input type="submit" value="ADD POST"><br><br>
    </form>
    <br>
</div>
```

In this code it is also visible that we have action attribute that is connected to newPostdb.php file. So when the user creates a new post, that post will be added to database via connect.php included in the file newPostdb.php.

The code below shows the query from data newPostdb.php that fills the database with data that user inputs. Also it checks if the query is executed properly and if it is it will redirect user to addNewPost.php page. If the query was not successful, user will be redirected to the previous page.

```
<?php
require('connect.php');

$title = $_POST['post_title'];
$description = $_POST['description'];
$username = $ SESSION['username'];
//echo "$name, $surname, $username, $password, $address, $country,
$city";

$result = pg_query($conn, "INSERT INTO post VALUES(DEFAULT,
'$username','$title','$description', (NOW(),NULL ), NULL, 'TRUE', NULL,
NULL)");

if($result)
    header('Location: addNewPost.php');
else {
    header('Location: ' . $_SERVER["HTTP_REFERER"] );
    exit;
}
?>
```

5. Conclusion

Application making has its course of work to make it successful. In this case, it was started by picking the topic of the application. After that, ERA model was made and that significantly helped in making the database. The task was to make the database using PostgreSQL and considering that it is easy to use, database was finished fast. PostgreSQL allowed us to make wanted connection between tables (such as unary relation, which represents inheritance) and it allowed making costume data types (lifespan that is made from two different types). It is a suitable choice for our topic and it was easy to connect it to the .php, which we used for making the application.

Database was quickly filled with data because of the data generator made in Python programming language. It helped generate great amount of data, which is used after in application. All of the data was also available online, which proves how much technology has developed. Python was chosen for the generator because it is easy to use and it is easy to import wanted files with needed information.

As already mentioned the application was made via .php and styled via .css. We used those programming languages because the task was to make web application and that is one of the most popular programming languages for web development. It is supported on operating system in which we as a team worked, and because it is that popular among developer, problems can be solved quickly and easily. As mentioned, .php was connected with .css and which helped a lot when it comes to visual identity of the application.

There were many error codes in the process of developing the application and even though they are all solved, because of different possible solutions it was challenging finding the one, which will work with the used version of the .php.

The main point of application is made and admin can successfully manage the users. Admin has ability to ban user, delete posts from other users and directly change users permission and role. Applications for managing users are complex to make and they require lot of data and knowledge. This application has all the main functions and is available for using.

The main idea of the project is successfully completed and it serves the purpose of managing user and user accounts.

6. References

[1] „ISO - ISO 3166 — Country Codes“. https://www.iso.org/iso-3166-country-codes.html (pristupljeno 27. siječanj 2023.).

[2] „What is a Relationship? - Definition from Techopedia“. https://www.techopedia.com/definition/24438/relationship-databases (pristupljeno 27. siječanj 2023.).

[3] „Database relationships - IBM Documentation“. https://www.ibm.com/docs/en/control-desk/7.6.0?topic=structure-database-relationships (pristupljeno 27. siječanj 2023.).

[4] „What Is a Relational Database | Oracle“. https://www.oracle.com/database/what-is-a-relational-database/ (pristupljeno 27. siječanj 2023.).

[5] LibreTexts, „INT 1010: Concepts in Computing“, str. 35–71, 2004, [Na internetu]. Dostupno na: https://eng.libretexts.org/Courses/Prince_Georges_Community_College/INT_1010%3A_Concepts_in_Computing

[6] M. Johnson, R. Rosebrugh, i R. J. Wood, „Entity-relationship-attribute designs and sketches“, *Theory Appl. Categ.*, sv. 10, izd. 1, str. 94–112, 2002, [Na internetu]. Dostupno na: http://web.science.mq.edu.au/~mike/papers/40.pdf

[7] Q. Li i Y.-L. Chen, „Entity-Relationship Diagram“, u *Modeling and Analysis of Enterprise and Information Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, str. 125–139. doi: 10.1007/978-3-540-89556-5_6.

Figure table

Figure 1. ERA model 3
Figure 2. ERA model pt. 1 (tag 1 on Figure 1.) 4
Figure 3. ERA model pt.2 (tag 2 on Figure 1.) 4
Figure 4. Application for managing users - Reports 11
Figure 5. Application for managing users - Database table Reports 11
Figure 6. List of .php files 12
Figure 7. Application .css files 12
Figure 8. Administrator header 13
Figure 9. Manage user - ban 14
Figure 10. Users header 16
Figure 11. User point of view - adding post 16