UD5. Activity 1 (Christmas)

MULTIMEDIA CONTENT IMPLEMENTATION

Toni Barceló García

IES JOAN RAMIS I RAMIS | DAW

Content Table

Source Github project

If you clone the project, you will have all the activities I have done so far sorted by folders in “activities” root directory.

Source project:

https://github.com/abarcelogarcia/abarcelogarcia.github.io

Root folder for this activity: UD5. Activity 1 (Part 2)

activities/UD5A_1_IndexedDB (Part 2)

github-pages (latest version)

https://abarcelogarcia.github.io/

The website structure

Directory	Files	Concept
root	index.html	main website file.
	index_admin.html	admin web file
	index_profile.html	user profile web file
backups	UsersBK.json	json users backup file
css	avatar_effect.css	css file to apply avatar’s effect without js
	bootstrap_custom.css	css file genered by sass
	bootstrap_custom.css.map	Css map file genered by sass
	bootstrap_custom.scss	sass file code
	bootstrap_custom_dark.css	css file dark theme genered by sass
	bootstrap_custom_dark.css.map	Css dark theme map file genered by sass
	bootstrap_custom_dark.scss	sass dark theme file code
js	common.js	Common JS file
	form_password_validator	functions to reset user password
	form_validator.js	form validation functions
	index_admin.js	functions for admin page
	index_profile.js	functions for user profile
	index.js	functions for home page
fonts	BAHNSCHRIFT.TTF	typography chosen in the guide style.
img	*.png, *.jpg	images directory for the web.
node_modeules	*	Bootstrap sass modules
views	post.html	Post file

Anotations:

The name of the indexedDB database is **blogginDB.**

In the database there are 2 Objectstorage.

- **Users**: stores the users.
- **Login**: stores the user who has logged in.

When creating the database, in the users' storage I create an index for each form field so that in future updates it will be easier to search for a specific field.

The admin page is *index_admin.html*

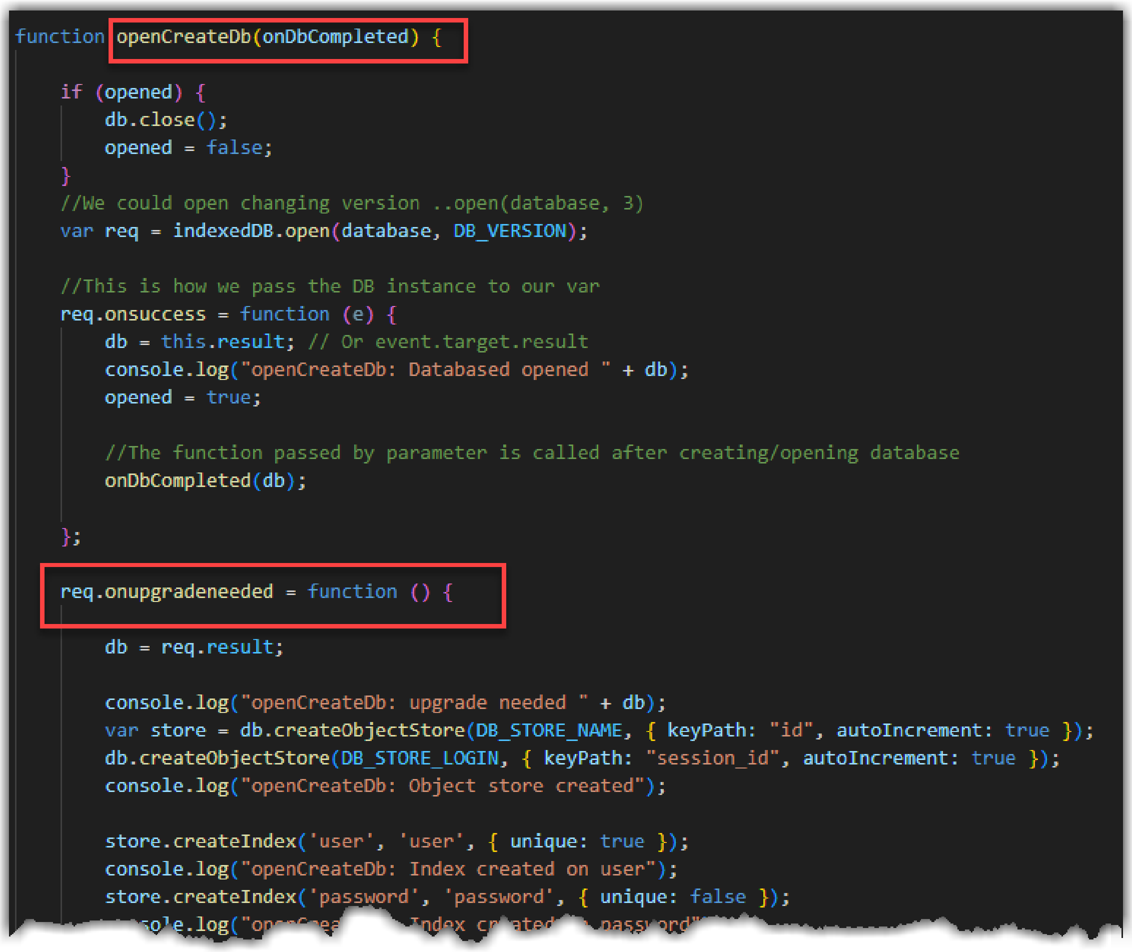
The user profile page is *index_profile.html*

Rubric UD5 Activity 1 – IndexedDB

HOMEPAGE

1. Database is created if it doesn't exist.

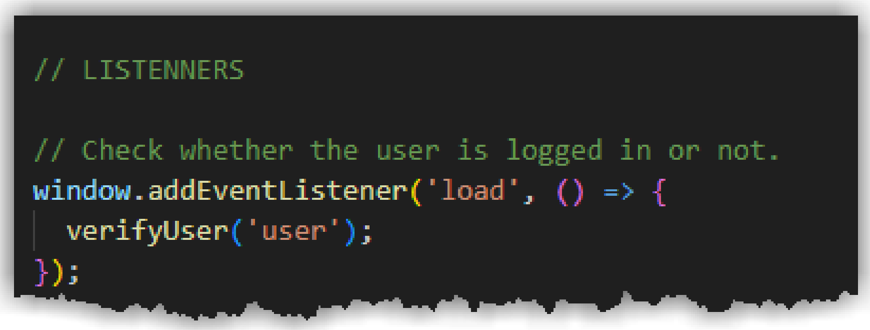
On all pages I check if the user is logged in using a method with a ***onload event***. The first thing is to open and/or create the database.



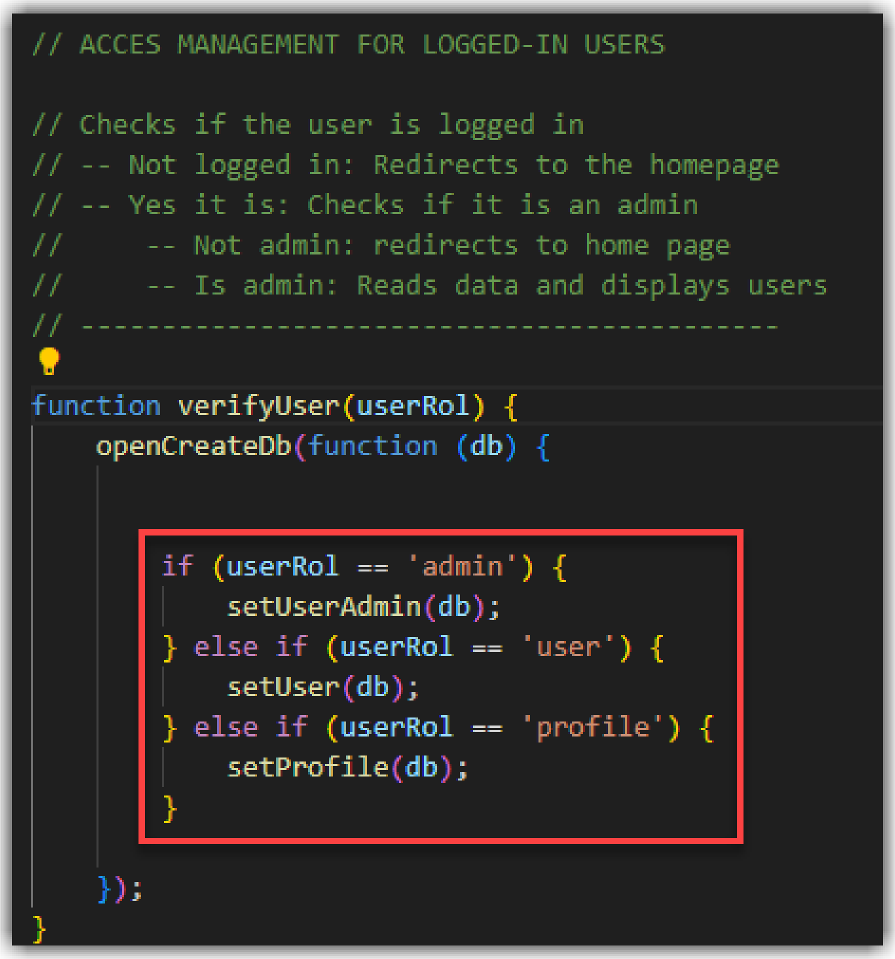
```
function openCreateDb(onDbCompleted) {

    if (opened) {
        db.close();
        opened = false;
    }
    //We could open changing version ..open(database, 3)
    var req = indexedDB.open(database, DB_VERSION);

    //This is how we pass the DB instance to our var
    req.onsuccess = function (e) {
        db = this.result; // Or event.target.result
        console.log("openCreateDb: Databased opened " + db);
        opened = true;

        //The function passed by parameter is called after creating/opening database
        onDbCompleted(db);

    };

    req.onupgradeneeded = function () {

        db = req.result;

        console.log("openCreateDb: upgrade needed " + db);
        var store = db.createObjectStore(DB_STORE_NAME, { keyPath: "id", autoIncrement: true });
        db.createObjectStore(DB_STORE_LOGIN, { keyPath: "session_id", autoIncrement: true });
        console.log("openCreateDb: Object store created");

        store.createIndex('user', 'user', { unique: true });
        console.log("openCreateDb: Index created on user");
        store.createIndex('password', 'password', { unique: false });
```

2. Check if a user is logged in and redirect if the user is admin, show avatar and "Hi, username" message if not.

When the home page loads, after creating or opening the database, it verifies the user and redirects according to the user's role.



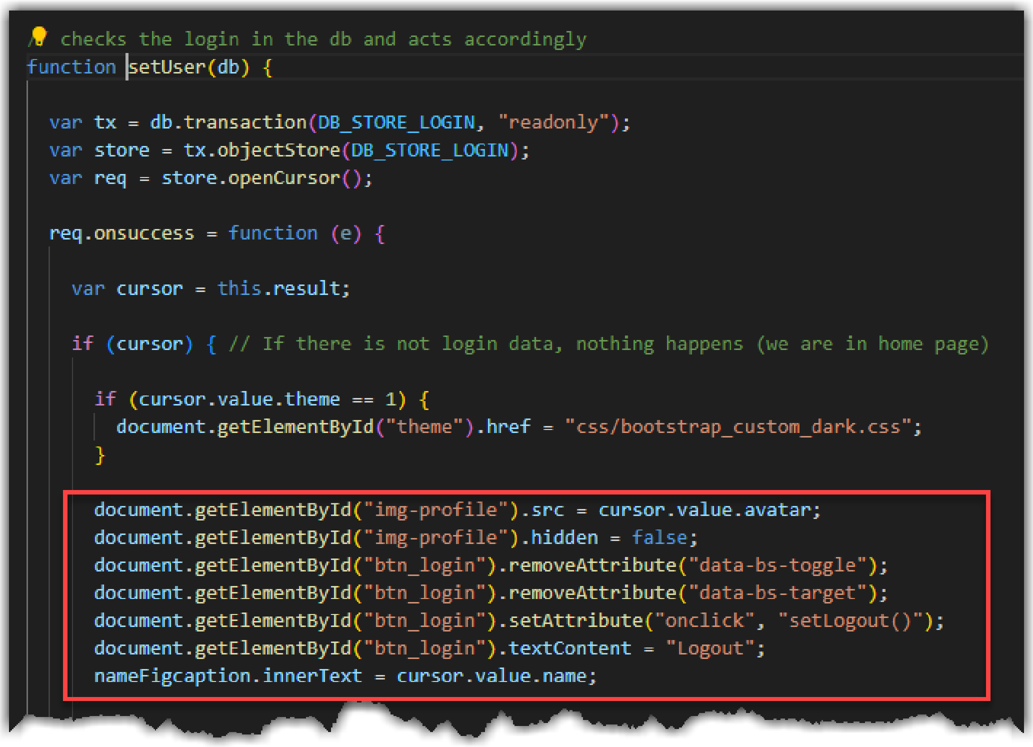
```
// LISTENNERS

// Check whether the user is logged in or not.
window.addEventListener('load', () => {
  verifyUser('user');
});
```



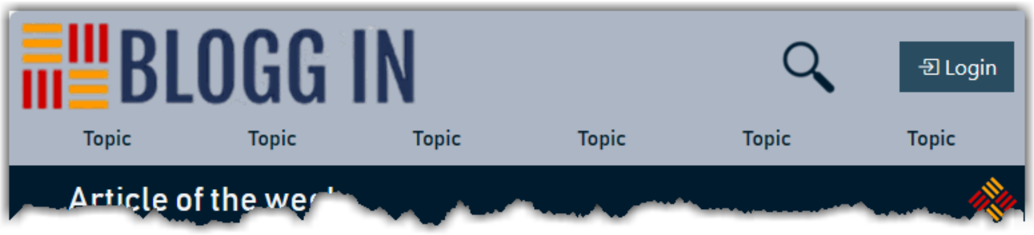
```
// ACCES MANAGEMENT FOR LOGGED-IN USERS

// Checks if the user is logged in
// -- Not logged in: Redirects to the homepage
// -- Yes it is: Checks if it is an admin
//     -- Not admin: redirects to home page
//     -- Is admin: Reads data and displays users
// -------------------------------------------

function verifyUser(userRol) {
    openCreateDb(function (db) {

        if (userRol == 'admin') {
            setUserAdmin(db);
        } else if (userRol == 'user') {
            setUser(db);
        } else if (userRol == 'profile') {
            setProfile(db);
        }

    });
}
```

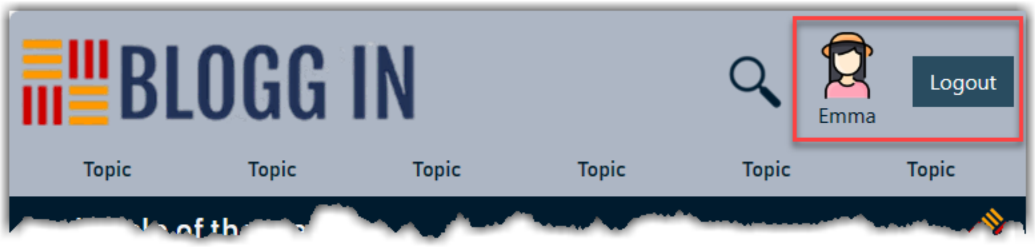
3. If a user is logged in, the registration and login buttons should be hidden and instead there should be a settings button and a logout button (all pages: home, settings, admin). Otherwise, the logout and settings buttons should not be visible.

After logging in or logging in again on the website, it changes the functionality of the login button to logout by assigning a new onclick event for the logout and changes the text and icon of the button. It also adds the avatar and the user's name.



```
// checks the login in the db and acts accordingly
function setUser(db) {

  var tx = db.transaction(DB_STORE_LOGIN, "readonly");
  var store = tx.objectStore(DB_STORE_LOGIN);
  var req = store.openCursor();

  req.onsuccess = function (e) {

    var cursor = this.result;

    if (cursor) { // If there is not login data, nothing happens (we are in home page)

      if (cursor.value.theme == 1) {
        document.getElementById("theme").href = "css/bootstrap_custom_dark.css";
      }

      document.getElementById("img-profile").src = cursor.value.avatar;
      document.getElementById("img-profile").hidden = false;
      document.getElementById("btn_login").removeAttribute("data-bs-toggle");
      document.getElementById("btn_login").removeAttribute("data-bs-target");
      document.getElementById("btn_login").setAttribute("onclick", "setLogout()");
      document.getElementById("btn_login").textContent = "Logout";
      nameFigcaption.innerText = cursor.value.name;
```

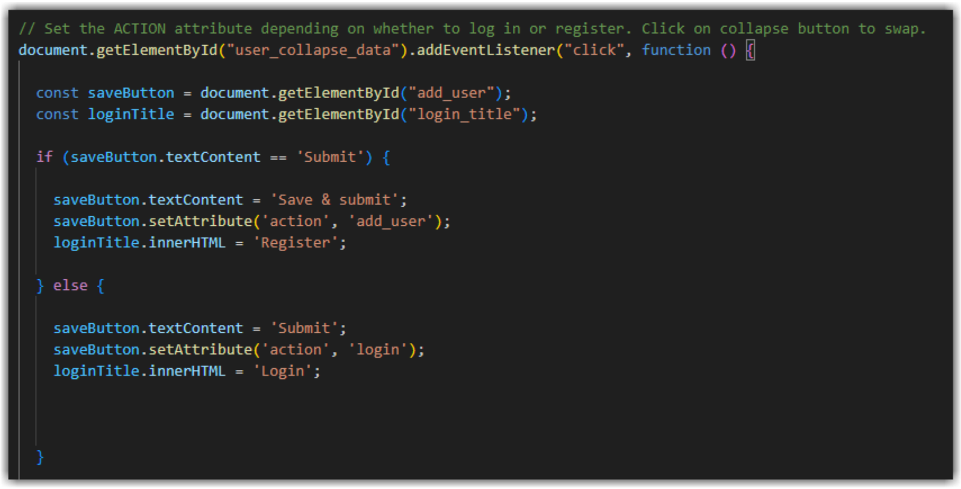




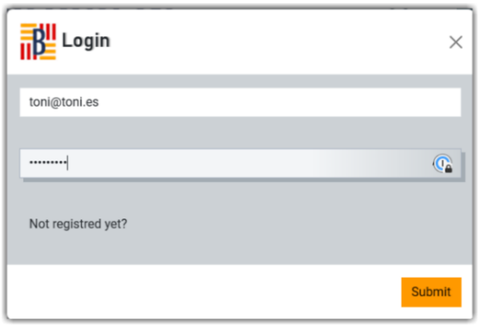
REGISTER

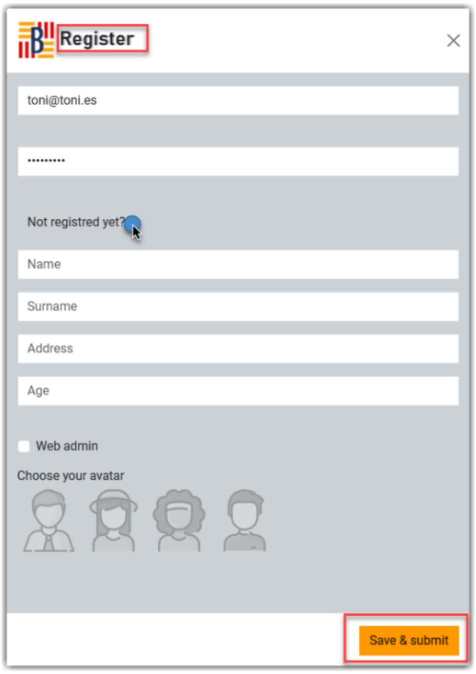
1. There's an option to register a standard or an admin user. When we register those users, the new user is created on the indexedDB. We can register different users.

Registration and login are done via a modal. To differentiate it, when clicking on the dropdown to register, I add an attribute called "action" and depending on its value the modal properties are modified, and I call the method to add user or directly the login method. For the administrator user I have added a checkbox that determines it.



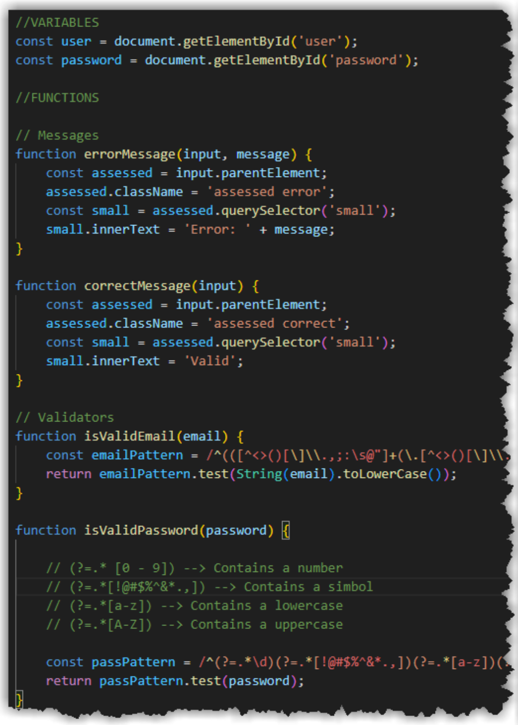
```
// Set the ACTION attribute depending on whether to log in or register. Click on collapse button to swap.
document.getElementById("user_collapse_data").addEventListener("click", function () {

  const saveButton = document.getElementById("add_user");
  const loginTitle = document.getElementById("login_title");

  if (saveButton.textContent == 'Submit') {

    saveButton.textContent = 'Save & submit';
    saveButton.setAttribute('action', 'add_user');
    loginTitle.innerHTML = 'Register';

  } else {

    saveButton.textContent = 'Submit';
    saveButton.setAttribute('action', 'login');
    loginTitle.innerHTML = 'Login';



  }
```



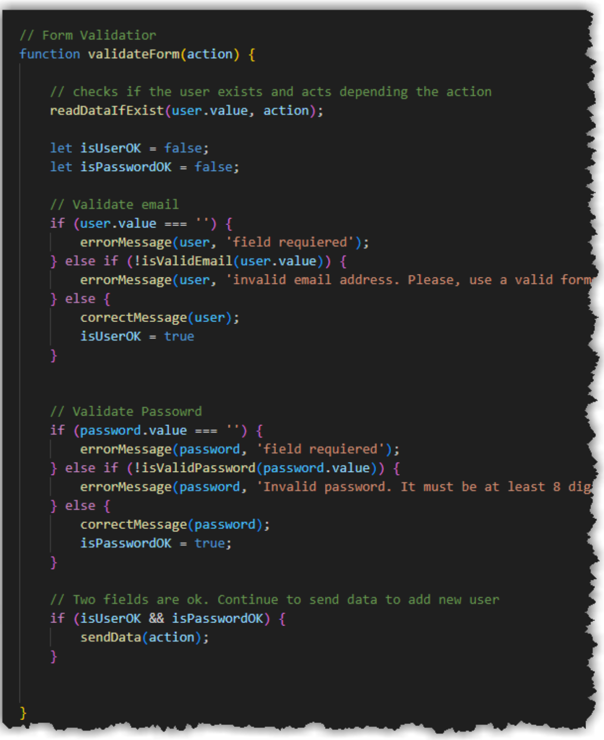


2. Form is validated using JavaScript: no empty inputs are allowed, the email has the correct format, the password follows the requirements (contains at least 8 characters, containing lowercase and uppercase letters, a number and a special character).

I have created two validators with corresponding messages. When clicking on the submit button, the first thing that happens is the validation of the user (email), the password security and if the user exists. In case any of the three checks is not fulfilled, it shows a message under the input that helps the user to do it correctly.

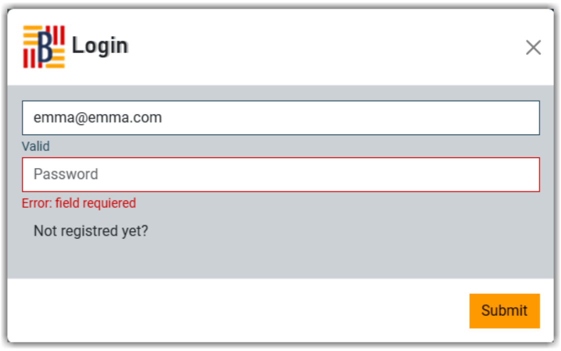


```
//VARIABLES
const user = document.getElementById('user');
const password = document.getElementById('password');

//FUNCTIONS

// Messages
function errorMessage(input, message) {
    const assessed = input.parentElement;
    assessed.className = 'assessed error';
    const small = assessed.querySelector('small');
    small.innerText = 'Error: ' + message;
}

function correctMessage(input) {
    const assessed = input.parentElement;
    assessed.className = 'assessed correct';
    const small = assessed.querySelector('small');
    small.innerText = 'Valid';
}

// Validators
function isValidEmail(email) {
    const emailPattern = /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\
    return emailPattern.test(String(email).toLowerCase());
}

function isValidPassword(password) {

    // (?=.* [0 - 9]) --> Contains a number
    // (?=.*[!@#$%^&*.,]) --> Contains a simbol
    // (?=.*[a-z]) --> Contains a lowercase
    // (?=.*[A-Z]) --> Contains a uppercase

    const passPattern = /^(?=.*\d)(?=.*[!@#$%^&*.,])(?=.*[a-z])(
    return passPattern.test(password);
}
```



```
// Form Validatior
function validateForm(action) {

    // checks if the user exists and acts depending the action
    readDataIfExist(user.value, action);

    let isUserOK = false;
    let isPasswordOK = false;

    // Validate email
    if (user.value === '') {
        errorMessage(user, 'field requiered');
    } else if (!isValidEmail(user.value)) {
        errorMessage(user, 'invalid email address. Please, use a valid form
    } else {
        correctMessage(user);
        isUserOK = true
    }


    // Validate Passowrd
    if (password.value === '') {
        errorMessage(password, 'field requiered');
    } else if (!isValidPassword(password.value)) {
        errorMessage(password, 'Invalid password. It must be at least 8 dig
    } else {
        correctMessage(password);
        isPasswordOK = true;
    }

    // Two fields are ok. Continue to send data to add new user
    if (isUserOK && isPasswordOK) {
        sendData(action);
    }

}
```

Field required

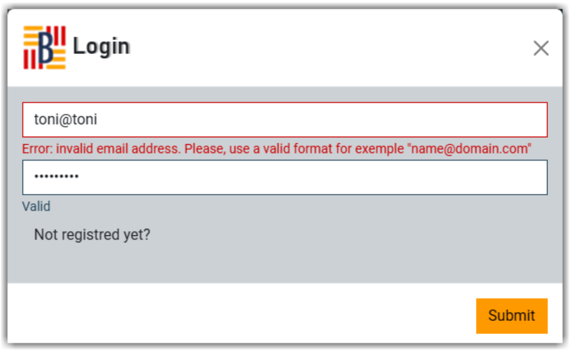


Login

emma@emma.com
Valid
Password
Error: field requiered
Not registred yet?

Submit

EMAIL FORMAT

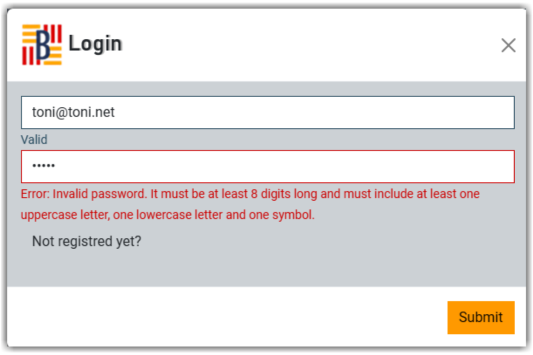


Login

toni@toni
Error: invalid email address. Please, use a valid format for exemple "name@domain.com"
•••••••••
Valid
Not registred yet?

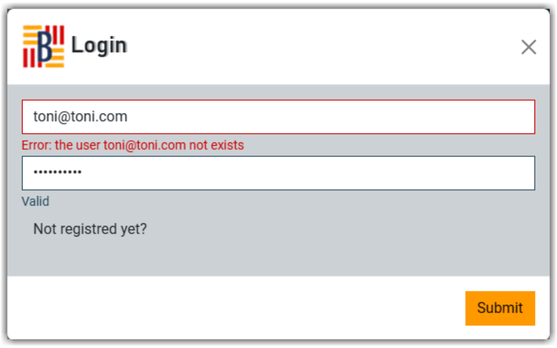
Submit

Password requirement



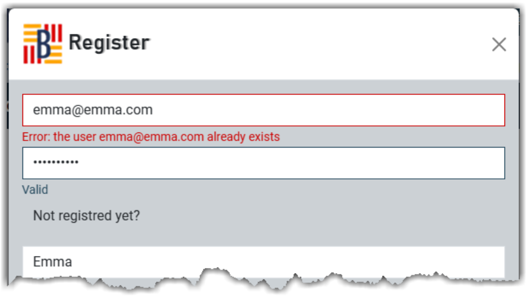
toni@toni.net
Valid
•••••
Error: Invalid password. It must be at least 8 digits long and must include at least one uppercase letter, one lowercase letter and one symbol.
Not registred yet?

User not exists (Login)



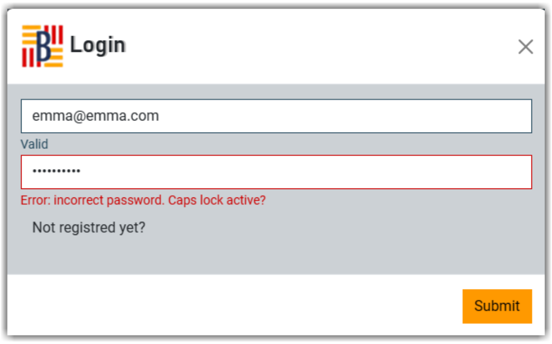
toni@toni.com
Error: the user toni@toni.com not exists
••••••••••
Valid
Not registred yet?

User already exists (Register)



emma@emma.com
Error: the user emma@emma.com already exists
••••••••••
Valid
Not registred yet?
Emma

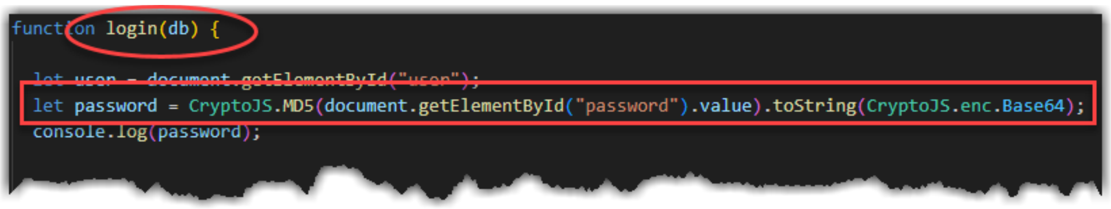
User exists, wrong password (Login)



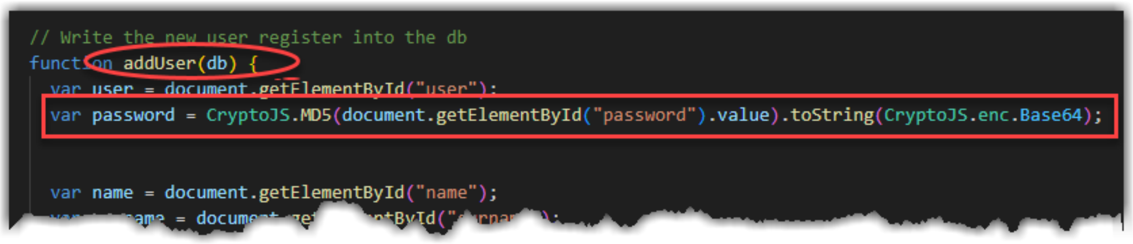
emma@emma.com
Valid
••••••••••
Error: incorrect password. Caps lock active?
Not registred yet?

3. Password is encrypted.

The password is not saved in the database. A hash of the password string is generated in MD5 using CryptoJS. To verify that it is correct, it will be compared.



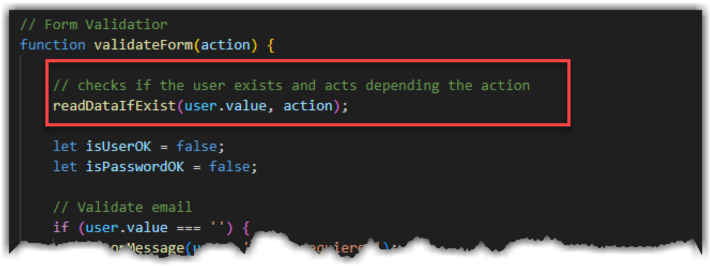
```
function login(db) {

  let user = document.getElementById("user");
  let password = CryptoJS.MD5(document.getElementById("password").value).toString(CryptoJS.enc.Base64);
  console.log(password);
```



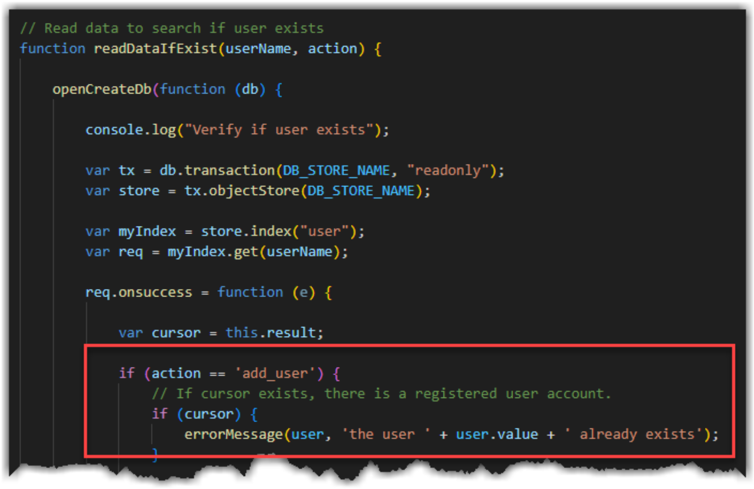
```
// Write the new user register into the db
function addUser(db) {
  var user = document.getElementById("user");
  var password = CryptoJS.MD5(document.getElementById("password").value).toString(CryptoJS.enc.Base64);


  var name = document.getElementById("name");
  var surname = document.getElementById("surname");
```

4. I cannot register with the same email

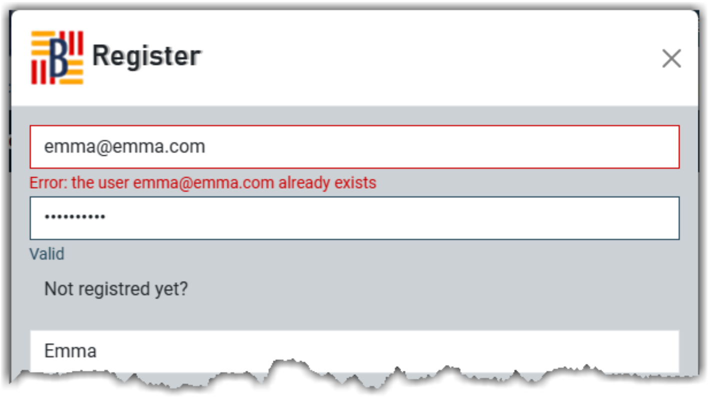
In the registration form, when validating, it executes a method that reads the database and shows an error if the user already exists.



```
// Form Validation
function validateForm(action) {

    // checks if the user exists and acts depending the action
    readDataIfExist(user.value, action);

    let isUserOK = false;
    let isPasswordOK = false;

    // Validate email
    if (user.value === '') {
```



```
// Read data to search if user exists
function readDataIfExist(userName, action) {

    openCreateDb(function (db) {

        console.log("Verify if user exists");

        var tx = db.transaction(DB_STORE_NAME, "readonly");
        var store = tx.objectStore(DB_STORE_NAME);

        var myIndex = store.index("user");
        var req = myIndex.get(userName);

        req.onsuccess = function (e) {

            var cursor = this.result;

            if (action == 'add_user') {
                // If cursor exists, there is a registered user account.
                if (cursor) {
                    errorMessage(user, 'the user ' + user.value + ' already exists');
```



Register

emma@emma.com

Error: the user emma@emma.com already exists

••••••••••

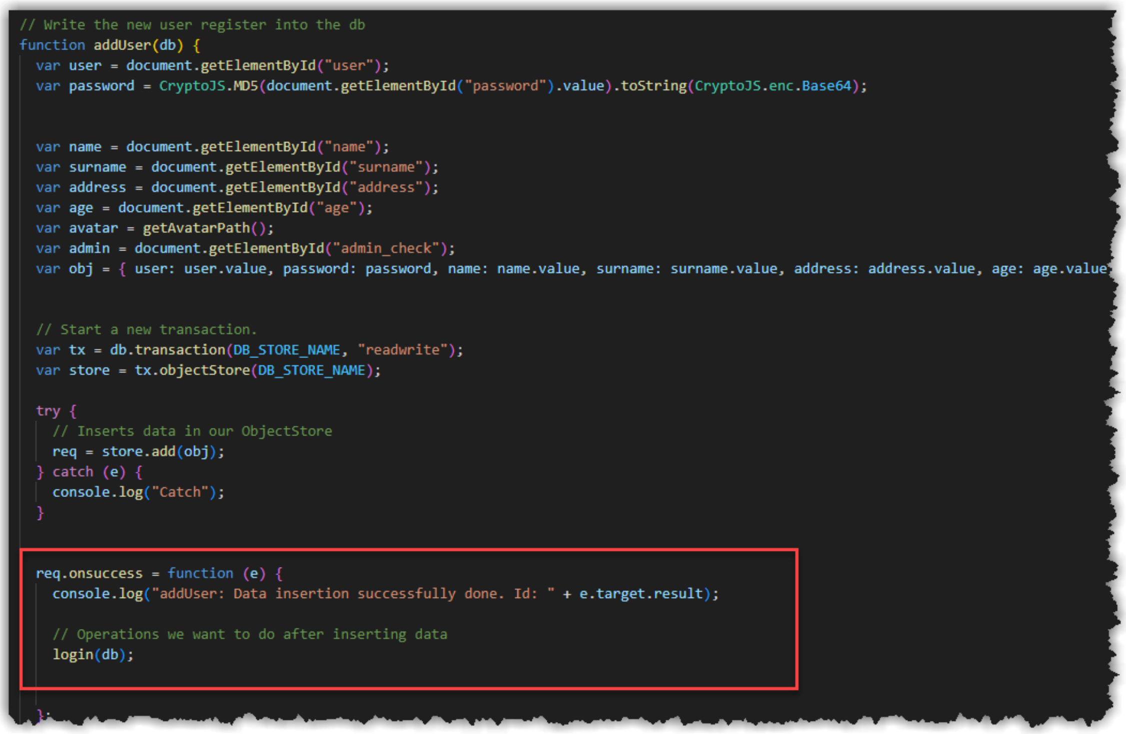
Valid

Not registred yet?

Emma

5. When a user is registered the user is automatically logged in and redirected to the homepage or admin page.

When a user registers, his data is added to the database using the function addUser(db) in the "users" storage. On success, it executes the login(db) method to log in to the web. It saves a record in the "login" storage of the database. Finally, it redirects to the website that corresponds to its role.

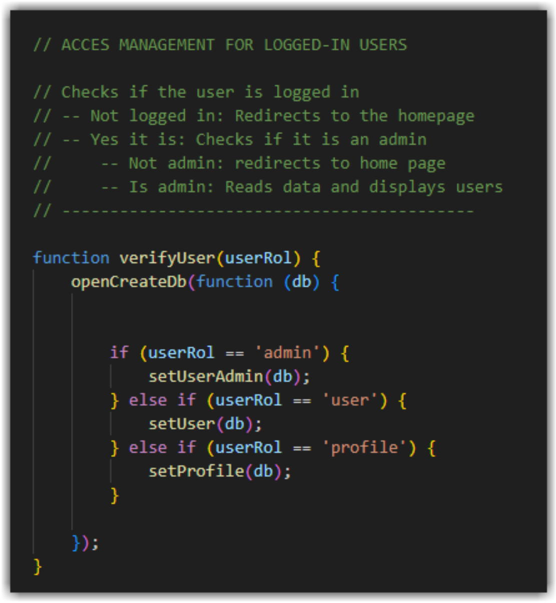


```
// Write the new user register into the db
function addUser(db) {
  var user = document.getElementById("user");
  var password = CryptoJS.MD5(document.getElementById("password").value).toString(CryptoJS.enc.Base64);


  var name = document.getElementById("name");
  var surname = document.getElementById("surname");
  var address = document.getElementById("address");
  var age = document.getElementById("age");
  var avatar = getAvatarPath();
  var admin = document.getElementById("admin_check");
  var obj = { user: user.value, password: password, name: name.value, surname: surname.value, address: address.value, age: age.value


  // Start a new transaction.
  var tx = db.transaction(DB_STORE_NAME, "readwrite");
  var store = tx.objectStore(DB_STORE_NAME);

  try {
    // Inserts data in our ObjectStore
    req = store.add(obj);
  } catch (e) {
    console.log("Catch");
  }

  req.onsuccess = function (e) {
    console.log("addUser: Data insertion successfully done. Id: " + e.target.result);

    // Operations we want to do after inserting data
    login(db);

  };
```

Because the login and registration form is the same, I use the same method for both registration and login. Simply, if it is a registration, I run the login after registering.

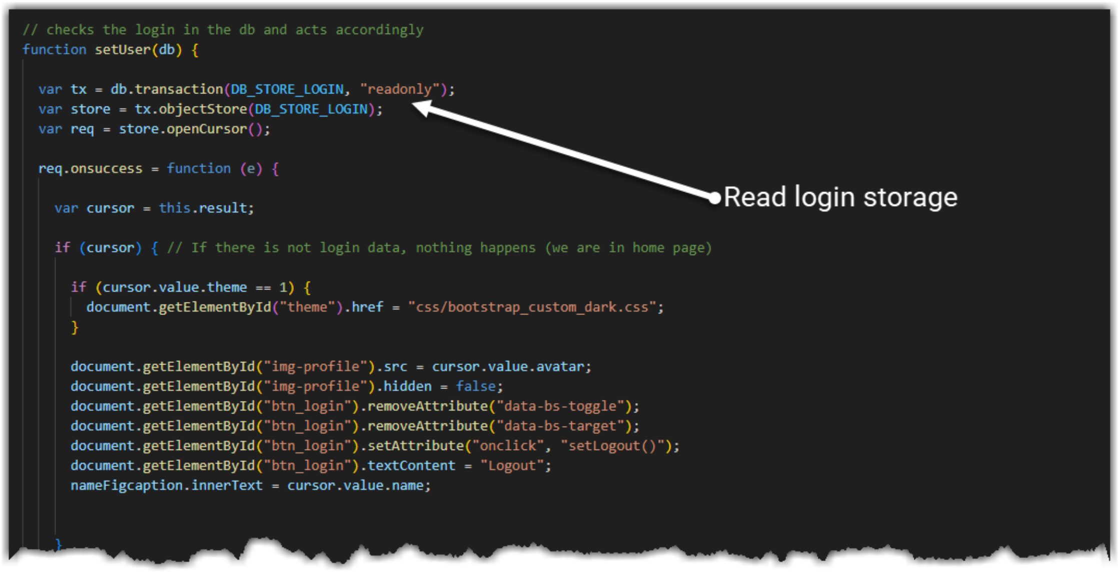
Each page has its own user verification with its required role as a parameter. Depending on the parameter (role) it executes one or another function to determine whether it has access or not and whether it is redirected.

COMMON.JS



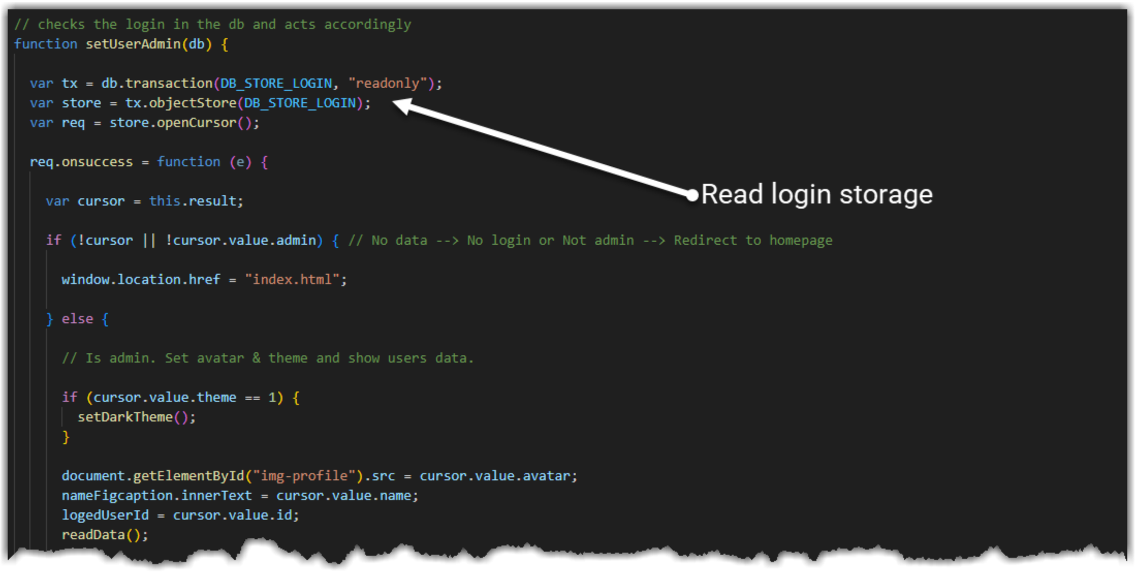
```
// ACCES MANAGEMENT FOR LOGGED-IN USERS

// Checks if the user is logged in
// -- Not logged in: Redirects to the homepage
// -- Yes it is: Checks if it is an admin
//     -- Not admin: redirects to home page
//     -- Is admin: Reads data and displays users
// -------------------------------------------

function verifyUser(userRol) {
    openCreateDb(function (db) {


        if (userRol == 'admin') {
            setUserAdmin(db);
        } else if (userRol == 'user') {
            setUser(db);
        } else if (userRol == 'profile') {
            setProfile(db);
        }

    });
}
```

INDEX.JS



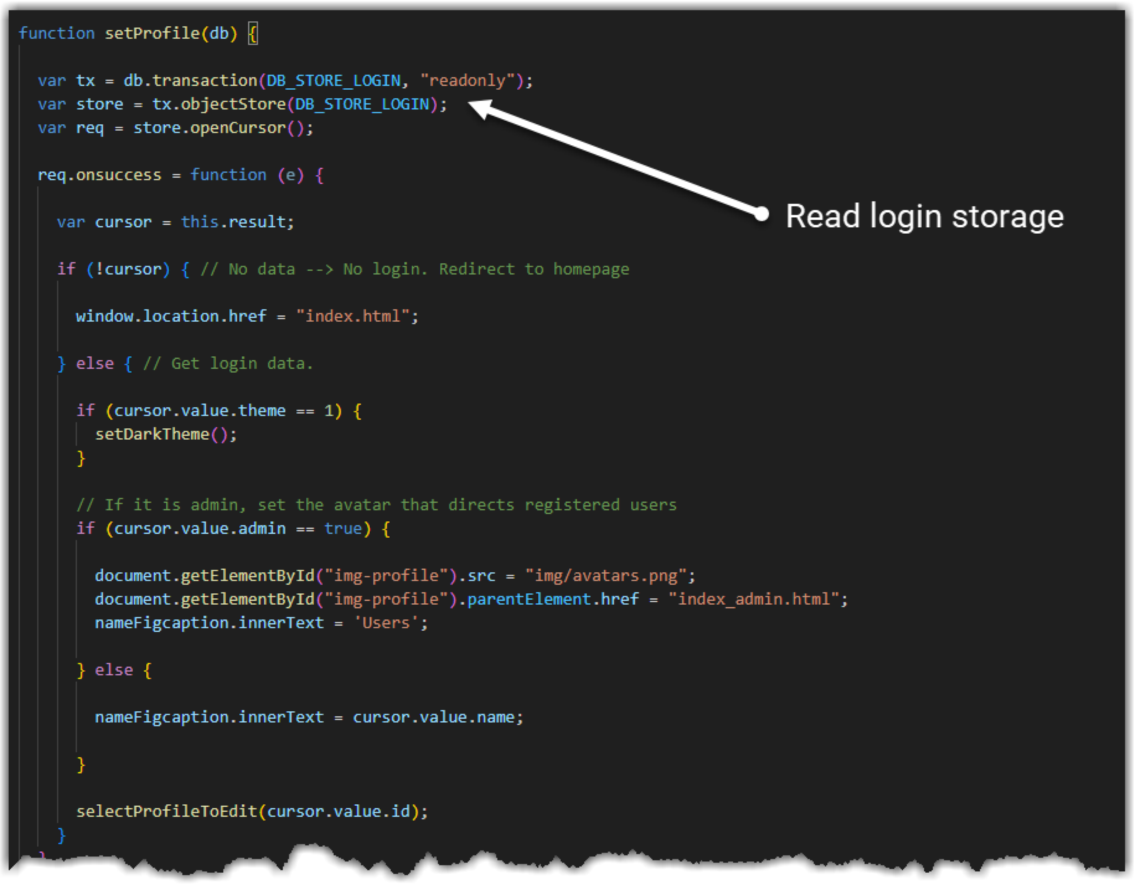
```
// checks the login in the db and acts accordingly
function setUser(db) {

  var tx = db.transaction(DB_STORE_LOGIN, "readonly");
  var store = tx.objectStore(DB_STORE_LOGIN);
  var req = store.openCursor();

  req.onsuccess = function (e) {

    var cursor = this.result;

    if (cursor) { // If there is not login data, nothing happens (we are in home page)

      if (cursor.value.theme == 1) {
        document.getElementById("theme").href = "css/bootstrap_custom_dark.css";
      }

      document.getElementById("img-profile").src = cursor.value.avatar;
      document.getElementById("img-profile").hidden = false;
      document.getElementById("btn_login").removeAttribute("data-bs-toggle");
      document.getElementById("btn_login").removeAttribute("data-bs-target");
      document.getElementById("btn_login").setAttribute("onclick", "setLogout()");
      document.getElementById("btn_login").textContent = "Logout";
      nameFigcaption.innerText = cursor.value.name;


    }
```

INDEX_ADMIN.JS



```
// checks the login in the db and acts accordingly
function setUserAdmin(db) {

  var tx = db.transaction(DB_STORE_LOGIN, "readonly");
  var store = tx.objectStore(DB_STORE_LOGIN);
  var req = store.openCursor();

  req.onsuccess = function (e) {

    var cursor = this.result;

    if (!cursor || !cursor.value.admin) { // No data --> No login or Not admin --> Redirect to homepage

      window.location.href = "index.html";

    } else {

      // Is admin. Set avatar & theme and show users data.

      if (cursor.value.theme == 1) {
        setDarkTheme();
      }

      document.getElementById("img-profile").src = cursor.value.avatar;
      nameFigcaption.innerText = cursor.value.name;
      logedUserId = cursor.value.id;
      readData();
```

INDEX_PROFILE.JS

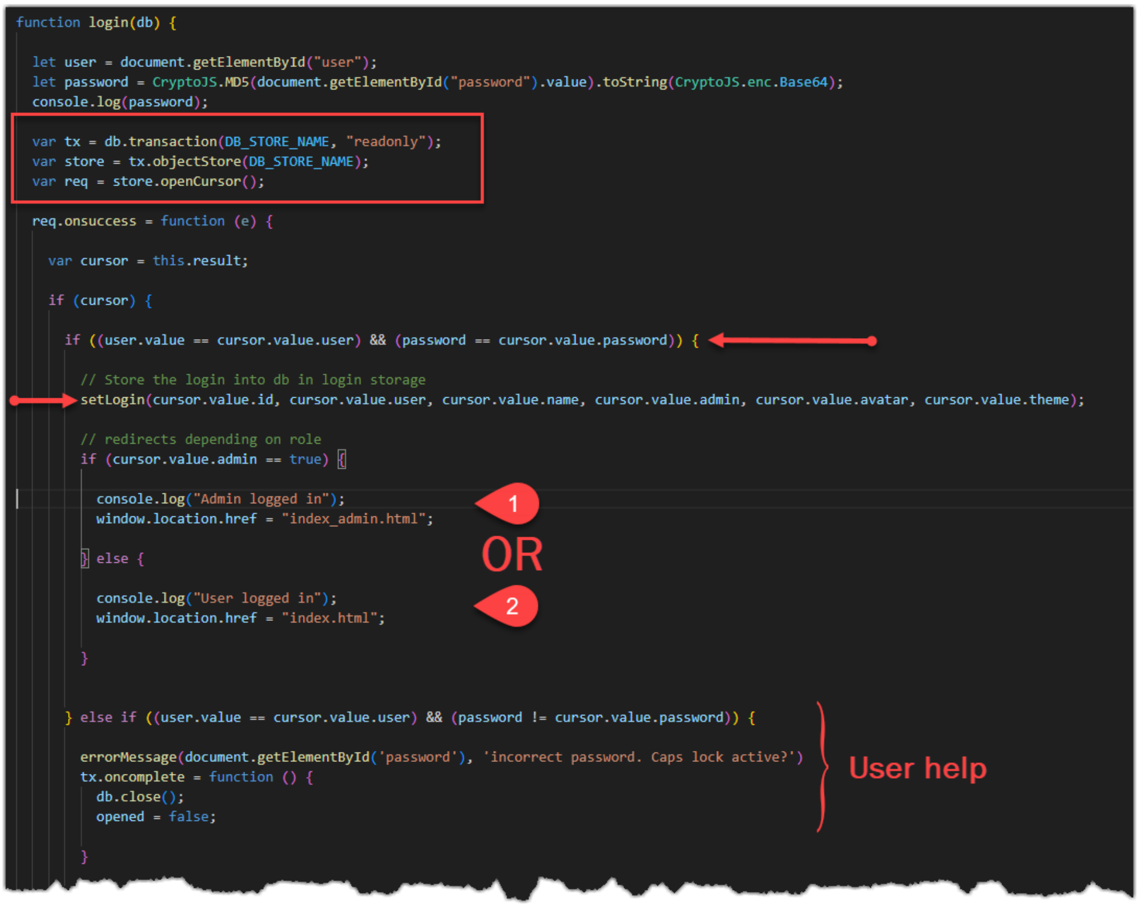


```
function setProfile(db) {

  var tx = db.transaction(DB_STORE_LOGIN, "readonly");
  var store = tx.objectStore(DB_STORE_LOGIN);
  var req = store.openCursor();

  req.onsuccess = function (e) {

    var cursor = this.result;

    if (!cursor) { // No data --> No login. Redirect to homepage

      window.location.href = "index.html";

    } else { // Get login data.

      if (cursor.value.theme == 1) {
        setDarkTheme();
      }

      // If it is admin, set the avatar that directs registered users
      if (cursor.value.admin == true) {

        document.getElementById("img-profile").src = "img/avatars.png";
        document.getElementById("img-profile").parentElement.href = "index_admin.html";
        nameFigcaption.innerText = 'Users';

      } else {

        nameFigcaption.innerText = cursor.value.name;

      }

      selectProfileToEdit(cursor.value.id);
    }
```

LOGIN

1. IndexedDB is read when somebody tries to login. User and password are checked in order to log in.
2. When a user is logged in, it is redirected to homepage or admin page.

When a user logs in, the application accesses the database and reads the records, checking one by one if the user and password match. If it matches a record, it creates the login record (setLogin()) and redirects to the corresponding page according to its role.

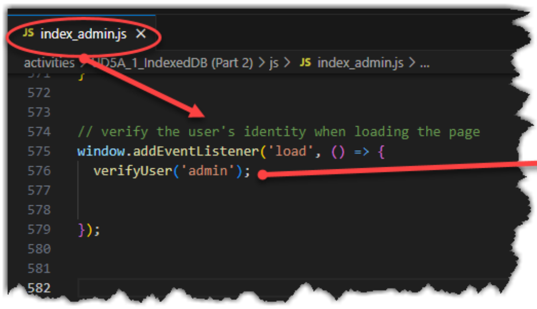


```
function login(db) {

  let user = document.getElementById("user");
  let password = CryptoJS.MD5(document.getElementById("password").value).toString(CryptoJS.enc.Base64);
  console.log(password);

  var tx = db.transaction(DB_STORE_NAME, "readonly");
  var store = tx.objectStore(DB_STORE_NAME);
  var req = store.openCursor();

  req.onsuccess = function (e) {

    var cursor = this.result;

    if (cursor) {

      if ((user.value == cursor.value.user) && (password == cursor.value.password)) {

        // Store the login into db in login storage
        setLogin(cursor.value.id, cursor.value.user, cursor.value.name, cursor.value.admin, cursor.value.avatar, cursor.value.theme);

        // redirects depending on role
        if (cursor.value.admin == true) {

          console.log("Admin logged in");
          window.location.href = "index_admin.html";

        } else {

          console.log("User logged in");
          window.location.href = "index.html";

        }


      } else if ((user.value == cursor.value.user) && (password != cursor.value.password)) {

        errorMessage(document.getElementById('password'), 'incorrect password. Caps lock active?')
        tx.oncomplete = function () {
          db.close();
          opened = false;

        }
```

ADMIN

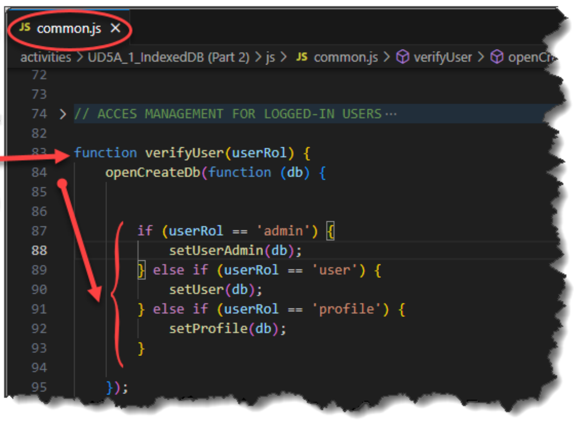
1. If we go to this page (or to any other) and no user or a standard user is logged in, the page must be redirect to the homepage or avoid see or edit information.

On each page there is a *listener* with a *load* event that reads from the login storage oh the db. If there is no login register, it is not logged in and redirects to the home page. In the case of the admin page where you see the logged in users, if there is a login, but it is not admin, it also redirects to the home page.


index_admin.js

activities > UD5A_1_IndexedDB (Part 2) > js > index_admin.js > ...

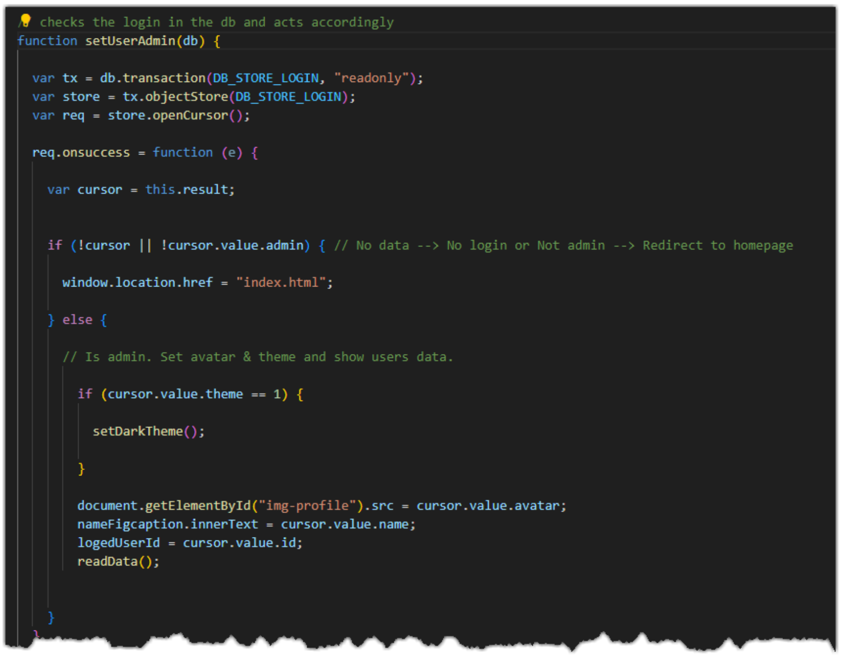
```
}


// verify the user's identity when loading the page
window.addEventListener('load', () => {
  verifyUser('admin');


});


```


common.js

activities > UD5A_1_IndexedDB (Part 2) > js > common.js > verifyUser > openCr

```


> // ACCES MANAGEMENT FOR LOGGED-IN USERS···

function verifyUser(userRol) {
    openCreateDb(function (db) {


        if (userRol == 'admin') {
            setUserAdmin(db);
        } else if (userRol == 'user') {
            setUser(db);
        } else if (userRol == 'profile') {
            setProfile(db);
        }

    });
```

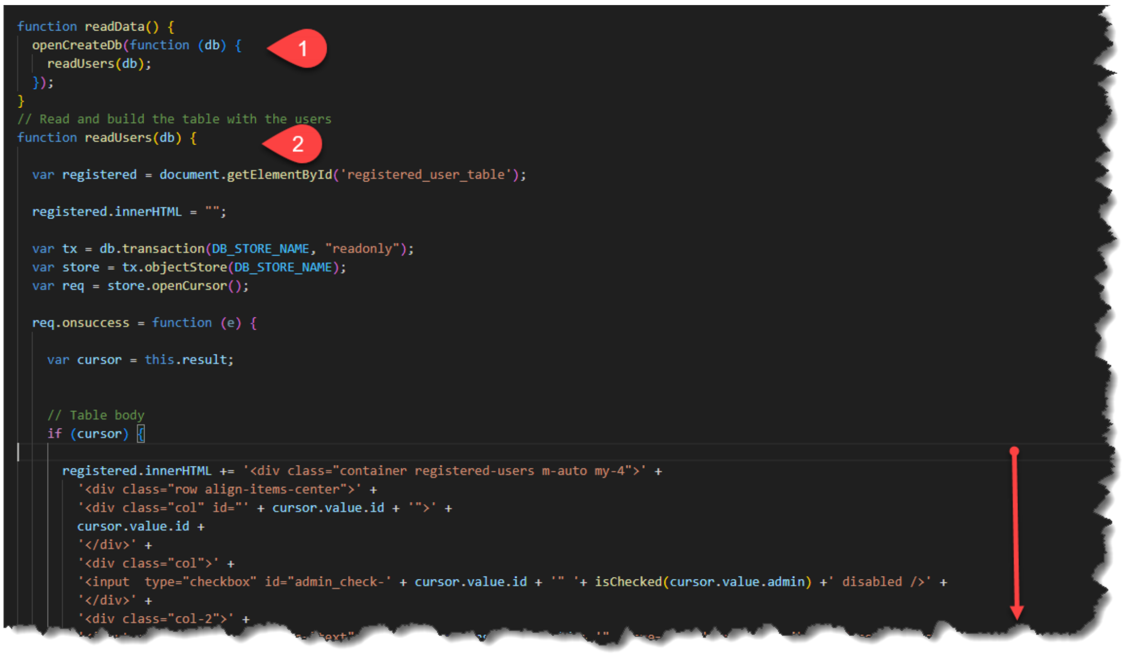
SETUSERADMIN(DB)

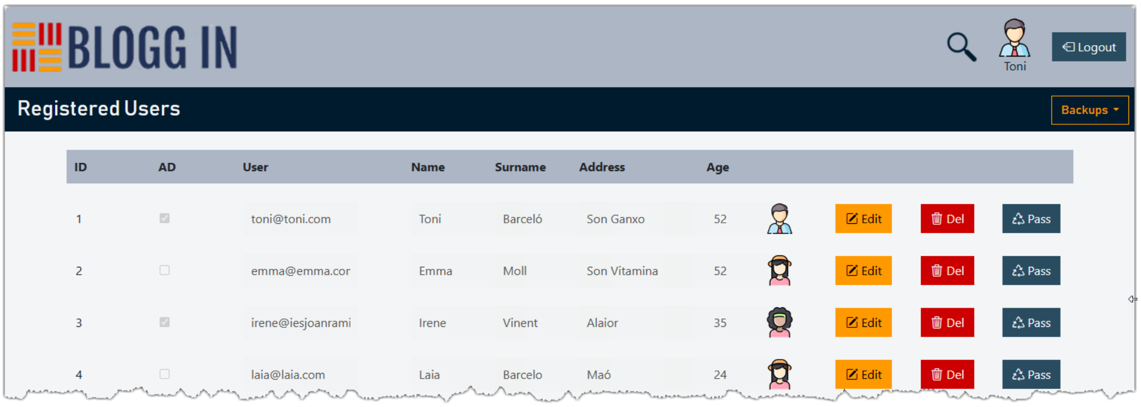


```
// checks the login in the db and acts accordingly
function setUserAdmin(db) {

  var tx = db.transaction(DB_STORE_LOGIN, "readonly");
  var store = tx.objectStore(DB_STORE_LOGIN);
  var req = store.openCursor();

  req.onsuccess = function (e) {

    var cursor = this.result;


    if (!cursor || !cursor.value.admin) { // No data --> No login or Not admin --> Redirect to homepage

      window.location.href = "index.html";

    } else {

      // Is admin. Set avatar & theme and show users data.

        if (cursor.value.theme == 1) {

          setDarkTheme();

        }

        document.getElementById("img-profile").src = cursor.value.avatar;
        nameFigcaption.innerText = cursor.value.name;
        logedUserId = cursor.value.id;
        readData();


    }
  }
```

2. When we enter this page we can see a list of users created on the database (each time we enter here, users are read from indexedDB)

Once logged in as admin and inside the page, run the read() function that opens the database, reads it (readUsers(db)) and displays all registered users.


```
function readData() {
  openCreateDb(function (db) {
    readUsers(db);
  });
}
// Read and build the table with the users
function readUsers(db) {

  var registered = document.getElementById('registered_user_table');

  registered.innerHTML = "";

  var tx = db.transaction(DB_STORE_NAME, "readonly");
  var store = tx.objectStore(DB_STORE_NAME);
  var req = store.openCursor();

  req.onsuccess = function (e) {

    var cursor = this.result;


    // Table body
    if (cursor) {

      registered.innerHTML += '<div class="container registered-users m-auto my-4">' +
        '<div class="row align-items-center">' +
        '<div class="col" id="' + cursor.value.id + '">' +
        cursor.value.id +
        '</div>' +
        '<div class="col">' +
        '<input  type="checkbox" id="admin_check-' + cursor.value.id + '" '+ isChecked(cursor.value.admin) +' disabled />' +
        '</div>' +
        '<div class="col-2">' +
```

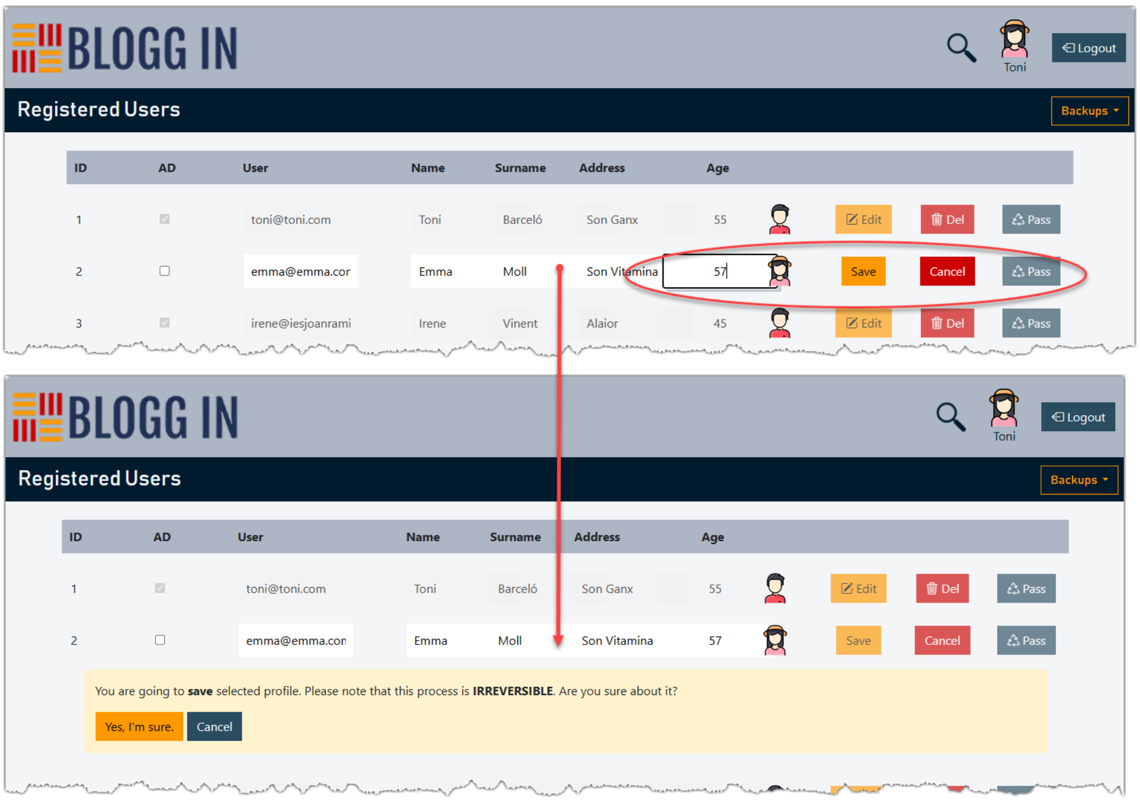


Registered Users

ID	AD	User	Name	Surname	Address	Age				
1	✓	toni@toni.com	Toni	Barceló	Son Ganxo	52		Edit	Del	Pass
2	☐	emma@emma.cor	Emma	Moll	Son Vitamina	52		Edit	Del	Pass
3	✓	irene@iesjoanrami	Irene	Vinent	Alaior	35		Edit	Del	Pass
4	☐	laia@laia.com	Laia	Barcelo	Maó	24		Edit	Del	Pass

3. We can edit user information and this is updated without refreshing the page.

In each record there is a button to edit the record. The fields are edited online in the record table itself. When the edit button is pressed, the function editFields(user_id) is called which enables the inputs, disables the rest of the buttons of the other records and changes the function of the button to confirmEdit(user_id) which asks for confirmation to save the edited data.

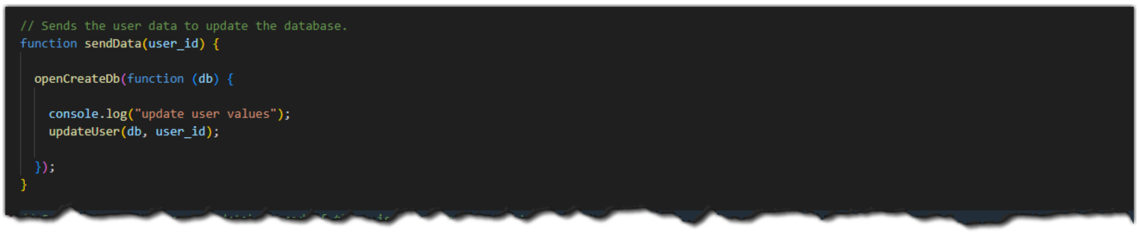


ID	AD	User	Name	Surname	Address	Age				
1	☑	toni@toni.com	Toni	Barceló	Son Ganx	55		Edit	Del	Pass
2	☐	emma@emma.cor	Emma	Moll	Son Vitamina	57		Save	Cancel	Pass
3	☑	irene@iesjoanrami	Irene	Vinent	Alaior	45		Edit	Del	Pass

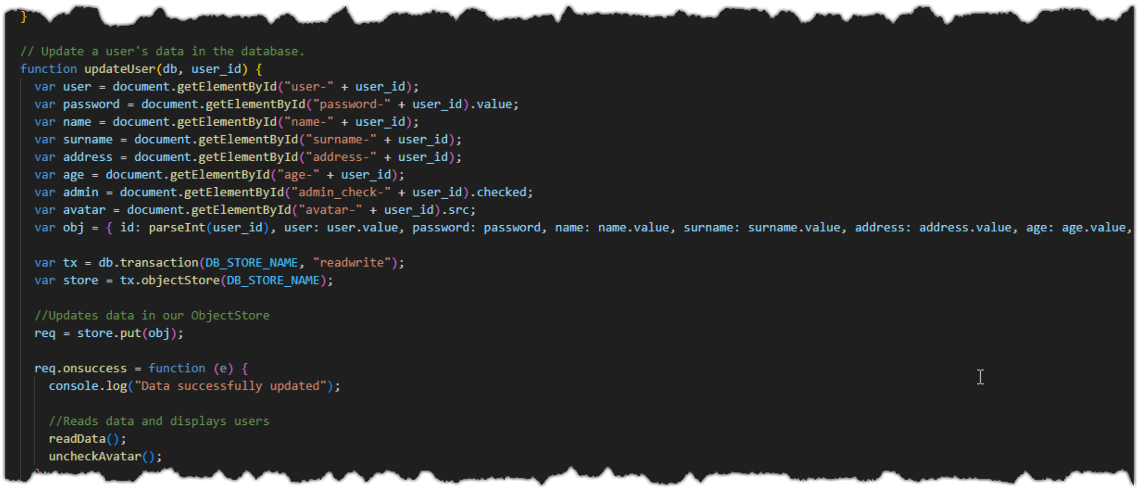
ID	AD	User	Name	Surname	Address	Age				
1	☑	toni@toni.com	Toni	Barceló	Son Ganx	55		Edit	Del	Pass
2	☐	emma@emma.con	Emma	Moll	Son Vitamina	57		Save	Cancel	Pass

You are going to **save** selected profile. Please note that this process is **IRREVERSIBLE**. Are you sure about it?

After confirmation, we execute the function sendData(user_id) which opens the database and calls updateUser(db, user_id) to collect the current data in all inputs and update them in the database. In the onsucces event of the update, it executes the read() function to reload the users without the need to reload the whole page.



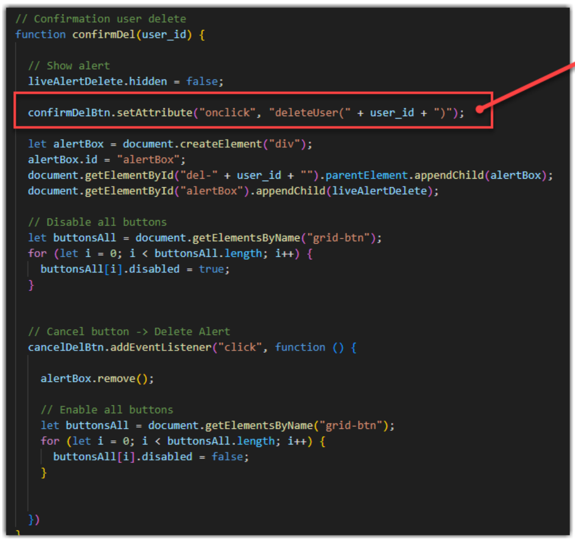
```
// Sends the user data to update the database.
function sendData(user_id) {

  openCreateDb(function (db) {

    console.log("update user values");
    updateUser(db, user_id);

  });
}
```



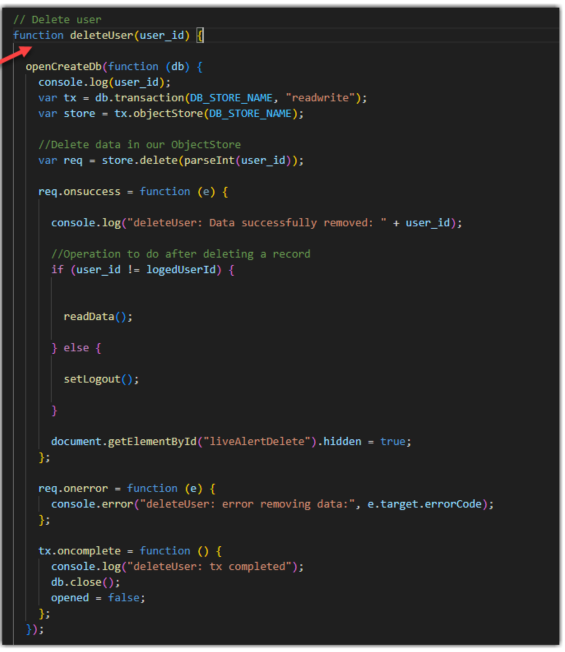
```
}

// Update a user's data in the database.
function updateUser(db, user_id) {
  var user = document.getElementById("user-" + user_id);
  var password = document.getElementById("password-" + user_id).value;
  var name = document.getElementById("name-" + user_id);
  var surname = document.getElementById("surname-" + user_id);
  var address = document.getElementById("address-" + user_id);
  var age = document.getElementById("age-" + user_id);
  var admin = document.getElementById("admin_check-" + user_id).checked;
  var avatar = document.getElementById("avatar-" + user_id).src;
  var obj = { id: parseInt(user_id), user: user.value, password: password, name: name.value, surname: surname.value, address: address.value, age: age.value,

  var tx = db.transaction(DB_STORE_NAME, "readwrite");
  var store = tx.objectStore(DB_STORE_NAME);

  //Updates data in our ObjectStore
  req = store.put(obj);

  req.onsuccess = function (e) {
    console.log("Data successfully updated");

    //Reads data and displays users
    readData();
    uncheckAvatar();
```

4. We can delete users and the information is updated without refreshing.

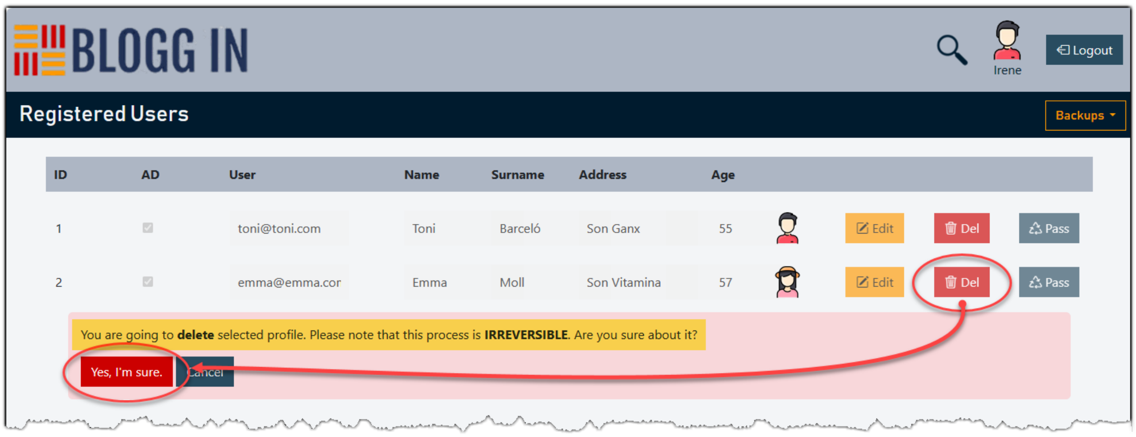
As in editing, each record has a button that deletes the record itself. When clicked, it calls the function confrimDel(user_id) which displays a confirmation panel with a button that finally deletes the record when clicked by executing the function deleteUser(user_id).



```
// Confirmation user delete
function confirmDel(user_id) {

  // Show alert
  liveAlertDelete.hidden = false;

  confirmDelBtn.setAttribute("onclick", "deleteUser(" + user_id + ")");

  let alertBox = document.createElement("div");
  alertBox.id = "alertBox";
  document.getElementById("del-" + user_id + "").parentElement.appendChild(alertBox);
  document.getElementById("alertBox").appendChild(liveAlertDelete);

  // Disable all buttons
  let buttonsAll = document.getElementsByName("grid-btn");
  for (let i = 0; i < buttonsAll.length; i++) {
    buttonsAll[i].disabled = true;
  }


  // Cancel button -> Delete Alert
  cancelDelBtn.addEventListener("click", function () {

    alertBox.remove();

    // Enable all buttons
    let buttonsAll = document.getElementsByName("grid-btn");
    for (let i = 0; i < buttonsAll.length; i++) {
      buttonsAll[i].disabled = false;
    }


  })
```



```
// Delete user
function deleteUser(user_id) {

  openCreateDb(function (db) {
    console.log(user_id);
    var tx = db.transaction(DB_STORE_NAME, "readwrite");
    var store = tx.objectStore(DB_STORE_NAME);

    //Delete data in our ObjectStore
    var req = store.delete(parseInt(user_id));

    req.onsuccess = function (e) {

      console.log("deleteUser: Data successfully removed: " + user_id);

      //Operation to do after deleting a record
      if (user_id != logedUserId) {


        readData();

      } else {

        setLogout();

      }

      document.getElementById("liveAlertDelete").hidden = true;
    };

    req.onerror = function (e) {
      console.error("deleteUser: error removing data:", e.target.errorCode);
    };

    tx.oncomplete = function () {
      console.log("deleteUser: tx completed");
      db.close();
      opened = false;
    };
  });
```



Registered Users

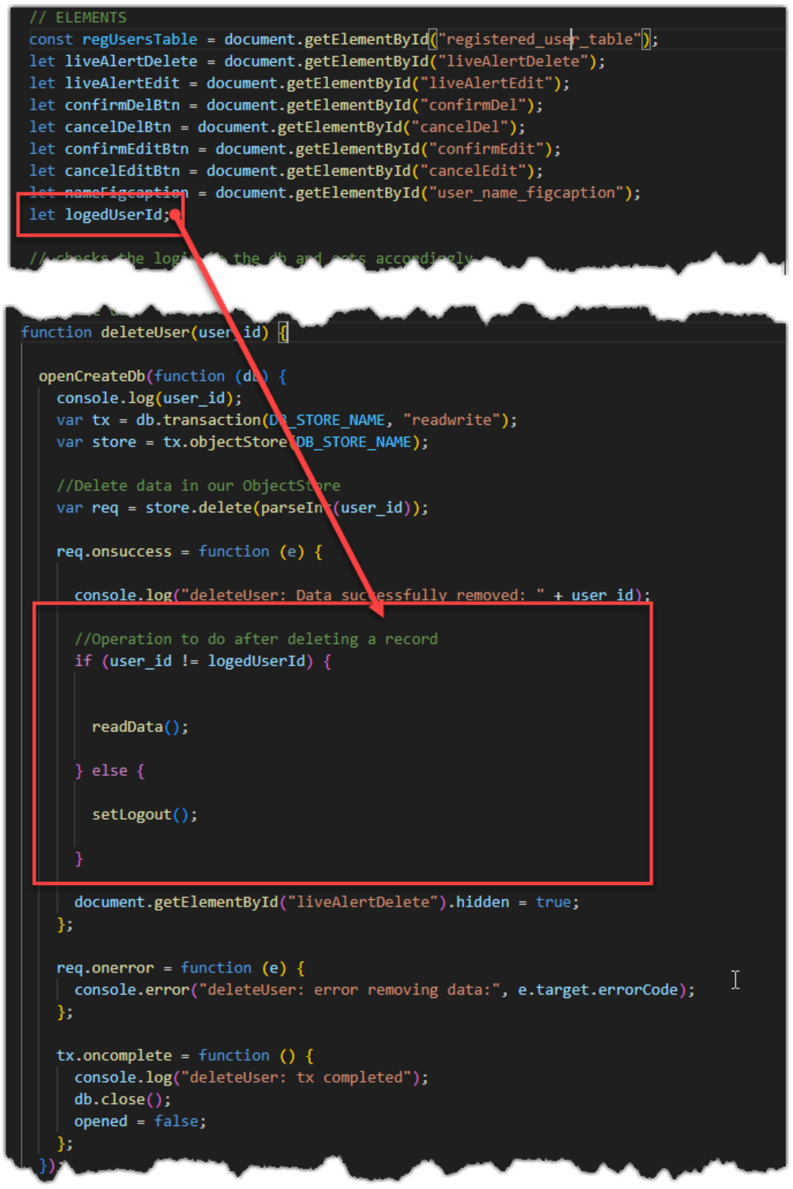
ID	AD	User	Name	Surname	Address	Age				
1	✓	toni@toni.com	Toni	Barceló	Son Ganx	55		Edit	Del	Pass
2	✓	emma@emma.cor	Emma	Moll	Son Vitamina	57		Edit	Del	Pass

You are going to **delete** selected profile. Please note that this process is **IRREVERSIBLE**. Are you sure about it?

Yes, I'm sure. Cancel

5. If the user deleted is the one we are logged in with, the app must logout automatically.

To achieve this, the id of the user to be deleted is assigned to a global variable when the 'Delete' button is clicked. After confirmation, a condition decides that if the user_id does not match the user_id of the user to be deleted, it reloads the users, otherwise it logs out.

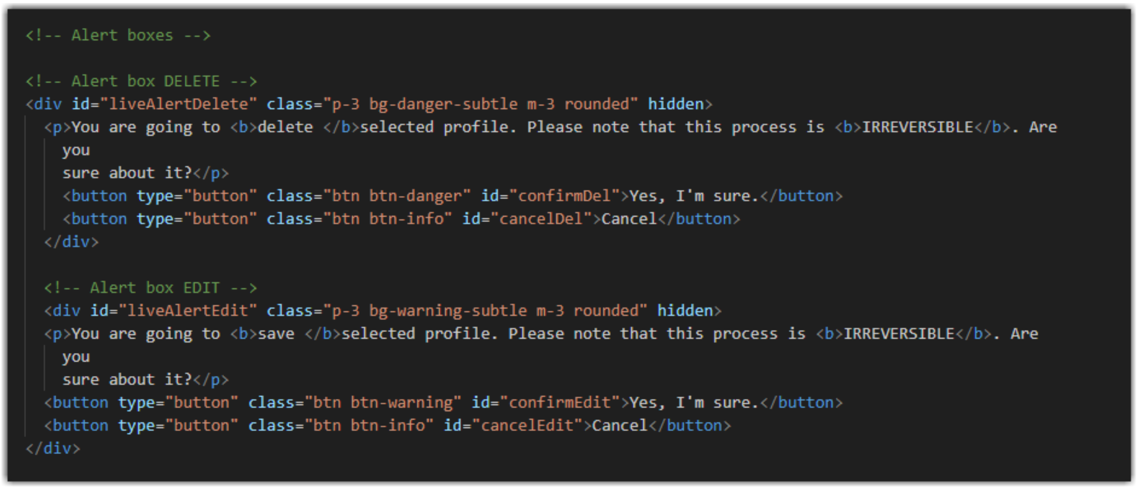


```
// ELEMENTS
const regUsersTable = document.getElementById("registered_user_table");
let liveAlertDelete = document.getElementById("liveAlertDelete");
let liveAlertEdit = document.getElementById("liveAlertEdit");
let confirmDelBtn = document.getElementById("confirmDel");
let cancelDelBtn = document.getElementById("cancelDel");
let confirmEditBtn = document.getElementById("confirmEdit");
let cancelEditBtn = document.getElementById("cancelEdit");
let nameFigcaption = document.getElementById("user_name_figcaption");
let logedUserId;

// checks the login in the db and acts accordingly
```

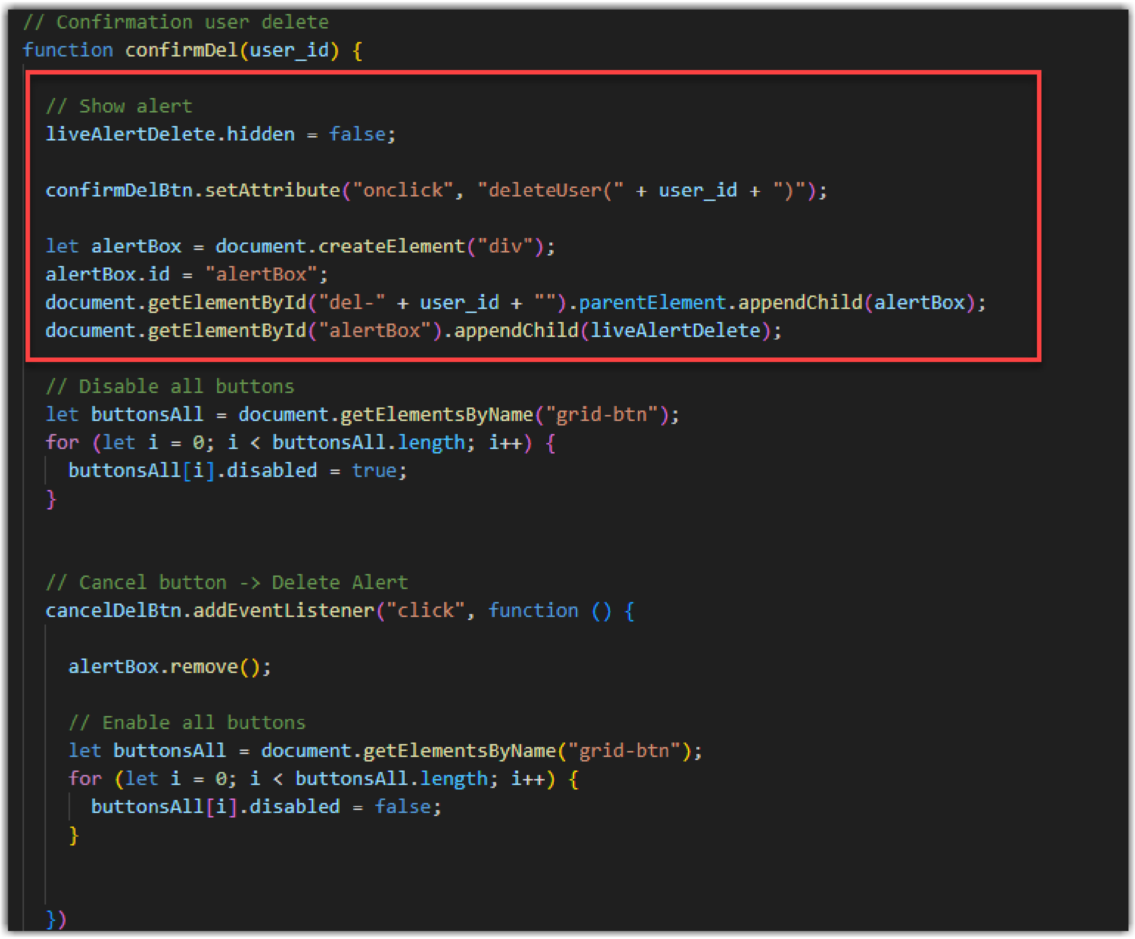
```
function deleteUser(user_id) {

  openCreateDb(function (db) {
    console.log(user_id);
    var tx = db.transaction(DB_STORE_NAME, "readwrite");
    var store = tx.objectStore(DB_STORE_NAME);

    //Delete data in our ObjectStore
    var req = store.delete(parseInt(user_id));

    req.onsuccess = function (e) {

      console.log("deleteUser: Data successfully removed: " + user_id);

      //Operation to do after deleting a record
      if (user_id != logedUserId) {


        readData();

      } else {

        setLogout();

      }

      document.getElementById("liveAlertDelete").hidden = true;
    };

    req.onerror = function (e) {
      console.error("deleteUser: error removing data:", e.target.errorCode);
    };

    tx.oncomplete = function () {
      console.log("deleteUser: tx completed");
      db.close();
      opened = false;
    };
  });
```

6. There's a confirmation before deleting a user.

In the html file are the 'alert box'. These are containers that follow the style of the website with corresponding confirmation buttons for deleting and saving changes. They are initially hidden and are displayed and placed below the record when the 'Delete' or 'Save' button is pressed.



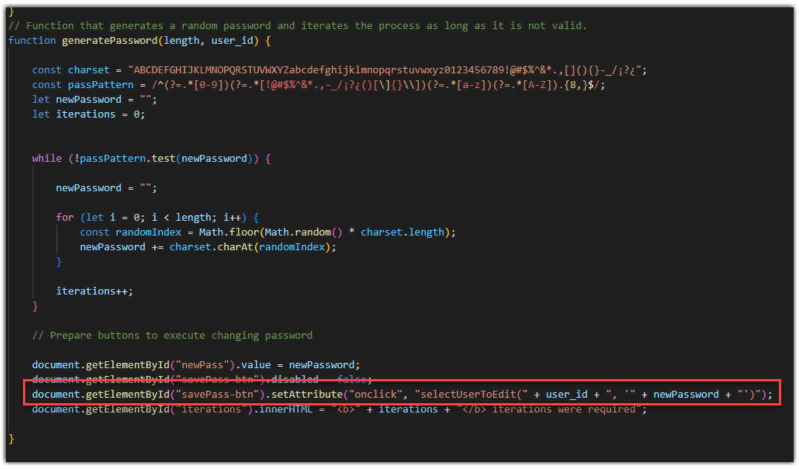
```html
<!-- Alert boxes -->

<!-- Alert box DELETE -->
<div id="liveAlertDelete" class="p-3 bg-danger-subtle m-3 rounded" hidden>
  <p>You are going to <b>delete </b>selected profile. Please note that this process is <b>IRREVERSIBLE</b>. Are
    you
    sure about it?</p>
    <button type="button" class="btn btn-danger" id="confirmDel">Yes, I'm sure.</button>
    <button type="button" class="btn btn-info" id="cancelDel">Cancel</button>
  </div>

  <!-- Alert box EDIT -->
  <div id="liveAlertEdit" class="p-3 bg-warning-subtle m-3 rounded" hidden>
  <p>You are going to <b>save </b>selected profile. Please note that this process is <b>IRREVERSIBLE</b>. Are
    you
    sure about it?</p>
  <button type="button" class="btn btn-warning" id="confirmEdit">Yes, I'm sure.</button>
  <button type="button" class="btn btn-info" id="cancelEdit">Cancel</button>
</div>
```



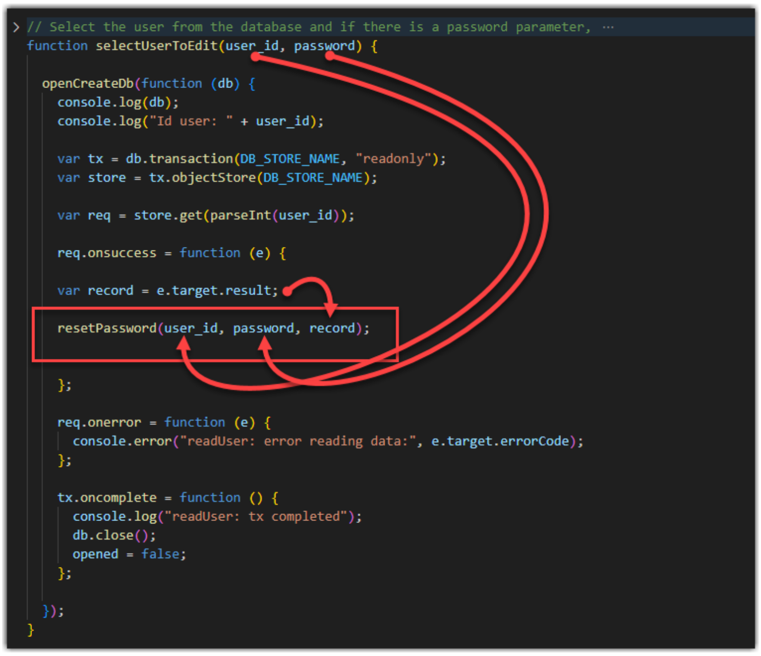
```javascript
// Confirmation user delete
function confirmDel(user_id) {

  // Show alert
  liveAlertDelete.hidden = false;

  confirmDelBtn.setAttribute("onclick", "deleteUser(" + user_id + ")");

  let alertBox = document.createElement("div");
  alertBox.id = "alertBox";
  document.getElementById("del-" + user_id + "").parentElement.appendChild(alertBox);
  document.getElementById("alertBox").appendChild(liveAlertDelete);

  // Disable all buttons
  let buttonsAll = document.getElementsByName("grid-btn");
  for (let i = 0; i < buttonsAll.length; i++) {
    buttonsAll[i].disabled = true;
  }


  // Cancel button -> Delete Alert
  cancelDelBtn.addEventListener("click", function () {

    alertBox.remove();

    // Enable all buttons
    let buttonsAll = document.getElementsByName("grid-btn");
    for (let i = 0; i < buttonsAll.length; i++) {
      buttonsAll[i].disabled = false;
    }


  })
```

7. There's an option to change only the password.

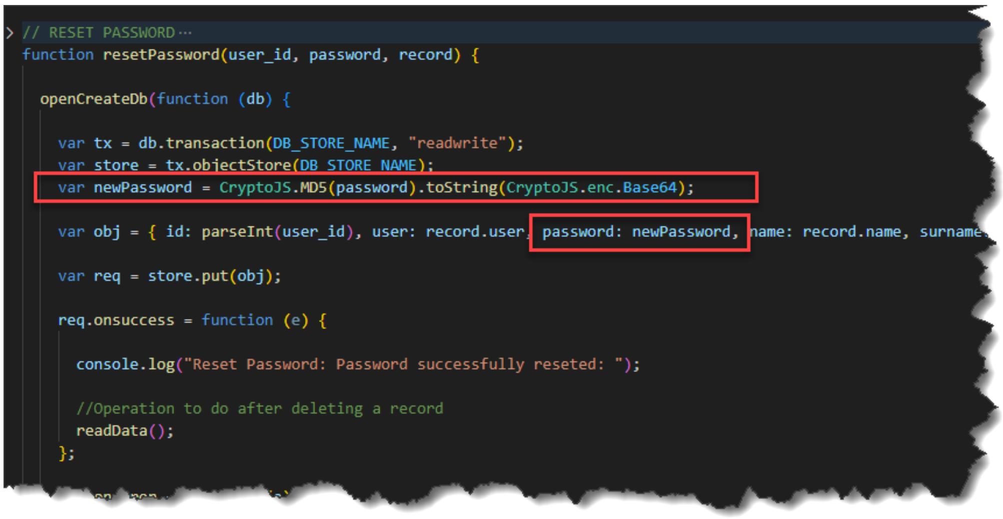
When you click on the change password button, it launches a modal where you can generate a randomly generated secure password. When saving, the hash is inserted into the database as it is encrypted.



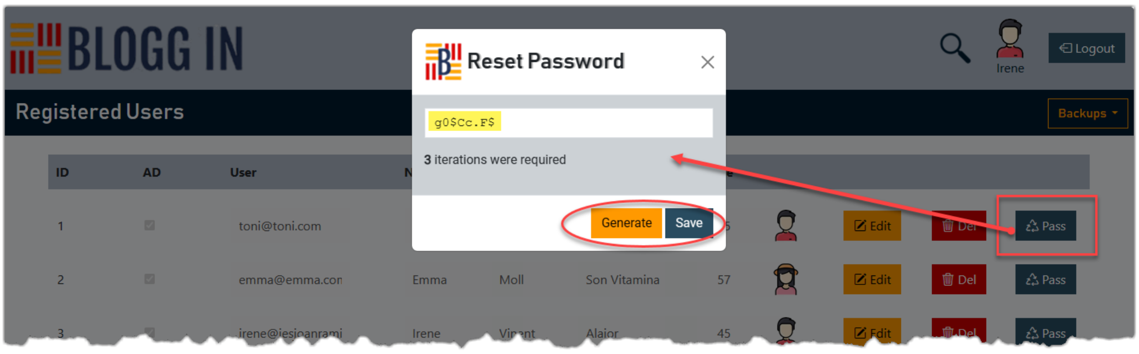
```
}
// Function that generates a random password and iterates the process as long as it is not valid.
function generatePassword(length, user_id) {

    const charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*.,[](){}-_/¡?¿";
    const passPattern = /^(?=.*[0-9])(?=.*[!@#$%^&*.,-_/¡?¿()[\]{}\\])(?=.*[a-z])(?=.*[A-Z]).{8,}$/;
    let newPassword = "";
    let iterations = 0;


    while (!passPattern.test(newPassword)) {

        newPassword = "";

        for (let i = 0; i < length; i++) {
            const randomIndex = Math.floor(Math.random() * charset.length);
            newPassword += charset.charAt(randomIndex);
        }

        iterations++;
    }

    // Prepare buttons to execute changing password

    document.getElementById("newPass").value = newPassword;
    document.getElementById("savePass-btn").disabled = false;
    document.getElementById("savePass-btn").setAttribute("onclick", "selectUserToEdit(" + user_id + ", '" + newPassword + "')");
    document.getElementById("iterations").innerHTML = "<b>" + iterations + "</b> iterations were required";

}
```



```
> // Select the user from the database and if there is a password parameter, …
  function selectUserToEdit(user_id, password) {

    openCreateDb(function (db) {
      console.log(db);
      console.log("Id user: " + user_id);

      var tx = db.transaction(DB_STORE_NAME, "readonly");
      var store = tx.objectStore(DB_STORE_NAME);

      var req = store.get(parseInt(user_id));

      req.onsuccess = function (e) {

      var record = e.target.result;

      resetPassword(user_id, password, record);


      };

      req.onerror = function (e) {
        console.error("readUser: error reading data:", e.target.errorCode);
      };

      tx.oncomplete = function () {
        console.log("readUser: tx completed");
        db.close();
        opened = false;
      };

    });
  }
```



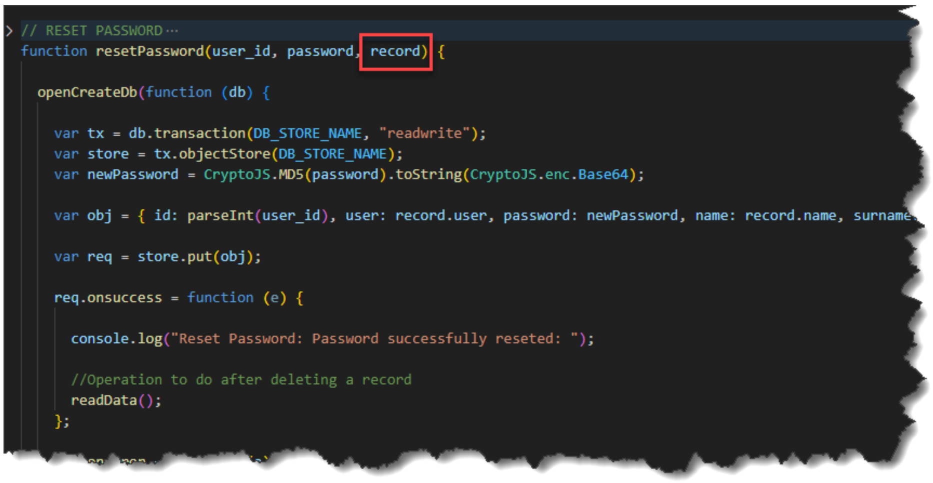
```
// RESET PASSWORD···
function resetPassword(user_id, password, record) {

  openCreateDb(function (db) {

    var tx = db.transaction(DB_STORE_NAME, "readwrite");
    var store = tx.objectStore(DB_STORE_NAME);
    var newPassword = CryptoJS.MD5(password).toString(CryptoJS.enc.Base64);

    var obj = { id: parseInt(user_id), user: record.user, password: newPassword, name: record.name, surname

    var req = store.put(obj);

    req.onsuccess = function (e) {

      console.log("Reset Password: Password successfully reseted: ");

      //Operation to do after deleting a record
      readData();
    };
```



ID	AD	User	N							
1	☑	toni@toni.com				5		Edit	Del	Pass
2	☑	emma@emma.con	Emma	Moll	Son Vitamina	57		Edit	Del	Pass
3	☑	irene@iesjoanrami	Irene	Virent	Alaior	45		Edit	Del	Pass

8. No information is lost after saving changes.

As can be seen in point 7, before changing the data in the database, they are collected from the record to be updated and all of them are inserted.



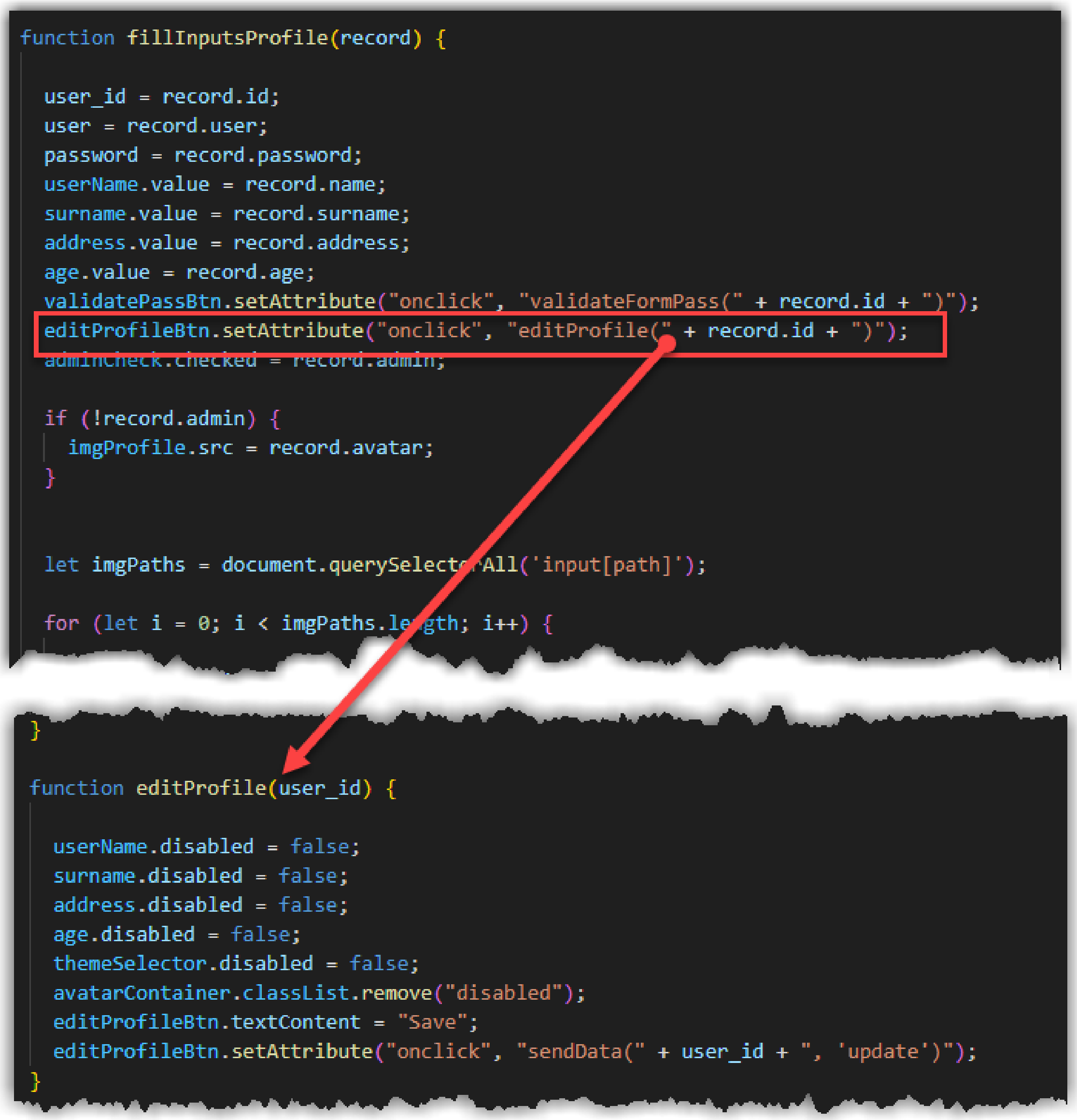
```
// RESET PASSWORD···
function resetPassword(user_id, password, record) {

  openCreateDb(function (db) {

    var tx = db.transaction(DB_STORE_NAME, "readwrite");
    var store = tx.objectStore(DB_STORE_NAME);
    var newPassword = CryptoJS.MD5(password).toString(CryptoJS.enc.Base64);

    var obj = { id: parseInt(user_id), user: record.user, password: newPassword, name: record.name, surname

    var req = store.put(obj);

    req.onsuccess = function (e) {

      console.log("Reset Password: Password successfully reseted: ");

      //Operation to do after deleting a record
      readData();
    };
```

SETTINGS

1. There's an option to edit personal data. Changes are saved. No data lost.

The index_profile.html page is where we will manage user data. After logging in, our avatar is displayed. If we click on it, we go to our profile.

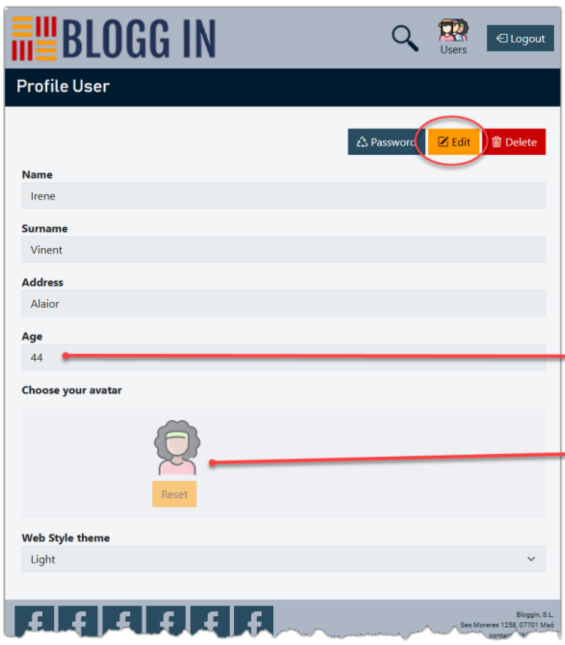
When loading the page, it verifies that we have logged in and selects all the user data with selectProfileData(user_id, password) function and fill it in inputs using the function fillInputsProfile('record') as well as adding a new *onclick* event on the edit button that calls the function editProfile('user_id') that edits the inputs when clicked on.



```
function fillInputsProfile(record) {

  user_id = record.id;
  user = record.user;
  password = record.password;
  userName.value = record.name;
  surname.value = record.surname;
  address.value = record.address;
  age.value = record.age;
  validatePassBtn.setAttribute("onclick", "validateFormPass(" + record.id + ")");
  editProfileBtn.setAttribute("onclick", "editProfile(" + record.id + ")");
  adminCheck.checked = record.admin;

  if (!record.admin) {
    imgProfile.src = record.avatar;
  }


  let imgPaths = document.querySelectorAll('input[path]');

  for (let i = 0; i < imgPaths.length; i++) {
```

```
}

function editProfile(user_id) {

  userName.disabled = false;
  surname.disabled = false;
  address.disabled = false;
  age.disabled = false;
  themeSelector.disabled = false;
  avatarContainer.classList.remove("disabled");
  editProfileBtn.textContent = "Save";
  editProfileBtn.setAttribute("onclick", "sendData(" + user_id + ", 'update')");
}
```

Once the changes have been made, executing the Save button will execute the sendData(user_id, 'update') which updates the data, all without losing any of them.



Profile User

Name: Irene

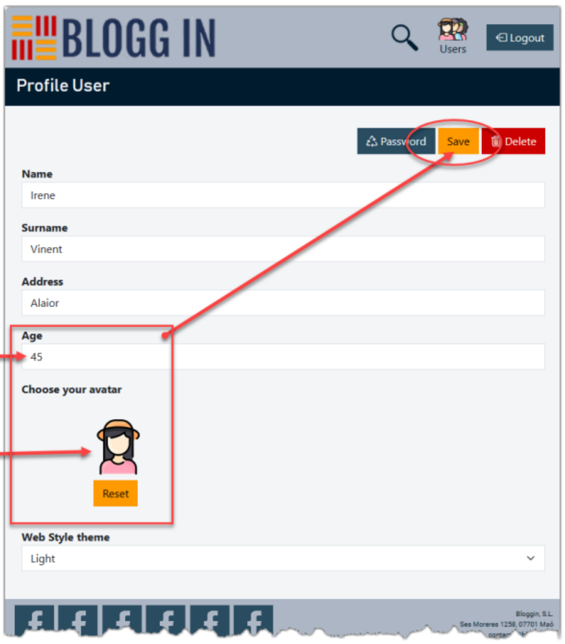
Surname: Vinent

Address: Alaior

Age: 44

Choose your avatar

Web Style theme: Light



Profile User

Name: Irene

Surname: Vinent

Address: Alaior

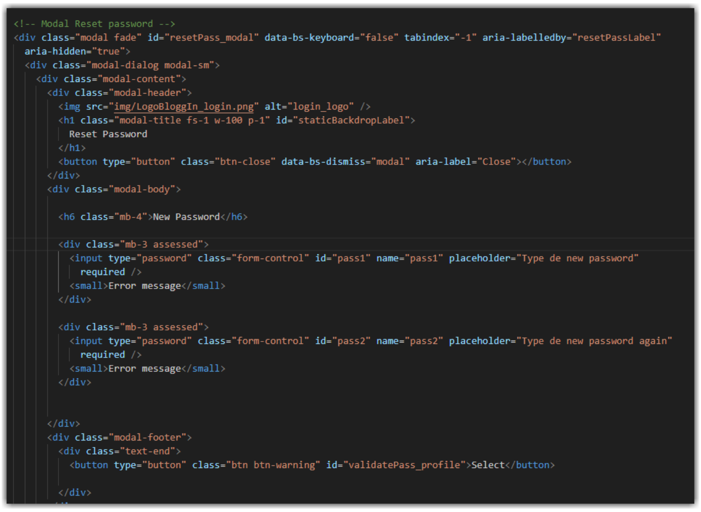
Age: 45

Choose your avatar

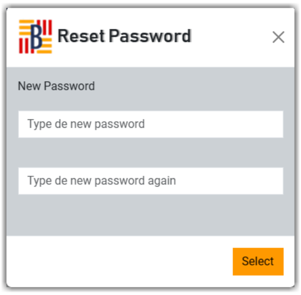
Web Style theme: Light

2. There's an option to change password only.

In profile editing, there is a button for changing the password. When you click on it, it displays a modal for entering a new password. If the password passes the secure password check, it is changed in the database. To change it, use the same function as for changing the profile data.



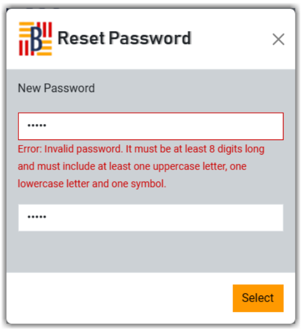
```html
<!-- Modal Reset password -->
<div class="modal fade" id="resetPass_modal" data-bs-keyboard="false" tabindex="-1" aria-labelledby="resetPassLabel"
  aria-hidden="true">
  <div class="modal-dialog modal-sm">
    <div class="modal-content">
      <div class="modal-header">
        <img src="img/LogoBloggIn_login.png" alt="login_logo" />
        <h1 class="modal-title fs-1 w-100 p-1" id="staticBackdropLabel">
          Reset Password
        </h1>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">

        <h6 class="mb-4">New Password</h6>

        <div class="mb-3 assessed">
          <input type="password" class="form-control" id="pass1" name="pass1" placeholder="Type de new password"
            required />
          <small>Error message</small>
        </div>

        <div class="mb-3 assessed">
          <input type="password" class="form-control" id="pass2" name="pass2" placeholder="Type de new password again"
            required />
          <small>Error message</small>
        </div>


      </div>
      <div class="modal-footer">
        <div class="text-end">
          <button type="button" class="btn btn-warning" id="validatePass_profile">Select</button>

        </div>
```



New Password

Type de new password

Type de new password again



New Password

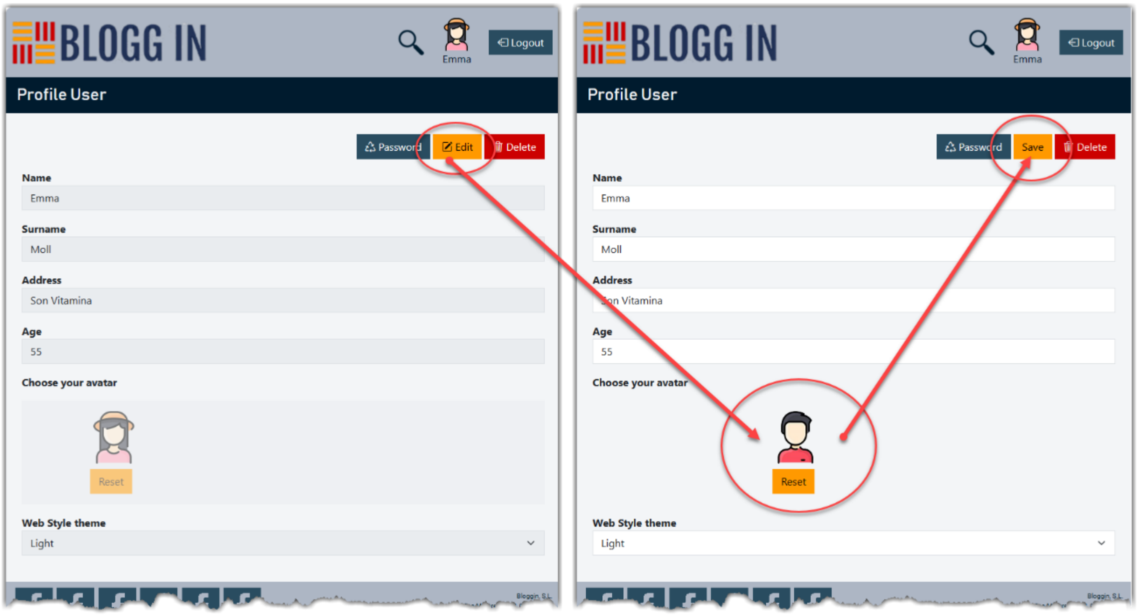
•••••

Error: Invalid password. It must be at least 8 digits long and must include at least one uppercase letter, one lowercase letter and one symbol.

•••••

3. There’s an option to change the avatar.

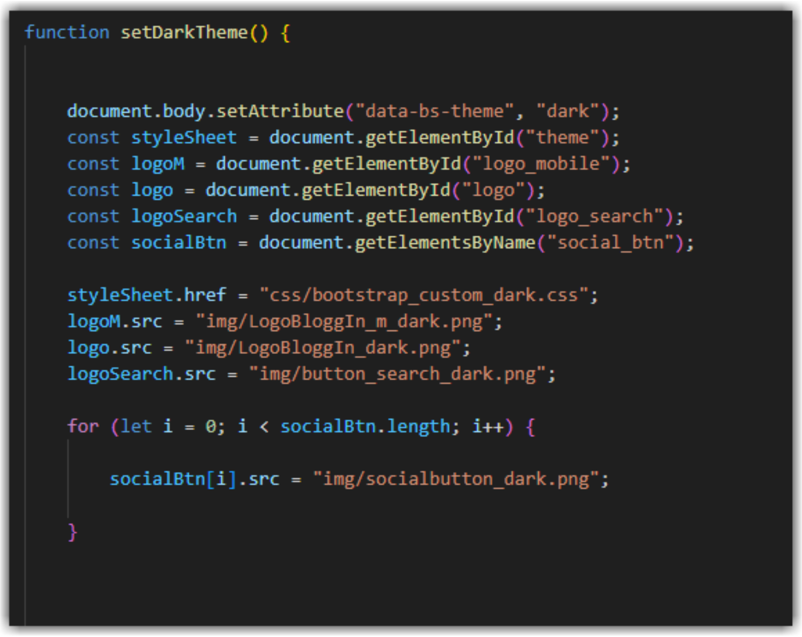
Once on the edit page, you have the option to change the avatar as well as the rest of the data. The same code is used as when you register.



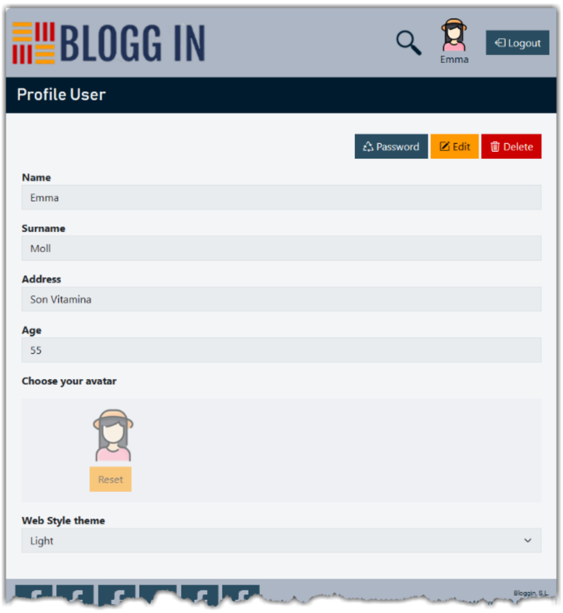
Name
Emma
Surname
Moll
Address
Son Vitamina
Age
55
Choose your avatar
Web Style theme
Light

4. There’s an option to change the theme of the page (light or dark).

In the database, there is a theme attribute in the user data that refers to the active theme: 0 -> light, 1->dark. When verify the user on load page, it triggers the function setDarkTheme() if the attribute is 1 which changes the stylesheet to apply the dark mode.



```
function setDarkTheme() {


    document.body.setAttribute("data-bs-theme", "dark");
    const styleSheet = document.getElementById("theme");
    const logoM = document.getElementById("logo_mobile");
    const logo = document.getElementById("logo");
    const logoSearch = document.getElementById("logo_search");
    const socialBtn = document.getElementsByName("social_btn");

    styleSheet.href = "css/bootstrap_custom_dark.css";
    logoM.src = "img/LogoBloggIn_m_dark.png";
    logo.src = "img/LogoBloggIn_dark.png";
    logoSearch.src = "img/button_search_dark.png";

    for (let i = 0; i < socialBtn.length; i++) {

        socialBtn[i].src = "img/socialbutton_dark.png";

    }
```



Profile User

Name: Emma

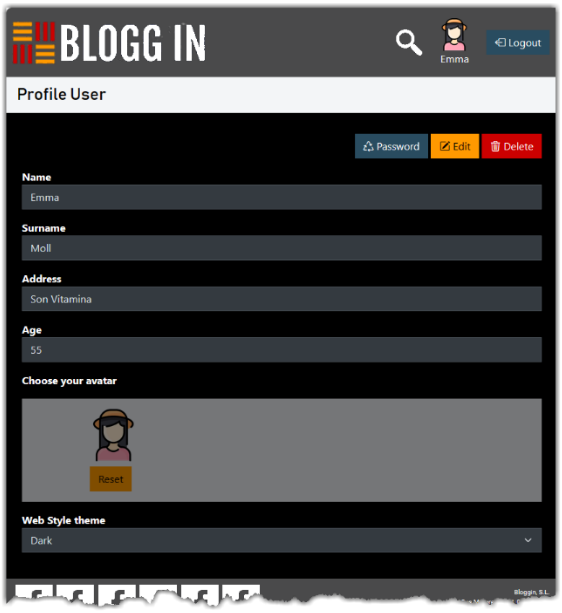
Surname: Moll

Address: Son Vitamina

Age: 55

Choose your avatar

Web Style theme: Light



Profile User

Name: Emma

Surname: Moll

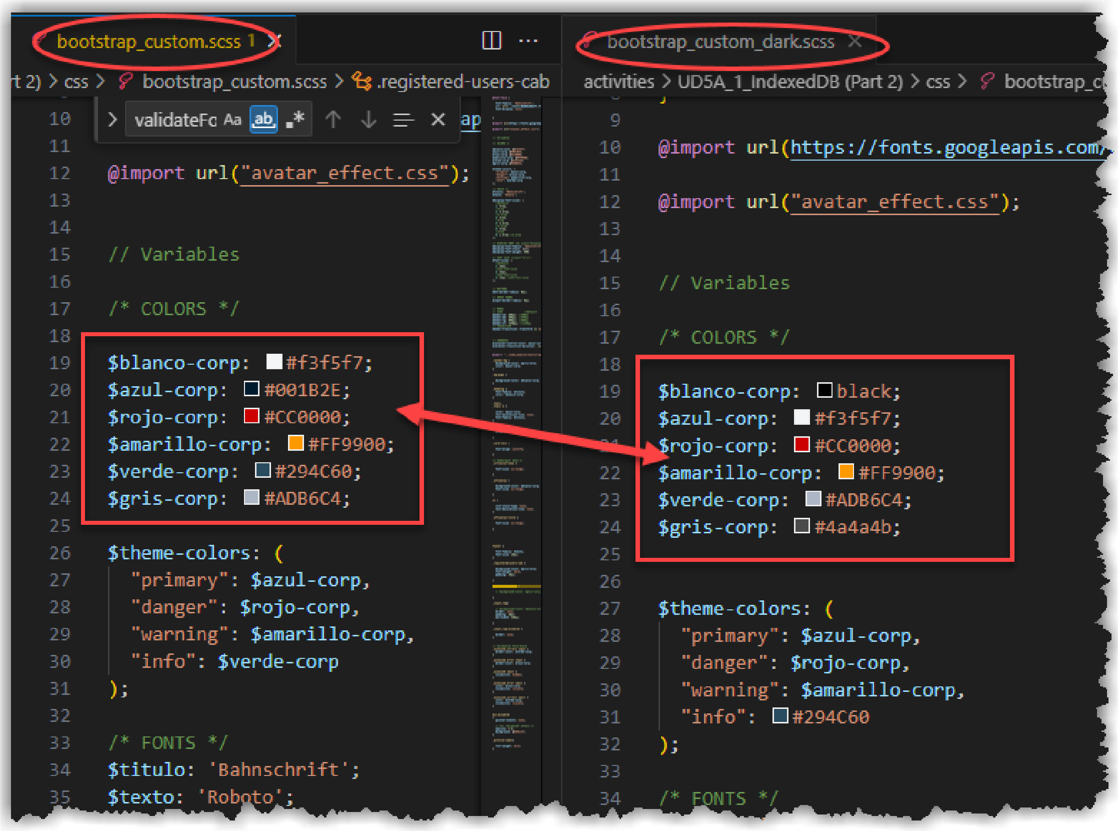
Address: Son Vitamina

Age: 55

Choose your avatar

Web Style theme: Dark

The theme consists of two custom bootstrap files using sass but with the colours reversed.

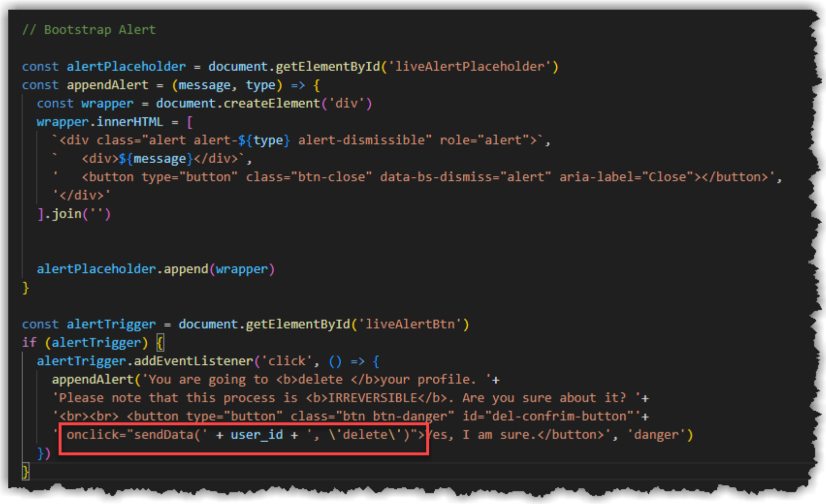

bootstrap_custom.scss 1
t 2) > css > bootstrap_custom.scss > .registered-users-cab
validateFc
bootstrap_custom_dark.scss
activities > UD5A_1_IndexedDB (Part 2) > css > bootstrap_c

```scss
@import url("avatar_effect.css");


// Variables

/* COLORS */

$blanco-corp: #f3f5f7;
$azul-corp: #001B2E;
$rojo-corp: #CC0000;
$amarillo-corp: #FF9900;
$verde-corp: #294C60;
$gris-corp: #ADB6C4;

$theme-colors: (
  "primary": $azul-corp,
  "danger": $rojo-corp,
  "warning": $amarillo-corp,
  "info": $verde-corp
);

/* FONTS */
$titulo: 'Bahnschrift';
$texto: 'Roboto';
```

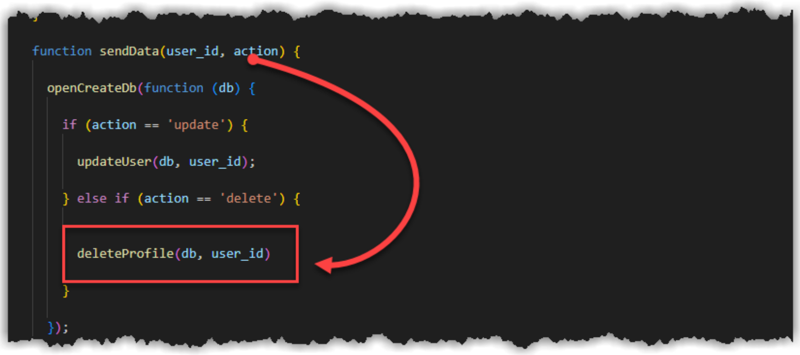
```scss
@import url(https://fonts.googleapis.com/

@import url("avatar_effect.css");


// Variables

/* COLORS */

$blanco-corp: black;
$azul-corp: #f3f5f7;
$rojo-corp: #CC0000;
$amarillo-corp: #FF9900;
$verde-corp: #ADB6C4;
$gris-corp: #4a4a4b;


$theme-colors: (
  "primary": $azul-corp,
  "danger": $rojo-corp,
  "warning": $amarillo-corp,
  "info": #294C60
);

/* FONTS */
```

5. If a user deletes the account, he/she automatically has to log out.

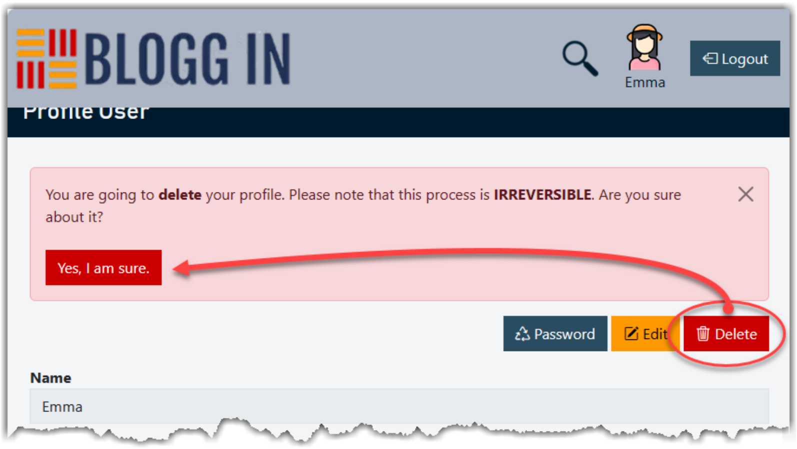
If you click on the 'Delete' button, it displays an alert and confirmation. This uses a Bootstrap class called alert which is intended to display alerts. It is assigned to an 'EventListener' that appears when the button is clicked. In the alert there is a button that confirms the deletion that finally calls the function sendData(db, action) with the value 'delete' of the action attribute to finally delete the record. On deletion, a logout is performed which causes a redirection to the home page.



```
// Bootstrap Alert

const alertPlaceholder = document.getElementById('liveAlertPlaceholder')
const appendAlert = (message, type) => {
  const wrapper = document.createElement('div')
  wrapper.innerHTML = [
    `<div class="alert alert-${type} alert-dismissible" role="alert">`,
    `   <div>${message}</div>`,
    '   <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>',
    '</div>'
  ].join('')


  alertPlaceholder.append(wrapper)
}

const alertTrigger = document.getElementById('liveAlertBtn')
if (alertTrigger) {
  alertTrigger.addEventListener('click', () => {
    appendAlert('You are going to <b>delete </b>your profile. '+
    'Please note that this process is <b>IRREVERSIBLE</b>. Are you sure about it? '+
    '<br><br> <button type="button" class="btn btn-danger" id="del-confrim-button"'+
    ' onclick="sendData(' + user_id + ', \'delete\')">Yes, I am sure.</button>', 'danger')
  })
}
```



```
function sendData(user_id, action) {

  openCreateDb(function (db) {

    if (action == 'update') {

      updateUser(db, user_id);

    } else if (action == 'delete') {

      deleteProfile(db, user_id)

    }

  });
```



You are going to **delete** your profile. Please note that this process is **IRREVERSIBLE**. Are you sure about it?

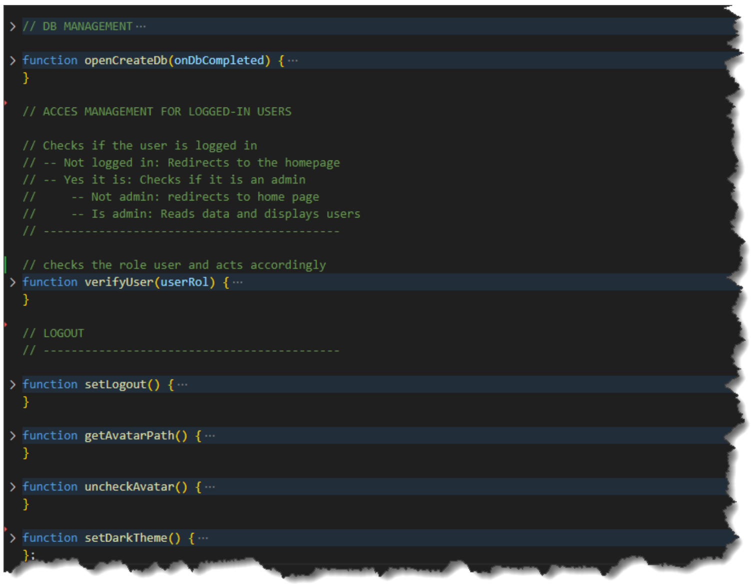
Name

Emma

CODE

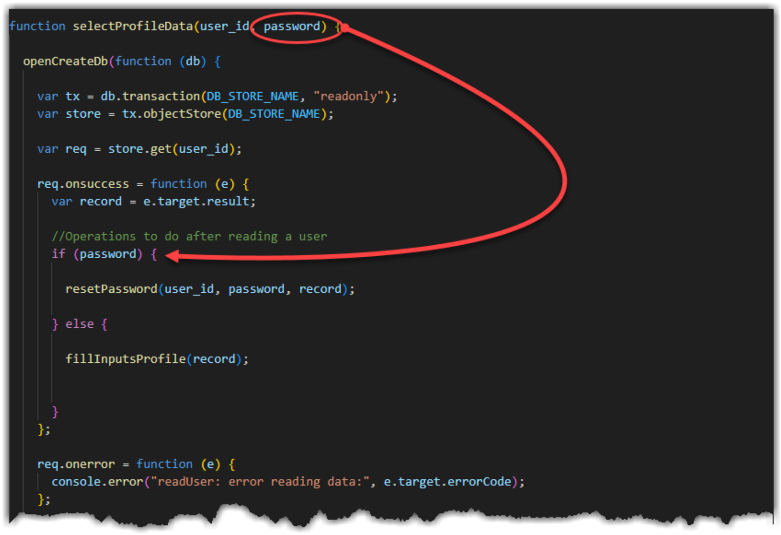
1. Functions have been reused in different parts of the website.

In the common.js file there are functions that are used on all pages. These include that opens or creates the database, that verifies the user, the logout function or the one to apply the dark theme.



```
> // DB MANAGEMENT···

> function openCreateDb(onDbCompleted) {···
  }

  // ACCES MANAGEMENT FOR LOGGED-IN USERS

  // Checks if the user is logged in
  // -- Not logged in: Redirects to the homepage
  // -- Yes it is: Checks if it is an admin
  //     -- Not admin: redirects to home page
  //     -- Is admin: Reads data and displays users
  // ------------------------------------------

  // checks the role user and acts accordingly
> function verifyUser(userRol) {···
  }

  // LOGOUT
  // ------------------------------------------

> function setLogout() {···
  }

> function getAvatarPath() {···
  }

> function uncheckAvatar() {···
  }

> function setDarkTheme() {···
  };
```

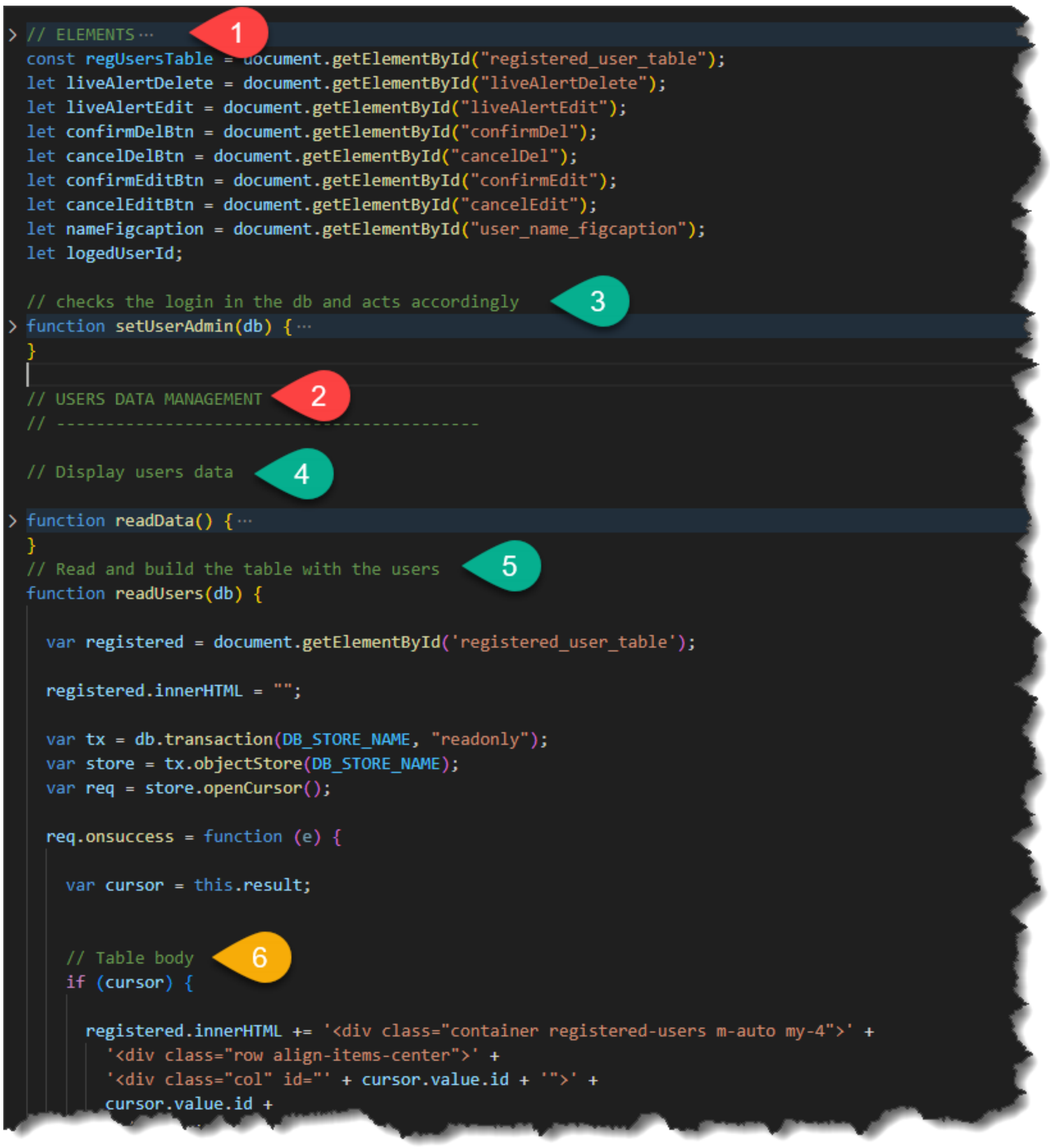
There are also functions that are used for more than one process, for example, one of them displays the user's profile data to be able to modify it, it is also used to change the password, using a parameter in the function that determines it.



```
function selectProfileData(user_id, password) {

  openCreateDb(function (db) {

    var tx = db.transaction(DB_STORE_NAME, "readonly");
    var store = tx.objectStore(DB_STORE_NAME);

    var req = store.get(user_id);

    req.onsuccess = function (e) {
      var record = e.target.result;

      //Operations to do after reading a user
      if (password) {

        resetPassword(user_id, password, record);

      } else {

        fillInputsProfile(record);


      }
    };

    req.onerror = function (e) {
      console.error("readUser: error reading data:", e.target.errorCode);
    };
```

2. The code is well tabulated and commented to explain the different parts of the code.

The code is correctly tabulated and structured. There are comments defining conceptual blocks (elements, database management, etc...), function definitions and important comments.

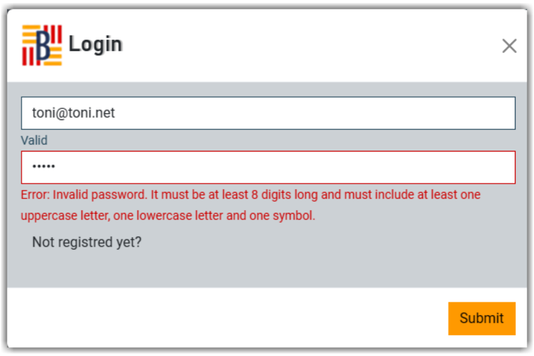


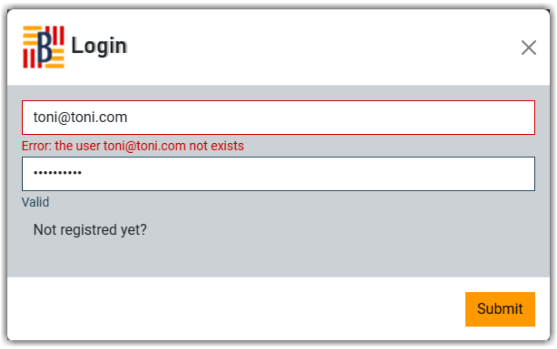
```
> // ELEMENTS ···
  const regUsersTable = document.getElementById("registered_user_table");
  let liveAlertDelete = document.getElementById("liveAlertDelete");
  let liveAlertEdit = document.getElementById("liveAlertEdit");
  let confirmDelBtn = document.getElementById("confirmDel");
  let cancelDelBtn = document.getElementById("cancelDel");
  let confirmEditBtn = document.getElementById("confirmEdit");
  let cancelEditBtn = document.getElementById("cancelEdit");
  let nameFigcaption = document.getElementById("user_name_figcaption");
  let logedUserId;

  // checks the login in the db and acts accordingly
> function setUserAdmin(db) {···
  }

  // USERS DATA MANAGEMENT
  // ------------------------------------------

  // Display users data

> function readData() {···
  }
  // Read and build the table with the users
  function readUsers(db) {

    var registered = document.getElementById('registered_user_table');

    registered.innerHTML = "";

    var tx = db.transaction(DB_STORE_NAME, "readonly");
    var store = tx.objectStore(DB_STORE_NAME);
    var req = store.openCursor();

    req.onsuccess = function (e) {

      var cursor = this.result;


      // Table body
      if (cursor) {

        registered.innerHTML += '<div class="container registered-users m-auto my-4">' +
          '<div class="row align-items-center">' +
          '<div class="col" id="' + cursor.value.id + '">' +
          cursor.value.id +
```

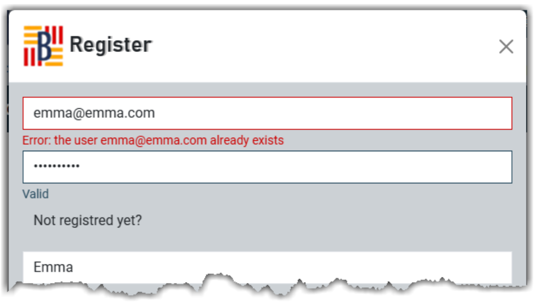
GENERAL ASPECTS

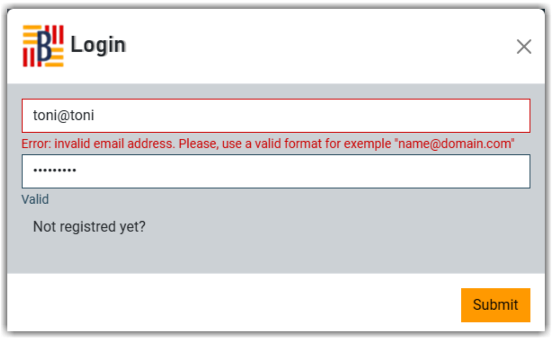
1. Usability (for example: error messages on forms are shown on the web, not only on console.log)

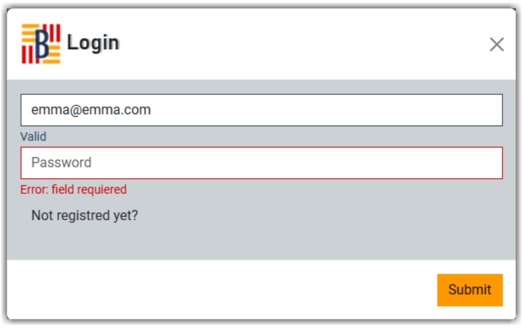
Messages are used in the forms at the bottom of the inputs to help the user in case of error.

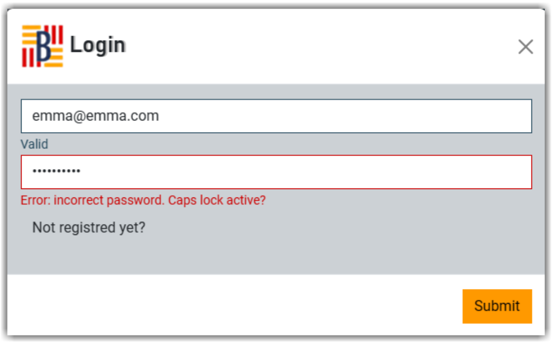

Login
toni@toni.net
Valid
•••••
Error: Invalid password. It must be at least 8 digits long and must include at least one uppercase letter, one lowercase letter and one symbol.
Not registred yet?
Submit


Login
toni@toni.com
Error: the user toni@toni.com not exists
••••••••••
Valid
Not registred yet?
Submit

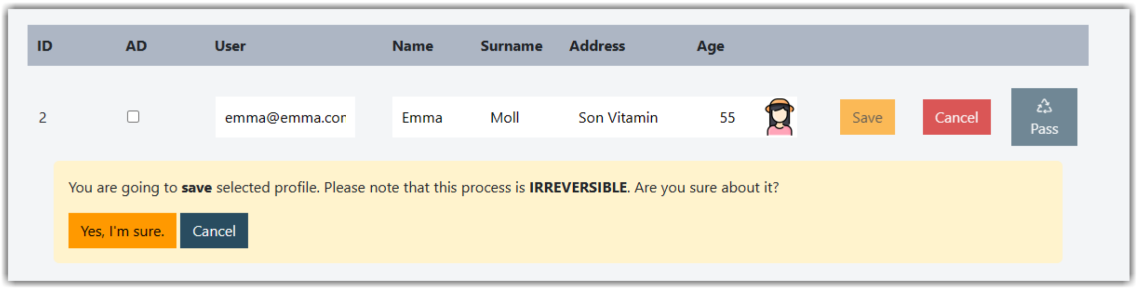

Register
emma@emma.com
Error: the user emma@emma.com already exists
••••••••••
Valid
Not registred yet?
Emma


Login
toni@toni
Error: invalid email address. Please, use a valid format for exemple "name@domain.com"
•••••••••
Valid
Not registred yet?
Submit


Login
emma@emma.com
Valid
Password
Error: field requiered
Not registred yet?
Submit


Login
emma@emma.com
Valid
••••••••••
Error: incorrect password. Caps lock active?
Not registred yet?
Submit

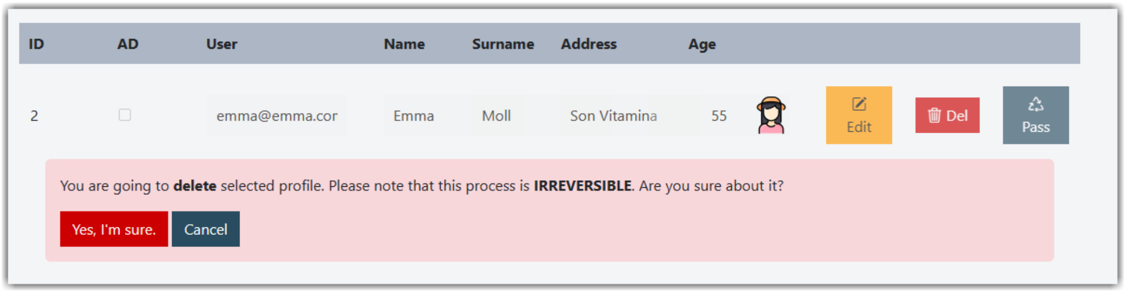
Confirmations are displayed for actions that are not reversible, such as updating user data, deleting a record, etc.



ID	AD	User	Name	Surname	Address	Age				
2	☐	emma@emma.cor	Emma	Moll	Son Vitamin	55		Save	Cancel	Pass

You are going to **save** selected profile. Please note that this process is **IRREVERSIBLE**. Are you sure about it?

Yes, I'm sure. Cancel

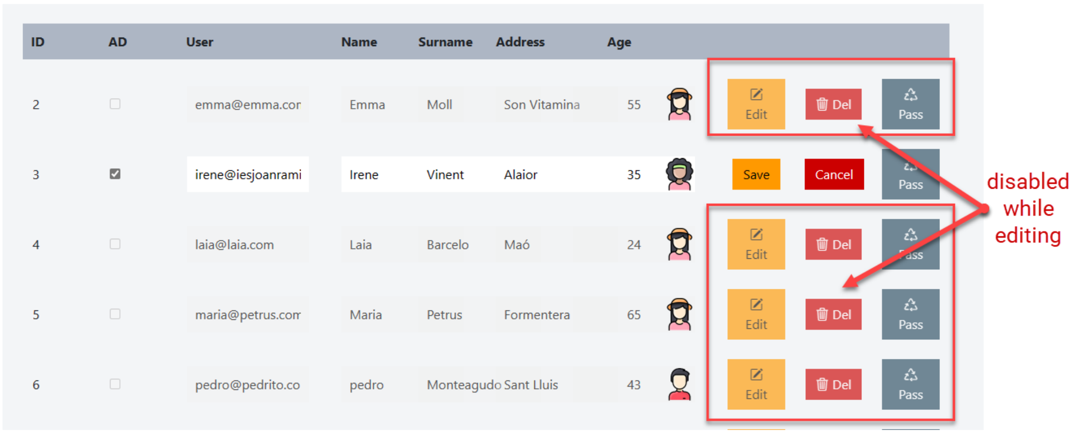


ID	AD	User	Name	Surname	Address	Age				
2	☐	emma@emma.cor	Emma	Moll	Son Vitamina	55		Edit	Del	Pass

You are going to **delete** selected profile. Please note that this process is **IRREVERSIBLE**. Are you sure about it?

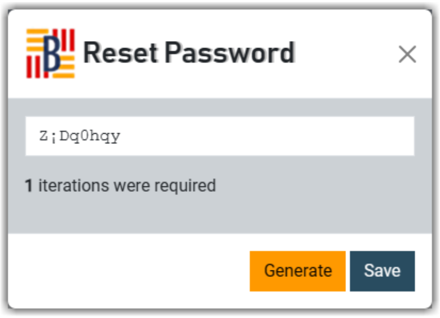
Yes, I'm sure. Cancel

To improve the UX I have considered the management of elements such as buttons when executing an action. For example, if a record is edited, disable the rest of the buttons so that they cannot be pressed.



ID	AD	User	Name	Surname	Address	Age				
2	☐	emma@emma.cor	Emma	Moll	Son Vitamina	55		Edit	Del	Pass
3	☑	irene@iesjoanrami	Irene	Vinent	Alaior	35		Save	Cancel	Pass
4	☐	laia@laia.com	Laia	Barcelo	Maó	24		Edit	Del	Pass
5	☐	maria@petrus.com	Maria	Petrus	Formentera	65		Edit	Del	Pass
6	☐	pedro@pedrito.co	pedro	Monteagudo	Sant Lluis	43		Edit	Del	Pass

Another option for improving the UX is to give the option to automatically generate a secure password when changing the password.



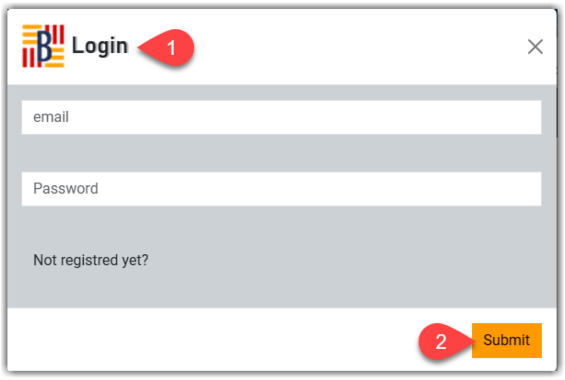
Reset Password

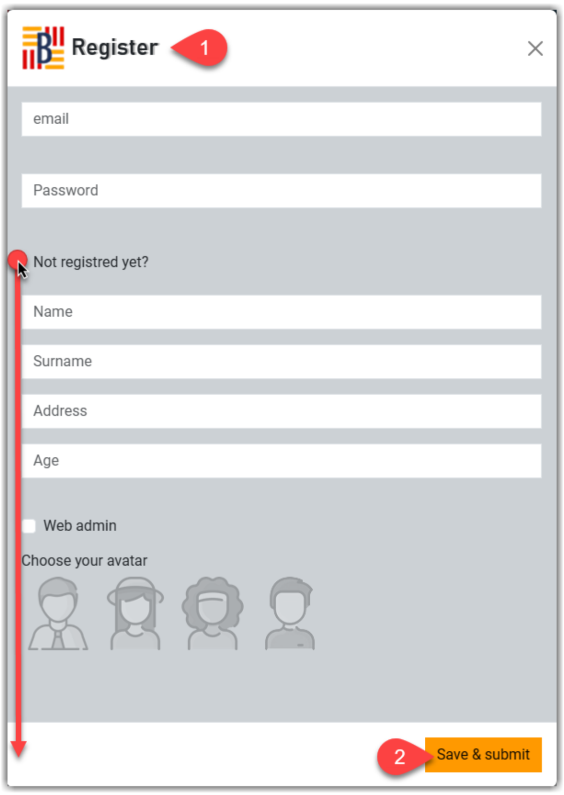
Z¡Dq0hqy

1 iterations were required

Generate Save

Using a single form for both login and registration simplifies the process and usability.





2. The style follows the web design

On all pages and options I have used the style of the website. Buttons in the same style and colour, messages below the inputs, modal for important processes that require exclusive attention, etc.