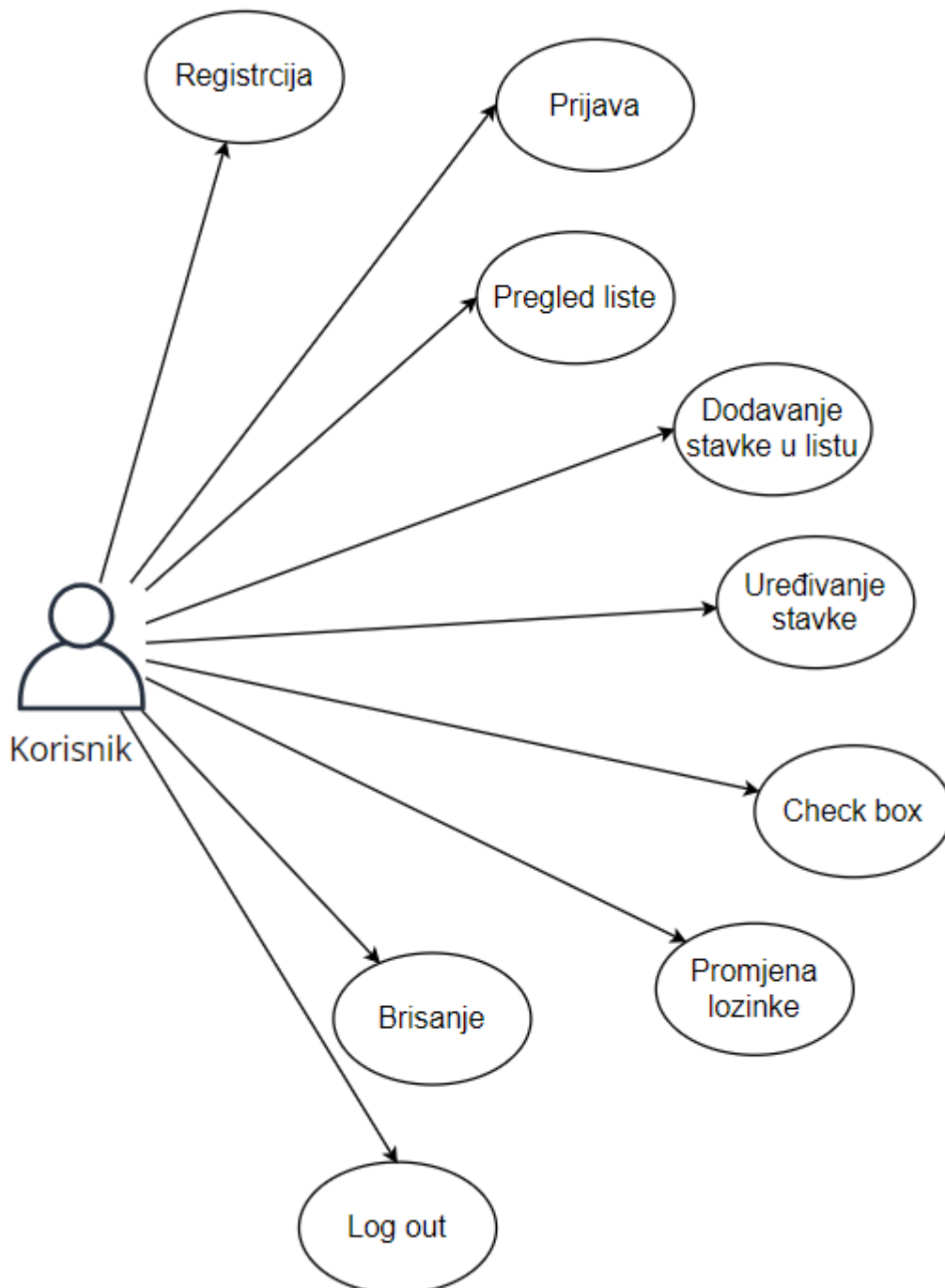


Dokumentacija projekta To-Do

Tim To-Do

UML DIJAGRAM



1 Use case dijagram:

- Aktor: Korisnik
- Use Case-ovi:
 - Dodaj zadatke
 - Označi zadatke kao dovršene
 - Pregledaj popis zadataka
 - Izbriši zadatke
 - Postavi podsjetnike..

2 Class dijagram:

- Klase
 - Task (zadatak)
 - Atributi: id, description, due_date, finished
 - User(Korisnik)
 - Atributi: id,email, username, passw

3 Sequence dijagram:

- Prikazuje interakciju između različitih objekata tijekom izvršenja određene funkcionalnosti
- Primjer: Dodavanje novog zadatka u aplikaciju- korisnik unosi podatke, aplikacija kreira novi zadatak.

4 Activity dijagram:

- Prikazuje tok aktivnosti
- Primjer: Pregled zadatka- korisnik otvara aplikaciju, prikazuje se popis zadataka, korisnik odabire pregled detalja zadataka

5 State Machine dijagram:

- Prikazuje različita stanja u kojima se objekt može nalaziti.
- Primjer: Stanje zadatka može biti dovršeno/nedovršeno.

Funkcionalnosti (JavaScript)

Web stranica se sastoji od 4 glavna dijela: main_menu, sign_up page, log_in page i to_do page. Funkcionalosti su raspoređene unutar istoimenih JavaScript datoteki i funkcije su grupirane po istoimenim HTML file-ovima na koje djeluju.

Jedina stavka koja nema .js je main_menu čije su jedine funkcionalnosti te da se klikom na dugme prebacimo na odgovarajuću sign_up ili log_in stranicu.

Svaka od stranica sadrži i svoje .css varijante, te koristimo i .php za dohvaćanje podataka iz baze.

U sljedećim odjeljcima je detaljniji prikaz funkcionalosti sign_up.js, log_in.js i to_do.js , ali prije svega se prikazuje osnovni izgled baze.

Baza podataka

Baza to_do se sastoji od tablica user i task.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 email	varchar(100)	utf8mb4_croatian_ci		No	None			Change Drop More
<input type="checkbox"/>	2 username	varchar(100)	utf8mb4_croatian_ci		No	None			Change Drop More
<input type="checkbox"/>	3 passw	varchar(100)	utf8mb4_croatian_ci		No	None			Change Drop More

user ima za ključ email, te svojstva passw i username koja su obavezna.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 task_id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2 description	varchar(1000)	utf8mb4_croatian_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	3 due_date	date			No	None			Change Drop More
<input type="checkbox"/>	4 finished	tinyint(1)			Yes	NULL			Change Drop More
<input type="checkbox"/>	5 user_id	varchar(100)	utf8mb4_croatian_ci		No	None			Change Drop More

task ima task_id za ključ, te svojstva description, due_date, finished (0 ili 1) i strani ključ user_id koji je zapravo email tablice user (označava čiji je task).

Sign-up page (sign_up.js)

```
<body>
  <div class="container">

    <h1>Sign-In</h1>

    <form name="prijava" onsubmit="attemptSignUp(event)">

      <div class="input">

        <label for="username">Username</b></label>
        <input type="text" id="username" name="username" required>

        <label for="email">E-mail</label>
        <input type="text" id="email" name="email" required>

        <label for="password">Password</label>
        <input type="password" id="password" name="password" required>

      </div>

      <input type="submit" name="submit_signup" value="Sign Up">

    </form>
    <div id="error"></div>

  </div>
</body>
```

```
function isValidEmail(email) {
  |   return /^[^@]+@[^@]+\.[^@]+$/.test(email);
}
}
```

isValidEmail(email) - funkcija provjerava je li email napisat u dobrom formatu uz pomoć regex izraza (koristi se kao pomoćna)

```
function displayError(message) {
  |   document.getElementById('error').innerHTML = message;
}
}
```

displayError(message) - u div s id-jem "error" ispisujemo poruku u slučaju krivog unosa (koristi se kao pomoćna funkcija)

```
function attemptSignUp(event) {
    event.preventDefault();

    const email = document.getElementById("email").value;
    const password = document.getElementById("password").value;
    const username = document.getElementById("username").value;

    if (!isValidEmail(email)) {
        displayError("Napišite mail u točnom formatu.");
        return;
    }

    sendSignUpRequest(email, password, username);
}
```

attemptSignUp(event) - Funkcija prima event koji nam govori kada smo kliknuli submit dugme tj. nakon unosa podataka u input polja, ova funkcija provjerava možemo li obaviti sign up. Dohvaća odgovarajuće podatke email, password, username is pripadne forme, te se s funkcijom isValidEmail() provjerava valjanost unesenog mail-a (ako mail nije u dobrom formatu poziva se funkcija koja javlja grešku displayError("naša poruka"). Ako je mail u dobrom formatu, podaci se proslijede funkciji sendSignUpRequest() koja sprema podatke u bazu. Preko ove funkcije se odvijaju sve radnje sign_up.js -a.

```
function handleSignUpResponse(status, response, email) {
    if (status == 200) {
        if (response === "success") {
            localStorage.setItem("email", email);
            window.location.href = "to_do.html";
        } else {
            displayError("Email se koristi.");
        }
    } else {
        alert("Error: " + status);
    }
}
```

handleSignUpResponse(status, response, email) - Ako je status 200, uspješno je obavljeno spajanje na bazu, te u slučaju da su podaci bili dopustivi, spremamo lokalnu varijablu email kako bi prebacili korisnika na novu stranicu, te potom se fizički prebacujemo na to_do.html. Ako format nije bio dobar, displayError() nam to prikaže.

```
function sendSignUpRequest(email, password, username) {
    const xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4) {
            handleSignUpResponse(xhr.status, xhr.responseText, email);
        }
    };

    xhr.open("POST", "../php/sign_up.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.send("email=" + encodeURIComponent(email) + "&passw=" + encodeURIComponent(password) + "&username=" + encodeURIComponent(username));
}
```

sendSignUpRequest(email, password, username) - Funkcija prima podatke o korisniku, te stvara novi XML Http Request xhr. xhr nam služi kako bi se mogli asinkrono spojiti na bazu tj. stvoriti HTTP POST zahtjev za server-side skriptu tj. .php file (želimo da stranica nastavi s radom i nakon šta se spojimo na bazu). Stvaramo evenhandler za xhr koji čeka da status postane 4 tj. čeka da zahtjev bude dovršen. Tad pozivamo funkciju handleSignUpResponse() koja nam javlja je li zapisano u bazu ili nije.

xhr.open() nam govori kakvu metodu koristimo i da radimo sa sign_up.php -om.
 xhr.setRequestHeader() nam govori u kojem obliku je poslani data - urlencoded.
 xhr.send() šalje naše podatke u sign_up.php gdje će se zapisati u bazu.

Ispod je prikazan kod za sign_up.php.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {

    $email = $_POST["email"];
    $password = $_POST["passw"];
    $username = $_POST["username"];

    $con = mysqli_connect("localhost", "root", "", "to_do");

    $upit = "SELECT * FROM user WHERE email = ?";

    $stmt = mysqli_prepare($con, $upit);

    // bind parametara
    mysqli_stmt_bind_param($stmt, "s", $email);

    // izvršimo $stmt
    mysqli_stmt_execute($stmt);

    $result = mysqli_stmt_get_result($stmt);

    // je li vraćen red
    if (mysqli_num_rows($result) === 1) {
        echo "failure";
        mysqli_stmt_close($stmt);
    } else {
        $upit_unos = "INSERT INTO user (email, username, passw) VALUES (?, ?, ?)";

        $stmt_unos = mysqli_prepare($con, $upit_unos);

        mysqli_stmt_bind_param($stmt_unos, "sss", $email, $username, $password);

        mysqli_stmt_execute($stmt_unos);

        mysqli_stmt_close($stmt_unos);
        echo "success";
    }
    mysqli_close($con);
}
```

Provjerava se je li metoda POST (je i ova se metoda općenito koristi kroz projekt). Stvaramo varijable sa primljenim parametrima. Varijabla \$con name predstavlja spajanje na bazu. U varijablu \$upit spremamo SQL query koji će dohvatiti sve korisnike s primljenim mailom. Pripremamo statement i bind-amo naš parametar \$email za proveru mail-a u upitu, te izvršimo statement i gledamo rezultat. Ako već postoji korisnik s ovim mailom, php vraća da spremanje nije valjano i zatvara statement, a ako ne postoji, stvaramo novi upit kojim ubacujemo novi podatak u bazu. Proces stvaranja statementa je analogan prethodnom. Na kraju moramo zatvoriti bazu.

Log-in page (log_in.js)

```
<body>
  <div class="container">

    <h1>Prijava</h1>

    <form name="prijava" onsubmit="attemptLogin(event)">
      <div class="input">

        <label for="email">E-mail</b></label>
        <input type="text" id="email" name="email" required>

        <label for="password">Lozinka</label>
        <input type="password" id="password" name="password" required>

      </div>

      <input type="submit" name="submit_login" value="Login">

    </form>
    <div id="error"></div>

  </div>
</body>
```

```
function attemptLogin(event) {
  event.preventDefault();

  var email = document.getElementById("email").value;
  var password = document.getElementById("password").value;

  // AJAX za komunikaciju php i js

  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      if (xhr.status == 200) {
        var response = xhr.responseText;
        console.log(response);

        if (response === "success") {
          localStorage.setItem("email", email);
          window.location.href = "to_do.html";
        } else {
          document.getElementById('error').innerHTML = "Netočan mail ili lozinka";
        }
      } else {
        alert("Error: " + xhr.status);
      }
    }
  };

  xhr.open("POST", "../php/log_in.php", true);
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhr.send("email=" + encodeURIComponent(email) + "&password=" + encodeURIComponent(password));
}
```

attemptLogin(event) - Funkcija, kao i attemptSignUp() prima događaj koji nam javlja kad smo submit-ali log-in podatke - email i password. Stvaranje XML Http Request-a je analogno, pa ćemo od sada samo komentirati specifične za ovu funkciju. U bazu se šalju

upisani email i lozinka, te se u slučaju uspješnog unosa u bazu prebacujemo na to_do.html nakon što spremimo varijablu email da znamo o kojem korisniku je riječ. Ako unos u bazu nije dobar, to se ispisuje na ekranu.

Ispod je prikazan log_in.php koji se poziva za povezivanje s bazom u prošloj funkciji.

```
<?php

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $email = $_POST["email"];
    $password = $_POST["password"];

    $con = mysqli_connect("localhost", "root", "", "to_do");

    $upit = "SELECT * FROM user WHERE email = ? AND passw = ?";

    $stmt = mysqli_prepare($con, $upit);

    // bind parametara
    mysqli_stmt_bind_param($stmt, "ss", $email, $password);

    // izvršimo $stmt
    mysqli_stmt_execute($stmt);

    $result = mysqli_stmt_get_result($stmt);

    // je li vraćen red
    if (mysqli_num_rows($result) === 1) {
        echo "success";
    } else {
        echo "failure";
    }

    // moramo sve zatvoriti
    mysqli_stmt_close($stmt);
    mysqli_close($con);
}

?>
```

Postupak povezivanja je analogan kao prije. Upitom provjeravamo postoji li korisnik u bazi i ako postoji vraćamo "success" i zatvaramo bazu.

To-Do page (to_do.js)

```
<div class="container">
  <header class="header">
    <form method="post">
      <h2 id="h2_hello">Bok </h2>
    </form>
    <button onclick="Logout()" name="log_out">LOG OUT</button>
  </header>

  <div class="task_list">

    <div id="task">

    </div>

    <div id="task_input">
      <input type="text" id="description_input" name="description_input">
      <input type="date" id="date_input" name="date_input">
      <button type="submit" name="add_task" onclick="AddTask()">Dodaj</button>
    </div>

  </div>

  <div class = "search_div">

    <input type="text" id="search_input" name="search_input">
    <button name="search_task" onclick="SearchTasks()">Traži</button>
    <button type="submit" name="refresh" onclick="Refresh()">Refresh tasks</button>

  </div>

  <footer class="footer">
    <button name="change_pass" onclick="ChangePassword()">Nova Lozinka?</button>
    <input type="text" id="new_pass" name="new_pass">
    <div id="error-pass"></div>
  </footer>
</div>
```

```
var email = "";
var user = "none";

function GetUser() {
  email = localStorage.getItem("email");
  GetUsername();
  GetTasks();
}
```

Stvaramo varijable email i user koje će nam pamtit i trenutno ulogiranog korisnika.

GetUser() - Jedina funkcija koja se automatski izvršava pozivanjem na kraju skripte to_do.js. Ona je potrebna za dohvatiti korisnika. U na početku definiranu varijablu email sprema varijablu email koju smo spremili u storage prilikom sign-up/log-in postupka. Potom

poziva funkciju `GetUsername()` koja sprema username korisnika sa spremljenim mailom, te `GetTasks()` koja dohvaća taskove tog korisnika i prikazuje ih.

```
function GetUsername() {  
  
    var xhr = new XMLHttpRequest();  
    xhr.onreadystatechange = function () {  
        if (xhr.readyState == 4) {  
            if (xhr.status == 200) {  
  
                var response = xhr.responseText;  
                console.log(response);  
  
                Obrada_Username(response);  
  
            } else {  
                alert("Error: " + xhr.status);  
            }  
        }  
    };  
  
    xhr.open("POST", "../php/to_do.php", true);  
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
    xhr.send("email=" + encodeURIComponent(email));  
}  
  
function Obrada_Username(response){  
  
    document.getElementById('h2_hello').innerHTML += response + "!";  
    user = response;  
}
```

GetUsername() - Spaja se na bazu putem `to_do.php` i šalje joj email. Ako vrati korisničko ime, poziva se funkcija `Obrada_Username(response)` koja sprema vraćeni username u varijablu `user` i ispisuje se u heading-u pozdrav korisniku.

Ispod je prikazan `to_do.php`.

```
to_do.php  
<?php  
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    $email = $_POST["email"];  
  
    $con = mysqli_connect("localhost", "root", "", "to_do");  
  
    $upit = "SELECT username FROM user WHERE email = ?";  
  
    $stmt = mysqli_prepare($con, $upit);  
  
    mysqli_stmt_bind_param($stmt, "s", $email);  
  
    mysqli_stmt_execute($stmt);  
  
    mysqli_stmt_bind_result($stmt, $username);  
  
    mysqli_stmt_fetch($stmt);  
  
    echo $username;  
  
    mysqli_stmt_close($stmt);  
    mysqli_close($con);  
}
```

```
function GetTasks(){
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4) {
            if (xhr.status == 200) {
                Obrada_Tasks(xhr.responseText);
            } else {
                alert("Error: " + xhr.status);
            }
        }
    };

    xhr.open("POST", "../php/get_tasks.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.send("email=" + encodeURIComponent(email));
}
```

GetTasks() - U bazi dohvatimo listu taskova korisnika i pozivamo Obrada_Tasks() koja ih prikazuje na ekranu (pozivamo get_tasks.php).

Ispod je prikazan get_tasks.php.

```
get_tasks.php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    ... $email = $_POST["email"];

    $con = mysqli_connect("localhost", "root", "", "to_do");

    $upit = "SELECT * FROM task WHERE user_id = ?";

    $stmt = mysqli_prepare($con, $upit);

    mysqli_stmt_bind_param($stmt, "s", $email);

    mysqli_stmt_execute($stmt);

    $result = mysqli_stmt_get_result($stmt);

    $rows = array();
    while ($row = mysqli_fetch_assoc($result)) {
        $rows[] = $row;
    }

    echo json_encode($rows);

    mysqli_close($con);
}
?>
```

Vraćene retke (taskove) spremamo u niz, te ga pretvoriti u JSON format radi lakšeg rada s podacima.

```

function Obrada_Tasks(response) {

    document.getElementById('task').innerHTML = "";

    var jsonResponse = JSON.parse(response);

    for (var i = 0; i < jsonResponse.length; i++) {
        var taskContainer = document.getElementById('task');
        var taskDiv = document.createElement('div');
        taskDiv.id = 'task_' + jsonResponse[i].task_id;

        // je li task uskoro
        var dueDate = new Date(jsonResponse[i].due_date);
        var currentDate = new Date();
        var timeDifference = dueDate - currentDate;
        var daysDifference = Math.floor(timeDifference / (1000 * 60 * 60 * 24));

        taskDiv.innerHTML = jsonResponse[i].description + " " + jsonResponse[i].due_date + " " +
            "<input type='submit' class='finish_submit' name='finish_submit' onclick='ChangeFinishedStatus(this)' value='Finished'> " +
            "<input type='submit' onclick='DeleteTask(this)' name='delete_task' value='X'>";

        taskContainer.appendChild(taskDiv);

        var finishButton = taskDiv.querySelector('.finish_submit');

        if (jsonResponse[i].finished == 1) {
            finishButton.style.backgroundColor = 'green';
        }
        else{
            finishButton.style.backgroundColor = 'red';
        }

        // Ako u roku od 3 dana, obavijesti da rok uskoro
        if (daysDifference <= 3) {
            taskDiv.innerHTML+= "<b> USKORO!!! </b>";
        }
    }
}

```

ObradaTasks(response) - Taskovi će se prikazati u prethodno kreiranom praznom div-u "task". Na početku stavljamo da je div prazan kako ne bi ponovnim povezivanjem na bazu se dodavalo duplikate. Parsiramo dobiveni JSON string u objekte - taskove. Za svaki task stvaramo novi div s id-jom "task_{task_id iz baze}" kako bi točno razaznali o kojem tasku je riječ, te unutar div-a iz JSON objekta dohvaćamo opis task-a, datum, te stvaramo submit dugme "finish_submit" s kojim označujemo je li task dovršen ili ne pozivom funkcije ChangeFinishedStatus() (crven ako ne, zelen ako da), te imamo "delete_task" submit dugme kojim brišemo taskove pozivom funkcije DeleteTask(). Provjeravamo i koliko je dana između današnjeg datuma i datuma task-a, te ukoliko je u roku od 3 dana, stavljamo poruku da je uskoro.

```

function AddTask() {
    var new_desc = document.getElementById('description_input').value;
    var new_date = document.getElementById('date_input').value;

    var xhr = new XMLHttpRequest();
    xhr.open("POST", "../php/add_task.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

    // tek nakon sta završi request pozvat osvježenje
    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {
            GetTasks();
        }
    };

    xhr.send("description=" + encodeURIComponent(new_desc) + "&due_date=" + encodeURIComponent(new_date) + "&email=" + encodeURIComponent(email));
}

```

AddTask() - Povezuje se na bazu pomoću add_task.php, te tu sprema nove unesene podatke iz polja description_input i date_input. Nakon završetka zahtjeva ponovno ispisujemo sve taskove s GetTasks().

Ispod je prikazan add_task.php.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {

    $desc = $_POST["description"];
    $date = $_POST["due_date"];
    $email = $_POST["email"];
    $finished = 0;

    $con = mysqli_connect("localhost", "root", "", "to_do");

    $insert_query = "INSERT INTO task (description, due_date, finished, user_id) VALUES (?, ?, ?, ?)";

    $stmt_insert = mysqli_prepare($con, $insert_query);
    mysqli_stmt_bind_param($stmt_insert, "ssis", $desc, $date, $finished, $email);

    if (mysqli_stmt_execute($stmt_insert)) {
        echo "New task added successfully.";
    } else {
        echo "Error: " . mysqli_error($con);
    }

    mysqli_stmt_close($stmt_insert);
    mysqli_close($con);
}
?>
```

```
function DeleteTask(button){

    let task_id = button.parentNode.id;
    let task_id_to_send = extractNumberFromString(task_id);

    var xhr = new XMLHttpRequest();

    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {

            document.getElementById(task_id).remove();

        }
    };

    xhr.open("POST", "../php/delete_task.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.send("task_id=" + encodeURIComponent(task_id_to_send));

}
```

DeleteTask(button) - Provjerava jedinstven id div-a pojedinog taska definiranog ranije, te iz njega izvlači task_id sa funkcijom extractNumberFromString(). S task_id nalazimo odgovarajući task u bazi (delete_task.php) i brišemo ga. Uklanjam task sa spomenutim div-om.

Ispod je prikazan delete_task.php.

```

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {

    $task_id = $_POST["task_id"];

    $con = mysqli_connect("localhost", "root", "", "to_do");

    $insert_query = "DELETE FROM task WHERE task_id = ?";

    $stmt = mysqli_prepare($con, $insert_query);
    mysqli_stmt_bind_param($stmt, "i", $task_id);
    mysqli_stmt_execute($stmt);

    mysqli_stmt_close($stmt);
    mysqli_close($con);
}
?>

```

```

function ChangeFinishedStatus(button){

    let task_id = button.parentNode.id;
    let task_id_to_send = extractNumberFromString(task_id);

    var xhr = new XMLHttpRequest();

    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {

            var response = xhr.responseText;

            var finishButton = document.getElementById(task_id).querySelector('.finish_submit');

            if (response == 1) {
                finishButton.style.backgroundColor = 'green';
            } else {
                finishButton.style.backgroundColor = 'red';
            }
        }
    };

    xhr.open("POST", "../php/finish_task.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.send("task_id=" + encodeURIComponent(task_id_to_send));
}

```

ChangeFinishedStatus(button) - Kao i funkcija DeleteTask, vraća div odabranog taska i izvlači njegov id, te ovisno o svojstvu task.finished boji dugme u zeleno ili crveno (finish_task.php).

Ispod je prikazan finished_task.php.

```

finish_task.php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {

    $task_id = $_POST["task_id"];

    $con = mysqli_connect("localhost", "root", "", "to_do");

    //dohvacanje staroga
    $query = "SELECT finished FROM task WHERE task_id = ?";
    $stmt = mysqli_prepare($con, $query);
    mysqli_stmt_bind_param($stmt, "i", $task_id);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_bind_result($stmt, $current_finished);
    mysqli_stmt_fetch($stmt);
    mysqli_stmt_close($stmt);

    // Update na novi
    $new_finished = $current_finished ? 0 : 1;
    $query = "UPDATE task SET finished = ? WHERE task_id = ?";
    $stmt = mysqli_prepare($con, $query);
    mysqli_stmt_bind_param($stmt, "ii", $new_finished, $task_id);
    mysqli_stmt_execute($stmt);

    echo $new_finished;

    mysqli_stmt_close($stmt);
    mysqli_close($con);
}
?>

```

```

function ChangePassword(){

    let password = document.getElementById("new_pass").value;

    console.log(password);

    if(password==""){
        document.getElementById("error-pass").innerHTML="Enter new password";
        return;
    }

    var xhr = new XMLHttpRequest();

    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {

            document.getElementById("error-pass").innerHTML="Password changed.";

        }
    };

    xhr.open("POST", "../php/change_password.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.send("email=" + encodeURIComponent(email) + "&pass=" + encodeURIComponent(password));

}

```

ChangePassword() - Spaja se na bazu (change_password) i mijenja korisnikovu lozinku. Ako ništa nije uneseno, javlja se da se unese nova lozinka.

Ispod je change_password.php.

```

change_password.php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {

    $email = $_POST["email"];
    $new_pass = $_POST["passw"];

    $con = mysqli_connect("localhost", "root", "", "to_do");

    $query = "UPDATE user SET passw = ? WHERE email = ?";
    $stmt = mysqli_prepare($con, $query);
    mysqli_stmt_bind_param($stmt, "ss", $new_pass, $email);
    mysqli_stmt_execute($stmt);

    mysqli_stmt_close($stmt);
    mysqli_close($con);
}
?>

```

```

function SearchTasks(){

    let search_value = document.getElementById("search_input").value;

    var xhr = new XMLHttpRequest();

    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {

            console.log("Tu smo.");
            console.log(JSON.parse(xhr.responseText));
            Obrada_Tasks(xhr.responseText);

        }
    };

    xhr.open("POST", "../php/search_task.php", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.send("search=" + encodeURIComponent(search_value) + "&email=" + encodeURIComponent(email));

}

```

SearchTasks() - Uzima vrijednost unesenu u "search_input" polje nakon klika na dugme "search_task", te vraća taskove u JSON formatu iz baze koji imaju unesenu "ključnu riječ". Poziva Obrada_Tasks() te ih prikazuje na ekranu.

Ispod se nalazi search_task.php.


```

search_task.php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {

    $search = "%" . $_POST["search"] . "%";
    $email = $_POST["email"];

    $con = mysqli_connect("localhost", "root", "", "to_do");

    $upit = "SELECT * FROM task WHERE user_id = ? AND description LIKE ?";

    $stmt = mysqli_prepare($con, $upit);

    mysqli_stmt_bind_param($stmt, "ss", $email, $search);

    mysqli_stmt_execute($stmt);

    $result = mysqli_stmt_get_result($stmt);

    $rows = array();
    while ($row = mysqli_fetch_assoc($result)) {
        $rows[] = $row;
    }

    echo json_encode($rows);

    mysqli_close($con);
}
?>

```

```

function Refresh(){
    GetTasks();
}

function extractNumberFromString(str) {
    const matches = str.match(/\d+/);

    if (matches) {
        const number = parseInt(matches[0], 10);
        return number;
    } else {
        return null;
    }
}

function LogOut(){
    console.log("Logging out...");
    email="";
    localStorage.setItem("email", email);
    window.location.href = "main_menu.html";
}

```

Preostaju 3 manje funkcije.

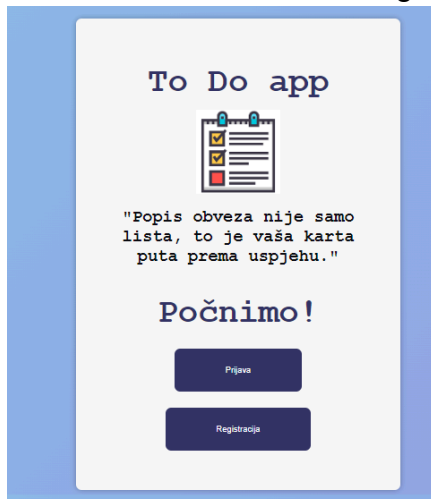
Refresh() - Klikom na submit dugme "refresh" poziva GetTasks(), te ponovno ispisuje sve taskove - ovo korisno nakon search-a.

extractNumberFromString(str) - Pomoćna funkcija koja vraća brojeve u stringu (koristili za div pojedinog task-a).

LogOut() - Zaboravimo email, te se vraćamo na main menu.

Upute za korištenje

1. MAIN MENU: Otvorite web stranicu aplikacije i odaberite botun registracija za registraciju ili ako već imate račun odaberite log in za prijavu u aplikaciju.



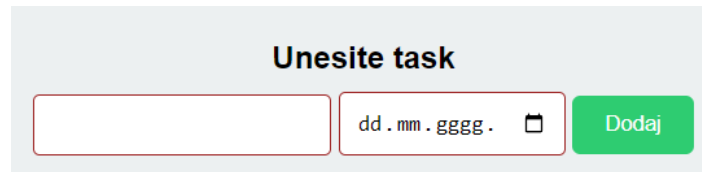
2. REGISTRACIJA: nakon što odaberete registraciju, preusmjereni ste na stranicu s registracijom. Tu unesite vaše korisničko ime, e-mail i lozinku, zatim kliknite na Sign up” botun. Nakon uspješne registracije, bit ćete preusmjereni na glavnu stranicu.

The screenshot shows the registration form titled 'Registracija' in a blue serif font. It contains three input fields: 'Korisničko ime' (Username) with a person icon, 'E-mail' with an envelope icon, and 'Lozinka' (Password) with a lock icon. Below the fields is a dark blue button labeled 'Sign Up'.

3. PRIJAVA: Ako već imate račun i prethodno ste se registrirali, kliknete na log in i upišite podatke koje ste unijeli pri registraciji.

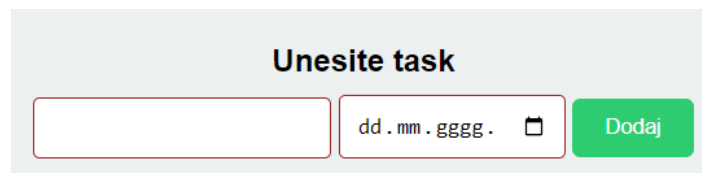
The screenshot shows the login form titled 'Prijava' in a blue serif font. It contains two input fields: 'E-mail' with a person icon and 'Lozinka' (Password) with a lock icon. Below the fields is a dark blue button labeled 'Login'.

4. DODAVANJE ZADATAKA: Nalazite se na glavnoj stranici. pronađite polje za dodavanje zadatka. Tu unesite zadatak koji trebate obaviti, te možete odabrati datum.



The form is titled "Unesite task". It contains a text input field for the task description, a date input field with a placeholder "dd . mm . gggg ." and a calendar icon, and a green "Dodaj" button.

5. SPREMANJE ZADATAKA: Nakon što ste unijeli svoj zadatak u polje za dodavanje zadatka, pronađite botun "Dodaj". Kliknite na taj gumb kako biste spremili novi zadatak.



The form is titled "Unesite task". It contains a text input field for the task description, a date input field with a placeholder "dd . mm . gggg ." and a calendar icon, and a green "Dodaj" button.

6. PRETRAGA ZADATAKA: Ako ste unijeli više zadataka i želite pronaći neki od njih, to možete napraviti pomoću botuna "pretraga". Nakon što unesete zadatak koji tražite u polje koje se nalazi pored botuna pretraga, kliknite na botun "pretraga". i pronaći ćete zadatak koji ste tražili. Klikom na Refresh se ponovno prikažu svi taskovi.



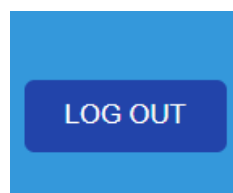
A search bar with a text input field and a blue "Traži" button.

7. PROMJENA LOZINKE: Ako želite, možete promijeniti lozinku. To ćete napraviti tako što ćete unijeti novu lozinku u polje koje se nalazi pokraj botuna "Nova lozinka". Nakon što unesete novu lozinku i kliknete na botun nova lozinka, vaša lozinka je promijenjena.



The form has a dark blue background. It contains a blue button labeled "Nova Lozinka?" and a white text input field for the new password.

8. ODJAVA: Ako ste završili s radom, možete se jednostavno odjaviti klikom na botun "odjava" koji se nalazi u gornjem desnom kutu.



A blue button with the text "LOG OUT" in white capital letters.

Rezultati unit testova

1.ChangePassword.test.js

Ovaj unit test provjerava ponašanje funkcije ChangePassword u okviru "to-do" aplikacije. Test koristi Jest za simulaciju okoline, a također koristi jsdom za stvaranje lažnog DOM-a kako bi simulirao rad s HTML-om.

1. Uvoz potrebnih modula

```
1  const { JSDOM } = require('jsdom');
2  const { ChangePassword } = require('../todo_js/to_do');
3
4  const XMLHttpRequest = require('xhr2');
5  global.XMLHttpRequest = XMLHttpRequest;
6
7  const dom = new JSDOM(`
8    <!DOCTYPE html>
9    <html lang="en">
10
11      <head>
12        <meta charset="UTF-8">
13        <meta name="viewport" content="width=device-width, initial-scale=1.0">
14        <link rel="stylesheet" href="to_do.css">
15        <script defer src="to_do.js"></script>
16        <title> To Do </title>
17      </head>
18
19      <body>
20        <div class="container">
21
22
23
24          <header class="header">
25            <form method="post">
26              <h2 id="h2_hello">Hello </h2>
27            </form>
28            <button onclick="Logout()" name="log_out">LOG OUT</button>
29          </header>
30
31
32
33          <div class="task_list">
34
35
36            <div id="task">
37
```

```

38     </div>
39
40     <div id="task_input">
41         <input type="text" id="description_input" name="description_input">
42         <input type="date" id="date_input" name="date_input">
43         <button type="submit" name="add_task" onclick="AddTask()">Add</button>
44     </div>
45
46 </div>
47
48 <div class = "search_div">
49
50     <input type="text" id="search_input" name="search_input">
51     <button name="search_task" onclick="SearchTasks()">Search</button>
52     <button type="submit" name="refresh" onclick="Refresh()">Refresh all tasks</button>
53
54 </div>
55
56 <footer class="footer">
57     <button name="change_pass" onclick="ChangePassword()">Change Password?</button>
58     <input type="text" id="new_pass" name="new_pass">
59     <div id="error-pass"></div>
60 </footer>
61
62
63
64 </div>
65
66 </body>
67
68 </html>
69
70 `);
71
72 global.document = dom.window.document;
73

```

Ovdje se uvoze potrebni moduli poput JSDOM za stvaranje lažnog DOM-a, ChangePassword funkcije koja će biti testirana, i XMLHttpRequest koji će biti zamijenjen lažnom implementacijom za potrebe testiranja.

2. Opisuje se testna skupina

```

74 describe('ChangePassword', () => {
75     |

```

describe blok koristi Jest funkcionalnost za opisivanje skupa testova. U ovom slučaju, svi testovi povezani su s funkcijom ChangePassword.

3. Testiranje promjene lozinke i ažuriranje poruke o pogrešci

```
76 Complexity is 3 Everything is cool!
77 it('should change the password and update the error message', async () => {
78   // Mock user input
79   document.getElementById('new_pass').value = 'newPassword';
80
81   // mock for XMLHttpRequest
82   const mockXHR = {
83     open: jest.fn(),
84     setRequestHeader: jest.fn(),
85     send: jest.fn(),
86     onreadystatechange: jest.fn(),
87     readyState: 4,
88     status: 200,
89     responseText: 'Password changed.',
90   };
91
92   global.XMLHttpRequest = jest.fn(() => mockXHR);
93
94   ChangePassword();
95
96   await new Promise((resolve) => setTimeout(resolve, 0));
97
98   // verificirajmo da pozvani
99   expect(mockXHR.open).toHaveBeenCalledWith('POST', './php/change_password.php', true);
100   expect(mockXHR.setRequestHeader).toHaveBeenCalledWith('Content-type', 'application/x-www-form-urlencoded');
101
102   // za test prolaska
103   const expectedSendParameter = `email=${encodeURIComponent('')}&passw=${encodeURIComponent('newPassword')}`;
104
105   // verificirajmo da povuklo prave
106   expect(mockXHR.send).toHaveBeenCalledWith(expectedSendParameter);
107
108   // simulacija onreadystatechange event
109   mockXHR.onreadystatechange();
110
111   // verifikacija update-a errora
112   const errorMessage = document.getElementById('error-pass');
113   expect(errorMessage.innerHTML).toBe('Password changed.');
```

Ovaj test simulira unos nove lozinke, lažira XMLHttpRequest objekt s odgovarajućim podacima i provjerava je li funkcija ChangePassword pravilno poslala zahtjev za promjenom lozinke, ažurirala poruku o pogrešci i obradila odgovor servera.

4. Testiranje rukovanja praznim unosom lozinke

```
117 it('should handle an empty password input', async () => {
118
119   // Mock user input
120   document.getElementById('new_pass').value = '';
121
122   // Spy on XMLHttpRequest methods
123   const openSpy = jest.spyOn(XMLHttpRequest.prototype, 'open');
124   const setRequestHeaderSpy = jest.spyOn(XMLHttpRequest.prototype, 'setRequestHeader');
125   const sendSpy = jest.spyOn(XMLHttpRequest.prototype, 'send');
126
127   ChangePassword();
128
129   // cekamo da se izvrši
130   await new Promise((resolve) => setTimeout(resolve, 0));
131
132   // verificiramo da nismo pozvali XML...
133   expect(openSpy).not.toHaveBeenCalled();
134   expect(setRequestHeaderSpy).not.toHaveBeenCalled();
135   expect(sendSpy).not.toHaveBeenCalled();
136 });
137
```

Ovaj test simulira situaciju kada korisnik ostavi polje za unos lozinke prazno. Provjerava se je li funkcija `ChangePassword` ispravno rukovala praznim unosom, tj. nije li poslala nikakav `XMLHttpRequest` zahtjev.

U konačnici, ovi testovi osiguravaju da funkcija `ChangePassword` pravilno reagira na različite scenarije, uključujući promjenu lozinke i rukovanje praznim unosom.

2.isValidEmail.test.js

Ovaj unit test provjerava funkciju `isValidEmail` koja se koristi za validaciju e-mail adresa.

1. Uvoz funkcije za provjeru ispravnosti e-mail adrese

```
1  const { isValidEmail } = require('../todo_js/sign_up');
2
```

Ovdje se uvozi funkcija `isValidEmail` iz modula `sign_up` koji se nalazi u direktoriju `'../todo_js/'`. Očekuje se da će ta funkcija biti izvedena u ispravnom stanju.

2. Opisuje se testna skupina za funkciju `isValidEmail`

```
3  describe('isValidEmail', () => {
```

describe blok koristi Jest funkcionalnost za opisivanje skupa testova. Svi testovi u ovom bloku odnose se na funkciju **isValidEmail**.

3. Testiranje povratne vrijednosti za ispravne e-mail adrese

```
4      it('should return true for a valid email', () => {
5          expect(isValidEmail('test@example.com')).toBe(true);
6          expect(isValidEmail('user123@gmail.com')).toBe(true);
7      });
```

Prvi test provjerava da funkcija **isValidEmail** vraća **true** za dva ispravna e-mail formata (**test@example.com** i **user123@gmail.com**).

4. Testiranje povratne vrijednosti za neispravne e-mail adrese

```
9  ✓ it('should return false for an invalid email', () => {  
10    |   expect(isValidEmail('invalidemail')).toBe(false);  
11    |   expect(isValidEmail('user.hr')).toBe(false);  
12    | });
```

Drugi test provjerava da funkcija **isValidEmail** vraća **false** za dva neispravna e-mail formata (**invalidemail** i **user.hr**).

5. Testiranje povratne vrijednosti za ispravnu, ali nekonvencionalnu e-mail adresu

```
14  it('should return true for a valid but unconventional email', () => {  
15    |   // Test with the email "s@c.b"  
16    |   expect(isValidEmail('s@c.b')).toBe(true);  
17    | });  
18  });  
19
```

Treći test provjerava da funkcija **isValidEmail** vraća **true** za ispravnu, ali nekonvencionalnu e-mail adresu (**s@c.b**).

Ovaj testni skup osigurava da funkcija **isValidEmail** ispravno prepoznaje ispravne, neispravne i nekonvencionalne e-mail adrese. Uzorci e-mail adresa u testovima predstavljaju različite scenarije koje funkcija treba obraditi.

3.sign_up.test.js

Ovaj unit test provjerava funkcije povezane s procesom prijave (sign-up) u "to-do" aplikaciji.

1. Uvoz potrebnih modula i postavljanje osnovnog DOM-a

```
6  const { JSDOM } = require('jsdom');
7  const { attemptSignUp, isValidEmail, displayError, sendSignUpRequest, handleSignUpResponse } = require('../todo_js/sign_up');
8
9
10 // treba xhr2 paket za simulirati XMLHttpRequest
11 const XMLHttpRequest = require('xhr2');
12 global.XMLHttpRequest = XMLHttpRequest;
13
14 // napravimo bazni DOM za testirati ga - naš OG
15 const dom = new JSDOM(`
16 <!DOCTYPE html>
17 <html lang="en">
18 <head>
19   <meta charset="UTF-8">
20   <meta name="viewport" content="width=device-width, initial-scale=1.0">
21   <link rel="stylesheet" href="sign_up.css">
22   <script defer src="sign_up.js"></script>
23   <title> Sign In </title>
24 </head>
25 <body>
26   <div class="container">
27     <h1>Sign-In</h1>
28     <form name="prijava" onsubmit="attemptSignUp(event)">
29       <div class="input">
30         <label for="username">Username</b></label>
31         <input type="text" id="username" name="username" required>
32         <label for="email">E-mail</label>
33         <input type="text" id="email" name="email" required>
34         <label for="password">Password</label>
35         <input type="password" id="password" name="password" required>
36       </div>
37       <input type="submit" name="submit_signup" value="Sign Up">
38     </form>
39     <div id="error"></div>
40   </div>
41 </body>
42 </html>
43 `);
44
45 //namistimo document
46 global.document = dom.window.document;
```

2. Mockanje funkcije isValidEmail

```
60 // Mocking isValidEmail
61 jest.mock('../todo_js/sign_up', () => ({
62   ...jest.requireActual('../todo_js/sign_up'),
63   isValidEmail: jest.fn(),
64 }));
65
```

Koristi se **jest.mock** kako bi se simulirala funkcija **isValidEmail** kako ne bi stvarno utjecala na rezultate testiranja.

3. Postavljanje globalnih objekata kao mockova

```
66 // simuliramo storage i window jer se koriste u funkcijama
67 global.localStorage = {
68   | setItem: jest.fn(),
69 };
70 global.window = {
71   | location: { href: '' },
72   | alert: jest.fn(),
73 };
74
```

4. Opisivanje skupa testova vezanih uz funkciju attemptSignUp:

```
complexity is 9 it's time to do something...
75 ✓ describe('attemptSignUp', () => {
76   | beforeEach(() => {
77     |   jest.clearAllMocks();
78     |   document.getElementById('error').innerHTML = '';
79     | });
80
```

beforeEach blok se izvršava prije svakog testa unutar ovog bloka kako bi se osiguralo da su svi mockovi resetirani i DOM postavljen na početno stanje.

5. Testiranje ispravnog unosa podataka

```
81 it('Correct input pass', async () => {
82   isValidEmail.mockReturnValue(true);
83
84   const mockEvent = {
85     preventDefault: jest.fn(),
86     target: {
87       id: 'your-form-id',
88       elements: {
89         email: { value: 'test@example.com' },
90         password: { value: 'yourpassword' },
91         username: { value: 'yourusername' },
92       },
93     },
94   };
95
96   // mock za XMLHttpRequest
97   const mockXHR = {
98     open: jest.fn(),
99     setRequestHeader: jest.fn(),
100    send: jest.fn(),
101    onreadystatechange: jest.fn(),
102    readyState: 4,
103    status: 200,
104    responseText: 'success',
105  };
106
107  global.XMLHttpRequest = jest.fn(() => mockXHR);
108
109  attemptSignUp(mockEvent);
110
111  // čekamo da se XMLHttpRequest izvrši
112  await new Promise((resolve) => setTimeout(resolve, 0));
113
114  expect(mockEvent.preventDefault).toHaveBeenCalled();
115 });
116
117
118
```

Prvi test simulira ispravan unos podataka i provjerava je li funkcija **attemptSignUp** ispravno postavila XMLHttpRequest zahtjev i izvršila očekivane radnje nakon uspješnog odgovora servera.

6. Testiranje prikaza poruke o grešci

```
119 | it('Successful error message', () => {  
120 | |   displayError('Test error message');  
121 | |   expect(document.getElementById('error').innerHTML).toBe('Test error message');  
122 | | });  
123 |
```

Drugi test provjerava je li funkcija **displayError** ispravno postavila poruku o grešci na HTML elementu s ID-om 'error'.

7. Testiranje slanja zahtjeva za prijavom

```
124 | it('should send signup request', () => {  
125 | |   const email = 'test@example.com';  
126 | |   const password = 'yourpassword';  
127 | |   const username = 'yourusername';  
128 | |  
129 | |   // Set up a mock for XMLHttpRequest  
130 | |   const mockXHR = {  
131 | | |   open: jest.fn(),  
132 | | |   setRequestHeader: jest.fn(),  
133 | | |   send: jest.fn(),  
134 | | |   onreadystatechange: jest.fn(),  
135 | | |   readyState: 4,  
136 | | |   status: 200,  
137 | | |   responseText: 'success',  
138 | | | };  
139 | |  
140 | |   global.XMLHttpRequest = jest.fn(() => mockXHR);  
141 | |  
142 | |   sendSignUpRequest(email, password, username);  
143 | |  
144 | |   expect(mockXHR.open).toHaveBeenCalledWith('POST', './php/sign_up.php', true);  
145 | | });  
146 |
```

Treći test provjerava je li funkcija **sendSignUpRequest** ispravno postavila XMLHttpRequest zahtjev za slanje podataka za prijavu na server.

8. Testiranje obrade odgovora servera nakon prijave

```
147   it('handle SignUpResponse', () => {
148     const email = 'test@example.com';
149     const status = 200;
150     const responseText = 'success';
151
152     handleSignUpResponse(status, responseText, email);
153
154     // Provjerava jesu li pozvani localStorage.setItem i window.location.href
155     expect(localStorage.setItem).toHaveBeenCalledWith('email', email);
156     expect(window.location.href).toBe('to_do.html');
157
158     // da nema alerta
159     expect(window.alert).not.toHaveBeenCalled();
160
161   });
162 });
163
```

Četvrti test provjerava je li funkcija `handleSignUpResponse` ispravno postavila `localStorage` i `window.location.href` nakon uspješne prijave, te provjerava da nije pozvan `window.alert`.

Ovaj unit test skup pokriva različite aspekte funkcionalnosti povezane s prijavom u aplikaciji, uključujući unos podataka, prikaz poruka o grešci, slanje zahtjeva na server i obradu odgovora servera.

4.to_do.test.js

Ovaj unit test provjerava funkcije **Obrada_Username** i **extractNumberFromString** koje su dio “to-do” aplikacije.

1. Uvoz potrebnih modula i mockanje funkcija

```
1  const { JSDOM } = require('jsdom');
2  const { extractNumberFromString, Obrada_UserTasks, Obrada_Username } = require('../todo_js/to_do');
3
4  jest.mock('../todo_js/to_do', () => ({
5    ...jest.requireActual('../todo_js/to_do'),
6    GetUser: jest.fn(),
7    // ... mock other functions if needed
8  }));
9
```

Ovdje se uvoze funkcije koje želimo testirati (**extractNumberFromString** i **Obrada_Username**) i mockaju funkcije koje nisu direktno povezane s ovim testom (npr., **GetUser**). Mockanje je korisno kako bismo kontrolirali ponašanje funkcija i fokusirali se samo na funkcije koje trenutno testiramo.

2. Postavljanje osnovnog DOM-a za testiranje

```
11  ✓ const dom = new JSDOM(`  
12    <!DOCTYPE html>  
13    <html lang="en">  
14  
15      <head>  
16        <meta charset="UTF-8">  
17        <meta name="viewport" content="width=device-width, initial-scale=1.0">  
18        <link rel="stylesheet" href="to_do.css">  
19        <script defer src="to_do.js"></script>  
20        <title> To Do </title>  
21      </head>  
22  
23      <body>  
24        <div class="container">  
25  
26  
27  
28          <header class="header">  
29            <form method="post">  
30              <h2 id="h2_hello">Hello </h2>  
31            </form>  
32            <button onclick="Logout()" name="log_out">LOG OUT</button>  
33          </header>  
34  
35  
36  
37          <div class="task_list">  
38  
39  
40            <div id="task">  
41  
42            </div>  
43  
44            <div id="task_input">  
45              <input type="text" id="description_input" name="description_input">  
46              <input type="date" id="date_input" name="date_input">  
47              <button type="submit" name="add_task" onclick="AddTask()">Add</button>  
48            </div>  
49          </div>  
50  
51  
52          <div class = "search_div">  
53  
54            <input type="text" id="search_input" name="search_input">  
55            <button name="search_task" onclick="SearchTasks()">Search</button>  
56            <button type="submit" name="refresh" onclick="Refresh()">Refresh all tasks</button>  
57          </div>  
58  
59  
60          <footer class="footer">  
61            <button name="change_pass" onclick="ChangePassword()">Change Password?</button>  
62            <input type="text" id="new_pass" name="new_pass">  
63            <div id="error-pass"></div>  
64          </footer>  
65  
66  
67        </div>  
68      </body>  
69    </html>  
70  
71  `);  
72  
73  
74  
75  `);  
76  
77  //namistimo document  
78  global.document = dom.window.document;
```

Ovdje se stvara lažni DOM pomoću JSDOM kako bi se simulirala HTML struktura aplikacije.

3. Mockanje document.getElementById funkcije

```
79
80 document.getElementById = jest.fn();
81
```

4. Testiranje funkcije Obrada_Username

```
83 describe('Obrada_Username', () => {
84   it('should update the innerHTML of the specified element', () => {
85     // Mock the necessary elements and values
86     const mockedElement = {
87       innerHTML: '',
88     };
89     document.getElementById.mockReturnValueOnce(mockedElement);
90
91     // Call the function
92     Obrada_Username('Hello, user');
93
94     // Check if the innerHTML is updated as expected
95     expect(mockedElement.innerHTML).toBe('Hello, user!');
96   });
97 });
98
```

Prvi test provjerava je li funkcija **Obrada_Username** ispravno ažurirala **innerHTML** određenog elementa na temelju predanog teksta.

5. Testiranje funkcije extractNumberFromString

```
99 describe('extractNumberFromString', () => {
100     it('should extract a number from a string', () => {
101         const inputString = 'abc123xyz';
102         const result = extractNumberFromString(inputString);
103         expect(result).toBe(123);
104     });
105
106     it('should return null if no numbers are found', () => {
107         const inputString = 'noNumbersHere';
108         const result = extractNumberFromString(inputString);
109         // Je li null
110         expect(result).toBeNull();
111     });
112 });
```

Drugi test provjerava funkciju `extractNumberFromString` koja treba izvući broj iz niza znakova. Prvi test provjerava je li funkcija uspješno izvukla broj iz niza, dok drugi test provjerava vraća li funkcija `null` ako nema brojeva u nizu.

Ovaj unit test skup osigurava da funkcije `Obrada_Username` i `extractNumberFromString` rade ispravno i da se ponašaju kako se očekuje u različitim scenarijima. Mockanje funkcija omogućava izolirano testiranje ovih funkcija bez utjecaja drugih dijelova aplikacije.

5. Rezultati testova i code coverage

```
PS C:\xampp\htdocs\To_Do_Projekt\To-Do-projekt-UI-> npm test
```

```
> to-do-projekt-upi-@1.0.0 test
> jest
```

```
PASS testovi/sign_up.test.js
PASS testovi/to_do.test.js
PASS testovi/ChangePassword.test.js
• Console
```

```
console.log
  newPassword
```

```
    at log (todo_js/to_do.js:186:13)
```

```
console.log
```

```
    at log (todo_js/to_do.js:186:13)
```

```
PASS testovi/isValidEmail.test.js
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	31.11	21.73	34.61	31.34	
sign_up.js	79.16	37.5	83.33	79.16	13,29-30,46-49
to_do.js	20.72	18.42	20	20.9	7-32,44-175,214-236,252-255

```
Test Suites: 4 passed, 4 total
```

```
Tests: 12 passed, 12 total
```

```
Snapshots: 0 total
```

```
Time: 4.714 s, estimated 6 s
```

```
Ran all test suites.
```

```
PS C:\xampp\htdocs\To_Do_Projekt\To-Do-projekt-UI->
```

Prolaznost testova:

- Svi testovi su prošli (PASS), što znači da nema grešaka ili problema u funkcionalnosti testiranih dijelova koda.

Vrijeme izvršavanja:

- Ukupno vrijeme izvršavanja svih testova je 4.714 sekundi, s procijenjenim vremenom izvršavanja od 6 sekundi.

Console Output (Ispis u konzoli):

- U dijelu "Console" vidimo ispis iz konzole tijekom izvršavanja testova. Postoje dva poziva funkcije console.log u datoteci todo_js/to_do.js na liniji 186. Ovaj ispis može biti koristan za praćenje i debugiranje.

Pokrivenost koda (Code Coverage):

- Prikazana je pokrivenost koda testovima u postotku (% Stmts, % Branch, % Funcs, % Lines). Na primjer, to_do.js ima pokrivenost od 20.72% za izjave (Stmts), 18.42% za grananja (Branch), 20% za funkcije (Funcs), i 20.9% za linije koda (Lines).
- Navedeni su i dijelovi koda koji nisu pokriveni testovima, označeni kao "Uncovered Lines".


Detalji o pokrivenosti po datotekama:

- Za svaku datoteku (u ovom slučaju sign_up.js i to_do.js), prikazani su postoci pokrivenosti za izjave, grananja, funkcije i linije koda. Također su navedene linije koda koje nisu bile pokrivene testovima.

Zaključak:

- Ukupno je 4 test suite-a izvršeno, s ukupno 12 testova, svi su prošli, i ukupno vrijeme izvršavanja bilo je 4.714 sekundi.


6. Rad funkcija u aplikaciji





The image shows a registration form with the title "Registracija". It includes three input fields: "Korisničko ime" (User name), "E-mail", and "Lozinka" (Password). The "E-mail" field is highlighted with a warning message "Ispunite ovo polje." (Fill this field). A "Sign Up" button is located at the bottom of the form.

REGISTRACIJA: nakon što odaberete registraciju, preusmjereni ste na stranicu s registracijom. Tu unesite vaše korisničko ime, e-mail i lozinku, zatim kliknite na Sign up” botun. Ako niste unijeli nešto od navedenoga iskoči upozorenje da morate ispuniti odabrano polje i ne možete se registrirati u aplikaciju.

Registracija

Korisničko ime 

E-mail 


Lozinka 


[Sign Up](#)

Napišite mail u točnom formatu.


Ako ste mail napisali u krivom formatu umjesto **test@example.com** i **user123@gmail.com**, i dalje se nećete moći registrirati i prikazat će vam se upozorenje da napišete mail u točnom formatu.

Prijava

E-mail 


Lozinka 


[Login](#)

 Ispunite ovo polje.

Netočan mail ili lozinka

Prijava

E-mail 

Lozinka 

[Login](#)

Netočan mail ili lozinka

Tijekom prijave ako ne ispunite jedno od dva polja izbacit će vam obavijest da morate ispuniti prazno polje. Kada ste ispunili sva polja, ali ste unijeli mail ili lozinku koju niste registrirali izbacit će vam obavijest da ste unijeli netočan mail ili lozinku.

Unesite task

UPI 2024-01-21 Finished X

Informatika 2024-01-17 Finished X **USKORO!!!**

Projekt 2024-01-16 Finished X **USKORO!!!**

📅 Dodaj

Kada smo se prijavili i ušli u aplikaciju, imamo ispod naslova “Unesite task” botun “Dodaj” sa kojim dodajemo zadatke, poslove, obavijesti i slično koje moramo izvršiti. Kada unesemo zadatak, imamo opciju za odabrati datum do kada treba taj zadatak izvršiti. Ako se određeni zadatak treba izvršiti u roku od 3 dana, pored zadatka i datuma pisati će nam obavijest “USKORO!!!” koja nam govori da se uskoro bliži rok. Također zadatke možemo označiti i dovršenima klikom na botun “Finished” koji promijeni boju u zelenu ako je zadatak izvršen, a ako nije izvršen onda je crvene boje. Pored botuna “Finished” imamo botun sa znakom X, koji briše zadatak uz koji se nalazi.

Unesite task

UPI 2024-01-21 Finished X

Informatika 2024-01-17 Finished X **USKORO!!!**

Projekt 2024-01-16 Finished X **USKORO!!!**

Prezentacija 0000-00-00 Finished X

📅 Dodaj

Kada dodamo zadatak, a da pritom nismo označili datum do kojeg trebamo izvršiti taj zadatak, aplikacija postavi sve nule i nema obavijesti “USKORO!!!”

Unesite task

Informatika 2024-01-17 Finished X **USKORO!!!**

Projekt

16. 01. 2024.

Dodaj

Inf

Traži

Refresh tasks

Ako ste unijeli više zadataka i želite pronaći neki od njih, to možete napraviti pomoću botuna “Traži”. Nakon što unesete zadatak koji tražite u polje koje se nalazi pored botuna “Traži”, kliknite na botun “Traži” i pronaći ćete zadatak koji ste tražili. Klikom na Refresh se ponovno prikažu svi taskovi.

Nova Lozinka?

Enter new password

Ukoliko želite promijeniti lozinku, kliknete na botun “Nova Lozinka?” i prikaže vam se poruka “Enter new password”. Kada se prikaže poruka, u sredini možete upisati svoju novu lozinku i kliknuti na botun “Nova Lozinka?”.

Nova Lozinka?

12345

Password changed.

Kada ste promijenili lozinku prikazati će vam se poruka “Passwoer changed.” kao znak da ste uspješno promijenili svoju lozinku.

Code complexity

1. Sign_up.js

JSsign_up.jsCode Complexity Report

Code Complexity Report

So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.

— Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin*

Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
attemptSignUp	1	14	2	11	1
isValidEmail	16	19	1	3	1
displayError	21	23	1	3	1
xhr.onreadystatechange	28	32	2	5	0
sendSignUpRequest	25	37	1	7	3
handleSignUpResponse	39	51	3	12	3

Source Code: fdba059a-b232-11ee-ad28-0242ac100002.js

2. Log_in.js

JSlog_in.jsCode Complexity Report

Code Complexity Report

So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.

— Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin*

Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
xhr.onreadystatechange	10	28	4	16	0
attemptLogin	1	35	1	10	1

Source Code: c5f70d24-b232-11ee-ad28-0242ac100002.js

Generated by [Codalyze](#) on 2024-01-13 16:42

3.To_do.js

Code Complexity Report

So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.

— Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin*

Function Name	Start Line	End Line	Cyclomatic Complexity <small>(Threshold: 10)</small>	Lines of Code <small>(Threshold: 50)</small>	Parameter Count <small>(Threshold: 4)</small>
GetUser	6	10	1	5	0
xhr.onreadystatechange	15	28	3	11	0
GetUsername	12	33	1	7	0
Obrada_Username	35	39	1	4	1
xhr.onreadystatechange	45	55	3	9	0
GetTasks	42	60	1	7	0
Obrada_Tasks	61	99	4	27	1
xhr.onreadystatechange	111	116	3	5	0
AddTask	102	119	1	9	0
xhr.onreadystatechange	130	136	3	5	0
DeleteTask	122	144	1	9	1
xhr.onreadystatechange	157	171	4	11	0
ChangeFinishedStatus	148	178	1	9	1
xhr.onreadystatechange	197	203	3	5	0
ChangePassword	181	209	2	14	0
xhr.onreadystatechange	219	227	3	7	0
SearchTasks	212	233	1	8	0
Refresh	235	237	1	3	0
extractNumberFromString	240	249	2	9	1
LogOut	251	258	1	6	0

4. ChangePassword.test.js

JS ChangePassword.test.js

Code Complexity Report

Code Complexity Report

So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.

— Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin*

Function Name	Start Line	End Line	Cyclomatic Complexity <small>(Threshold: 10)</small>	Lines of Code <small>(Threshold: 50)</small>	Parameter Count <small>(Threshold: 4)</small>
(anonymous)	91	91	1	1	0
(anonymous)	95	95	1	1	0
jest.fn	85	113	1	16	0
(anonymous)	130	130	1	1	0
(anonymous)	117	136	1	11	0
jest.fn	84	137	1	4	0

Source Code: e19daf8c-b233-11ee-b93e-0242ac100002.js

5. isValidEmail.test.js

JS isValidEmail.test.js

Code Complexity Report

Code Complexity Report

So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.

— Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin*

Function Name	Start Line	End Line	Cyclomatic Complexity <small>(Threshold: 10)</small>	Lines of Code <small>(Threshold: 50)</small>	Parameter Count <small>(Threshold: 4)</small>
(anonymous)	4	7	1	4	0
(anonymous)	9	12	1	4	0
(anonymous)	14	17	1	3	0
(anonymous)	3	18	1	5	0

Source Code: fc39b17e-b233-11ee-b93e-0242ac100002.js

6.sign_up.test.js

JS sign_up.test.js Code Complexity Report					
Code Complexity Report					
So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.					
— Clean Code: A Handbook of Agile Software Craftsmanship <i>Robert C. Martin</i>					
Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
jest.fn	63	64	1	2	0
(anonymous)	76	79	1	4	0
(anonymous)	107	107	1	1	0
(anonymous)	112	112	1	1	0
jest.fn	101	115	1	10	0
(anonymous)	119	122	1	4	0
(anonymous)	140	140	1	1	0
jest.fn	134	145	1	9	0
(anonymous)	147	161	1	9	0
jest.fn	133	162	1	4	0
Source Code: 1469e84a-b234-11ee-b93e-0242ac100002.js					

7.to_do.test.js

JS to_do.test.js Code Complexity Report					
Code Complexity Report					
So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.					
— Clean Code: A Handbook of Agile Software Craftsmanship <i>Robert C. Martin</i>					
Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
jest.fn	6	8	1	2	0
(anonymous)	84	96	1	8	0
(anonymous)	83	97	1	3	0
(anonymous)	100	107	1	5	0
(anonymous)	109	117	1	5	0
(anonymous)	99	119	1	4	0
Source Code: 44334116-b234-11ee-b93e-0242ac100002.js					

Performance

