Utveckling

Databasdesign

För detta projektet har fyra tabeller skapats, detta inkluderar "Users", som lagrar användarinformation, "Lists" som innehåller data om varje lista, "Tasks" som sparar detaljer om varje uppgift, samt "ListMembers" som hanterar information om medlemmarna i varje lista. I “Users” tabellen sparas en id automatiskt vilket används i andra listor för att referera till användaren. Liknande id sparas i “Lists” tabellen vilket används tillsammans med användarid för att koppla samman listors medlemmar i "ListMembers"-tabellen. I "Lists"-tabellen finns en kolumn kallad "token", som automatiskt genereras av PHP vid skapandet av en ny lista. Token används för att referera till listorna genom att inkludera den i inbjudnings länken vilket gör länkarna unika.

För att spara information om vilka uppgifter som är avklarade och vem som har slutfört dem, används kolumnerna "status" och "completeUser". Kolumnen "status" lagrar ett värde som antingen är 0 (inte avklarad) eller 1 (avklarad), medan "completeUser" sparar användarens ID, vilket sedan används för att referera till "Users"-tabellen.

Säkerhetsaspekter

Enligt Nixon (2014, s. 294) bör lösenord inte sparas i klartext i databasen, eftersom webbplatsen kan bli utsatt för hot, vilket kan leda till att den kompromitteras och att lösenorden blir synliga för hackare. Av denna anledning har lösenorden hashats med PHP-funktionen password_hash(), som skapar en ny hash av lösenordet med en stark envägs hashing-algoritm (PHP Dokumentation Grupp, u.å.a). För att sedan verifiera lösenordet har PHP-funktionen password_verify() använts.

För att förhindra användare från att använda listor när de är utloggade, har superglobalen $_SESSION använts för att lagra variabeln 'user'. Denna variabel används sedan för att kontrollera om användaren är inloggad genom att kolla om värdet är satt.

Hantering av användardata kan innebära en säkerhetsrisk, eftersom den kan innehålla skadlig JavaScript-kod eller MySQL-kommandon som kan påverka webbplatsens funktion och äventyra databasen (Nixon, 2014, s. 277). För att minimera dessa risker är det viktigt att implementera skyddsmekanismer som säkerställer att indata behandlas på ett säkert sätt. För att lösa detta har funktionen sanitizeString() skapats. Den tar emot användarinmatning och returnerar en version där potentiellt skadliga element har filtrerats bort. Funktionen använder sig av flera PHP-funktioner, såsom strip_tags(), htmlentities() och stripslashes(), samt MySQLi-metoden real_escape_string() för att säkerställa att datan hanteras korrekt. Enligt Nixon (2014, s. 278) avlägsnar stripslashes() snedstreck, htmlentities() konverterar särskilda tecken till deras motsvarande HTML-entiteter, och strip_tags() rensar bort HTML-taggar. Utöver detta används real_escape_string(), vilket enligt PHP Dokumentationsgruppen (u.å.b) hanterar specialtecken och escape-sekvenser, såsom enkla och dubbla citattecken, för att förhindra att SQL-kommandon manipuleras. På så sätt kan indata behandlas på ett säkert sätt utan att riskera webbplatsens eller databasens integritet.

Övrigt

För att förenkla de olika åtgärderna som användare kan utföra har CRUD-operationer implementerats via ett API. Dessa operationer inkluderar att hämta alla listor och uppgifter, uppdatera uppgifternas status, ta bort listor eller uppgifter, samt skapa nya listor eller uppgifter. API:et har byggts med hjälp av Slim 3, ett PHP-ramverk som fungerar som en dispatcher. Det tar emot HTTP-förfrågningar, dirigerar dem till rätt callback-funktion och returnerar en HTTP-respons (Slim Framework, u.å.). Detta gör det möjligt att samla alla funktioner i en enda fil, där Slim sköter alla andra uppgifter, såsom att hitta den rätta funktionen och returnera den lämpliga responsen.

För att säkerställa att användare inte kan ta bort eller ändra statusen på uppgifter eller listor de inte äger, har en kontroll implementerats. Genom en if-sats jämförs användarens ID med ägarens ID, vilket gör att endast den som äger en lista eller som har slutfört en uppgift kan utföra ändringar på den.

Krav och utvärdering

Funktionella krav

- Användare skall behöva logga in för att använda appen
    - Det ska gå att registrera sig som ny användare
- En användare skall kunna skapa en todo-lista
- En användare skall kunna bjuda in andra till en todo-lista
    - Om användaren redan befinner sig i listan ska de inte läggas till igen
- En användare skall inte kunna uppdatera data knutet till någon annan användare
- Alla deltagare i en todo-lista skall kunna lägga till saker till listan
    - En sak i listan definieras av *titel*, *antal karma-poäng* och ev. *beskrivning*
- Alla deltagare i en todo-lista skall kunna markera saker som avklarade
    - Avklarade saker skall finnas kvar men markeras som avklarade
    - Information om vem som klarat av en sak skall spara
    - Den användare som klarar en uppgift får det antal karma-poäng som uppgiften innehåller
- Det skall tydligt visas hur mycket karma-poäng varje användare har för varje lista
- Användargränssnittet skall vara responsiv och kunna användas på mobilen

Tekniska krav

- Backend skall byggas i PHP
- Data skall spara i mysql-databas på servern
- Backend skall innehålla ett REST-api
- Projektet ska byggas i form av en SPA
- Frontend ska byggas utan externt UI-ramverk.
- Javascript Kod skall bearbetas och anpassas för frontend

Kravet på en SPA (Single Page Application) har delvis uppfyllts genom användning av endast två HTML-sidor, men på grund av tidsbegränsningar och flera uppkomna problem har det inte kunnat implementeras helt.

Referenslista

Nixon, R. (2014) *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5*. 4 uppl. USA: O'Reilly Media, Inc.
PHP Dokumentation Grupp (u.å.a) *Password_hash.* https://www.php.net/manual/en/function.password-hash.php [2025-03-19]
PHP Dokumentation Grupp (u.å.b) *Mysqli::real_escape_string.* https://www.php.net/manual/en/mysqli.real-escape-string.php [2025-03-30]
Slim Framework (u.å) *Slim 3 Documentation.* https://www.slimframework.com/docs/v3/ [2025-03-30]