**טכנולוגיות אינטרנט מתקדמות - 61776 (WEB)**

**הגשת פרויקט**

**<**Customizable News Aggregator **> <** B13**><20>**

| **שם חבר.ת הצוות** | **תז** |
| --- | --- |
| מרוה חמוד | 212102263 |
| סילין אסדי | 324300706 |
| ענוד נעים | 324222553 |
| יונתן שרר | 318317682 |

**תקציר הפרויקט :**

האתר שלנו היה להביא ולהציג למשתמש חדשות מקטגוריות שונות והוא יכול בעת כניסה לפרופיל שלו לבחור חדשות ויסמן אותם כמועדפות

**הפונקציות המרכזיות של האתר :**

1. חיפוש חדשות : בעת רישום מילה משמעותית ולוחצים על כפתור search אז מוצג כל החדשות הקשורות למילת החיפוש

2.בחירת קטגוריה מהקטגוריות המוצעות למשתמש ובכך יוצג לו החדשות הקשורות לקטגוריה זו.

3. הוספת מאמר למאגר החדשות ה- Favorite של המשתמש .

4. מחיקת חדשה בחדשות שסימנו favorite - : דרך כפתור שנקרא remove, או דרך לחיצה על כוכב ה favorite שוב .

5.עדכון פרופיל משתמש : שנכנס המשתמש ל Profile Card יכול לשנות מספר טלפון, מייל, שם משתמש.

6. התחברות לאתר : בעת הזנת הפרטים - סיסמה , שם משתמש ולוחצים על כפתור Login המערכת מציגה הודעה מתאימה לפי הפרטים שהוזנו .

7. הרשמה לאתר : בעת הזנת הפרטים ולוחצים על כפתור register המערכת מציגה הודעה מתאימה לפי הפרטים שהוזנו .

**הוראות להרצת הקוד :**

נא לפתוח 2 טרמינלים - אחד לשרת ואחד ללקוח

בצד השרת -

cd server

npx nodemon / node server.js

בצד לקוח -

cd client

npm start

**מימוש-שמות הטכנולוגיות המרכזיות בכל אחד מהחלקים:**

Styling: CSS+ tailwind + React

Db: MongoDB

Backend: VS Code using Node.js and Express.js

Frontend: VS Code using React, JavaScript, HTML, Axios, React Router

קישור לתיקיית גיט : <https://github.com/CelineAsadi/website-project>

<https://website-project-client.vercel.app/> : vercel קישור

# קישור ל MTW:

<https://www.morethanwallet.com/app/883>

מהנדסת המערכת : מרוה חמוד

בחרנו את מרוה להיות אחראית על חלוקת המשימות כיוון שהיא יודעת ומבינה בבניית אתרים וגם מבחינת חלוקה הוגנת ושווה של המשימות בין חברי הצוות, יש לה את האופי של בן אדם מסודר אחראי..

ותמיד היה שיתוף פעולה בין חברי הצוות ועדכונים כל הזמן, ישבנו, דנו וכל בעיה שמתרחשת תמיד באותו עת הינו מבצעים תיקון.

| **שם חבר הצוות** | **משימות שהוקצו** | **משימות שהושלמו** |
| --- | --- | --- |
| מרוה חמוד | -בדיקת הקוד+ איתור באגים  - החיבור בין צד לקוח ושרת  - שיפור הפונקציונליות במסך הרישום  -Use case diagram | הכל |
| יונתן שרר | -עדכון קוד של ה dark/light  - בדיקה ועדכון הקוד ל react  -העיצוב יהיה דרך tailwind  - דיאגרמת מבנה DB. | הכל |
| ענוד נעים | -תיקון ובדיקת הסכמות ב MongoDB  -שינוי הקוד של מסך החדשות המועדפים  -שיפור קוד ה-Login  -שמירת החדשות במועדים ב DB | הכל |
| סילין אסדי | -החיבור עם הmongoDB  -עדכון קוד Profile Card  -העלאת האתר ל vercel  -שיפור הקוד של Forgot password | הכל |

**Functionality Requirements:**

-The system allows registration.

-The system allows identification.

The system allows reset user's password-

-The system displays news by category.

-The system allows users to select a category.

-The system allows users to choose their favorite news.

The system aggregates various news. -

-The system allows searching news

-The system allows update profile

The system allows removing news from favorite news .

The system allows the users to select dark or light mode.

**Non-Functionality Requirements:**

-  **Usability:**

The application should have an intuitive and easy-to-navigate interface.

- **Backup:**

constant backup of user information

- **Accessibility**:

The system argument and response time should be minimal

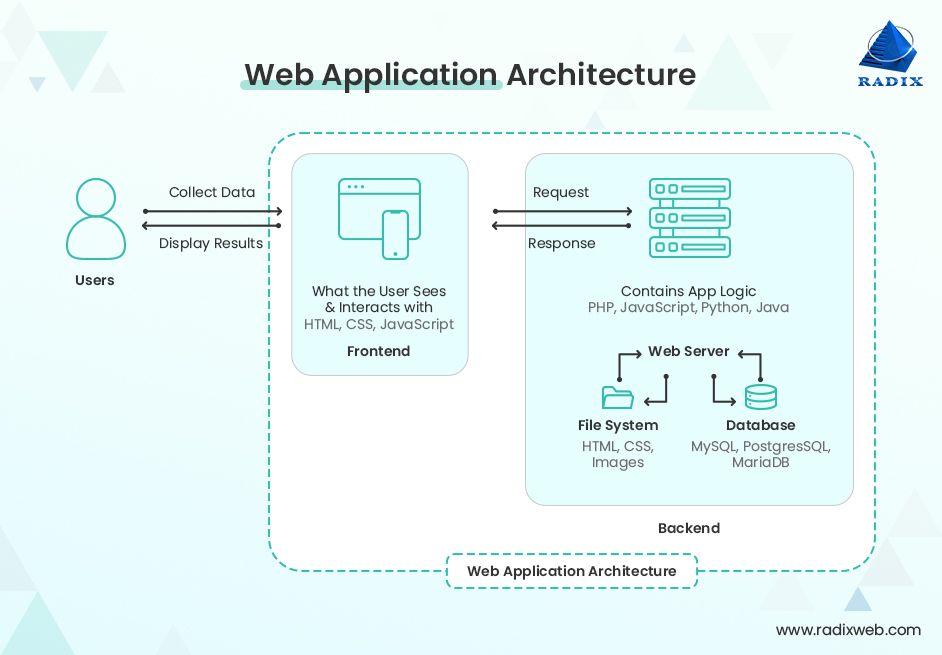
The system should handle many simultaneous users.

The system should be compatible with major we browser

**-Maintainability:**

The codebase should be modular and well-documented to facilitate easy updates and maintenance

**system architecture:**

**Frontend**:

* **ReactJS:** Create the user interface using ReactJS for structure, Tailwind for styling, and JavaScript for interactivity.
* **Tailwind CSS:** Utilize Tailwind CSS for quickly styling your UI components.

**Backend**:

* + **Express.js**: A web application framework for Node.js that simplifies routing, middleware, and other common tasks.
  + **Node.js**: Use Node.js to build the backend of your application. It's efficient for handling asynchronous I/O operations.

**Database**:

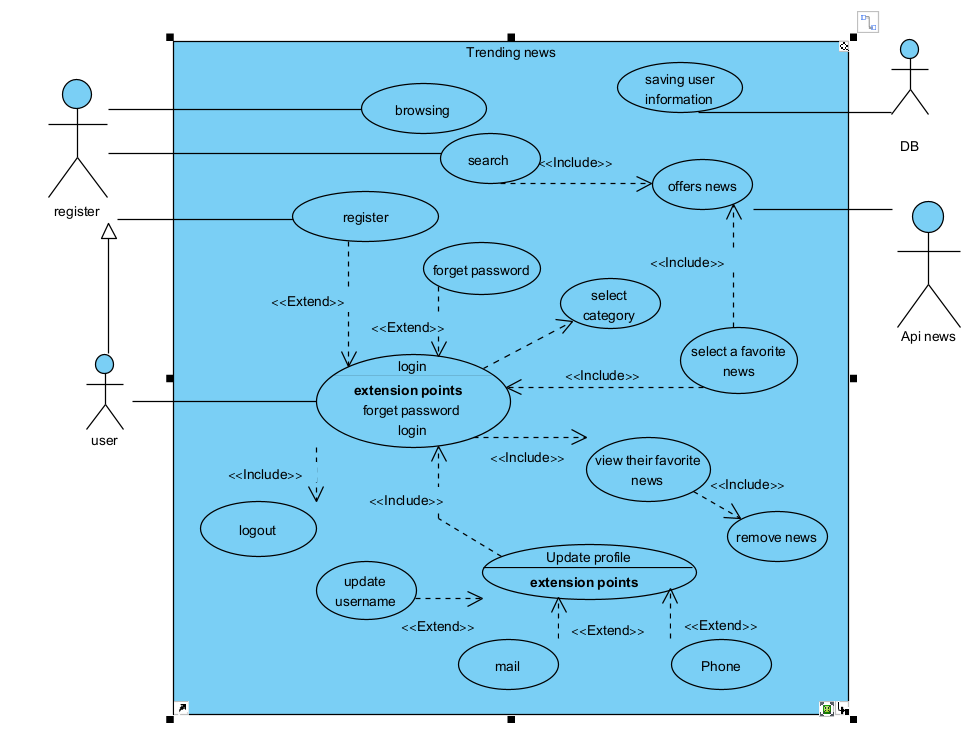
* + **MongoDB**: A NoSQL database that stores data in JSON-like documents. It's flexible and scalable, suitable for various types of applications.

**News API**

**Application Architecture**:

* + **Client-Server Architecture**: Follow the traditional client-server architecture where the frontend interacts with the backend through HTTP requests.

**Use case:**

****

**דיאגרמת מבנה DB :**

| user |
| --- |
| ID (ObjectID) |
| username (String) |
| Email (String) |
| Phone (String) |
| password (String) |
| phone (String) |
| favoriteNews (Array[url(string),description(string)]) |

**\*הערה לגבי מערך של favoritenews : בכל תא במערך יש שני שדות שסוגם string**

**נא להציג דיאגרמה המתארת את התיקיות והקבצים השונים. יש לפרט את הקומפוננטות השונות:**

PROJECT20/

│

├── client/

│ ├── node\_modules/ # Dependencies for the client-side

│ ├── public/ # Public files that are served directly

│ │ ├── favicon.ico # The website's favicon

│ │ ├── index.html # The main HTML template

│ │ ├── manifest.json # Metadata for the web application

│ │ └── robots.txt # Instructions for web crawlers

│ ├── src/

│ │ ├── components/ # Main directory for React components

│ │ │ ├── Logic/ # Logic files that handle component functionalities

│ │ │ │ ├── FavoritesPageLogic.js # Logic for Favorites page

│ │ │ │ ├── ForgotPasswordLogic.js # Logic for Forgot Password page

│ │ │ │ ├── HomeLogic.js # Logic for Home page

│ │ │ │ ├── LoginLogic.js # Logic for Login page

│ │ │ │ ├── ProfileCardLogic.js # Logic for Profile Card

│ │ │ │ ├── ProfileLogic.js # Logic for Profile page

│ │ │ │ └── RegisterLogic.js # Logic for Registration page

│ │ │ ├── Style/ # CSS files for styling components

│ │ │ │ ├── FavoritesPage.css # Styles for Favorites page

│ │ │ │ ├── ForgotPassword.css # Styles for Forgot Password page

│ │ │ │ ├── Home.css # Styles for Home page

│ │ │ │ ├── Login.css # Styles for Login page

│ │ │ │ ├── Profile.css # Styles for Profile page

│ │ │ │ ├── ProfileCard.css # Styles for Profile Card

│ │ │ │ └── Register.css # Styles for Registration page

│ │ │ ├── App.jsx # Main app component

│ │ │ ├── FavoriteNews.jsx # Component for displaying favorite news

│ │ │ ├── Footer.js # Footer component

│ │ │ ├── ForgotPassword.jsx # Forgot Password component

│ │ │ ├── Home.jsx # Home page component

│ │ │ ├── Login.jsx # Login page component

│ │ │ ├── Profile.jsx # Profile page component

│ │ │ ├── ProfileCard.jsx # Profile Card component

│ │ │ └── Register.jsx # Registration page component

│ │ ├── App.css # Global styles for the app

│ │ ├── index.js # Entry point for the React application

│ │ ├── index.css # Global styles for the index

│ │ ├── newsService.js # Service for handling news-related API requests

│ │ └── setupTests.js # Configuration for setting up tests

│ ├── .gitignore # Files and directories to be ignored by Git

│ ├── package-lock.json # Lockfile for the exact dependency tree

│ ├── package.json # Dependencies and scripts for the client-side

│ ├── README.md # Documentation for the client-side

│ └── tailwind.config.js # Configuration for Tailwind CSS

│

└── server/

├── node\_modules/ # Dependencies for the server-side

├── .gitignore # Files and directories to be ignored by Git

├── mongodb.js # MongoDB connection and configuration

├── package-lock.json # Lockfile for the exact dependency tree

├── package.json # Dependencies and scripts for the server-side

├── server.js # Main server file (Express.js)

├── vercel.json # Configuration for deploying to Vercel

└── README.md # Documentation for the server-side

**תיק מתכנת :**

**1.Login -**

The `/Login` function processes login requests by validating user credentials against a MongoDB collection named . It first extracts the `username` and `password` from the request body and then queries the MongoDB database to find a user with the provided username. If a user is found, it checks if the provided password matches the stored password. If the credentials are correct, it responds with a success message and the user's ID; if not, it responds with an "Invalid credentials" message. If no user is found or if there is an error during the process, it sends an appropriate error message.

**Code :**

// Endpoint to handle user login requests

app.post('/Login', async (req, res) => {

const { username, password } = req.body;

try {

// Check if there is a user with the provided username

const user = await User.findOne({ username: username });

if (!user) {

// If no user is found, respond with a 'User not found' message

return res.status(200).json({ message: 'User not found' });

}

// Check if the provided password matches the stored password

if (user.password === password) {

// If the passwords match, respond with the user's ID and a 'Login success' message

return res.status(200).json({ \_id: user.\_id, message: 'Login success' });

} else {

// If the passwords do not match, respond with an 'Invalid credentials' message

return res.status(200).json({ message: 'Invalid credentials' });

}

} catch (err) {

console.error(err);

return res.status(500).json({ message: 'Server error' });

}

})

**2. Register-**

The `/Register` function handles user registration by validating new user details. It first extracts the `username`, `email`, `phone`, `password`, and `confirmPassword` from the request body. The function then checks if the username or email already exists in the MongoDB database. If either is found, it responds with an appropriate message. If both are unique, it inserts the new user data into the database and confirms successful registration. In case of any errors, it sends a "Server error" message.

**Code :**

// Endpoint to handle user registration requests

app.post('/Register', async (req, res) => {

const { username, email, phone, password, confirmPassword } = req.body;

try {

const check = await User.findOne({ username: username });

if (check) {

return res.status(200).json({ message: 'User already exists' });

}

// Check if the email already exists

const userByEmail = await User.findOne({ email: email });

if (userByEmail) {

return res.status(200).json({ message: 'Email already exists' });

}

else {

const data = {

username: username,

email: email,

phone: phone,

password: password

};

await User.insertMany([data]);

return res.status(200).json({ message: 'Registration success' });

}

} catch (err) {

console.log(err);

return res.status(500).json({ message: 'Server error' });

}

});

**3. NewsRemoveRequest from the favorite -**

The `/favorites/remove` endpoint handles the removal of a news item from a user's list of favorites. It starts by extracting the `userId` and `newsUrl` from the request body. The function then finds the user in the MongoDB database using the `userId`. If the user is not found, it responds with a "User not found" message. If the user exists, it filters out the news item with the specified `newsUrl` from the user's list of favorite news and saves the updated list. The function then responds with a success message indicating that the news has been removed from favorites. If an error occurs, it returns a "Server error" message.

**Code :**

// Endpoint to remove a news item from user's favorites

app.post('/favorites/remove', async (req, res) => {

const { userId, newsUrl } = req.body;

try {

const user = await User.findById(userId);//find the userId

if (!user) {

return res.status(200).json({ message: 'User not found' });

}

// filters the news ,it keeps all favorite news items except the one that matches the specified URL

user.favoriteNews = user.favoriteNews.filter(item => item.url !== newsUrl);

await user.save();

return res.status(200).json({ message: 'News removed from favorites' });

} catch (err) {

console.error(err);

return res.status(500).json({ message: 'Server error' });

}

});

**4. News Add Request for the favorite -**

The `/favorites/add` function handles requests to add a news item to a user's list of favorites. It starts by extracting the `userId`, `newsUrl`, and `newsDescription` from the request body. The function then searches the MongoDB database for the user with the provided `userId`. If the user is found, it checks whether the news item is already in the user's favorites. If the news item is not already favorited, it adds the news item to the `favoriteNews` array and saves the updated user document to the database. The function responds with a success message if the news is added, or an error message if there’s an issue with the request or database operation.

**Code :**

// Endpoint to add a news item to user's favorites

app.post('/favorites/add', async (req, res) => {

const { userId, newsUrl, newsDescription } = req.body;

try {

const user = await User.findById(userId);

if (!user) {

return res.status(200).json({ message: 'User not found' });

}

// Check if the news is already in the favorites

const existingFavorite = user.favoriteNews.find(item => item.url === newsUrl);

if (!existingFavorite) {

// Add the news item to the user's list of favorite news

user.favoriteNews.push({ url: newsUrl, description: newsDescription });

await user.save();

}

return res.status(200).json({ message: 'News added to favorites' });

} catch (err) {

console.error(err);

return res.status(500).json({ message: 'Server error' });

}

});

**5. Fetching news :**

The fetchNews function is used to retrieve news articles from the Currents API. It first constructs the API request URL based on whether a search query is provided. If a query is given, it fetches news articles related to the query; otherwise, it retrieves the latest news. The function then makes an HTTP GET request to the API using axios, handles any errors by logging them to the console, and returns the list of news articles. If an error occurs during the request, it returns an empty array.

**Code :**

import axios from 'axios';

// Your API key for accessing the Currents API

const API\_KEY = 'gpVtrC7Id8I4Y8fMLZLzVvfXBSRt5Qe\_akVVaUlIO5s6-VVd';

// Base URL for the Currents API

const BASE\_URL = 'https://api.currentsapi.services/v1';

/\*\*

\* Fetches news articles from the Currents API.

\*

\* @param {string} query - The search query for news articles. If empty, fetch the latest news.

\* @param {number} page - The page number for pagination.

\* @returns {Promise<Array>} - A promise that resolves to an array of news articles.

\*/

export const fetchNews = async (query = '', page = 1) => {

// Construct the API URL based on whether a search query is provided

const url = query

? `${BASE\_URL}/search?keywords=${query}&apiKey=${API\_KEY}&page=${page}` // URL for search query

: `${BASE\_URL}/latest-news?apiKey=${API\_KEY}&page=${page}`; // URL for latest news

try {

// Make the API request using axios

const response = await axios.get(url);

// Return the list of news articles from the response data

return response.data.news;

} catch (error) {

// Log any errors that occur during the request

console.error('Error fetching news:', error);

// Return an empty array if there's an error

return [];

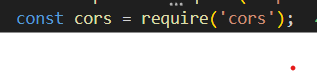
}

};

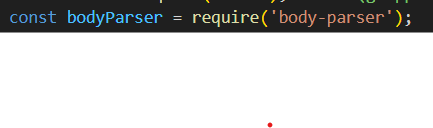
**ספריות שהשתמשנו -**

****

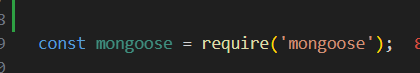
Express היא ספריית צד-שרת לפלטפורמת Node.js, המיועת לפשט את תהליך בניית אפליקציות אינטרנט ושירותי Express .

****

CORS הוא מאפיין אבטחה המיושם על ידי דפדפנים כדי להגביל את היכולת של דפים אינטרנטיים לבצע בקשות לדומיינים שונים מהדומיין שממנו נטען הדף.כאשר אתה מפעיל את cors, אתה מאפשר לשרת שלך לקבל בקשות מכתובות אתרים שונות (cross-origin requests), מה שמאפשר לקוחות (כמו דפדפנים) שמגיעים מדומיינים שונים לתקשר עם השרת שלך.



הספרייה body-parser ב-Node.js נועדה לנתח ולהמיר את התוכן של גוף הבקשות (request body) הנשלחות לשרת, כך שהמידע יהיה זמין לשימוש בקוד שלך. זו ספרייה פופולרית מאוד שנעשה בה שימוש באפליקציות Express כדי להקל על קריאת נתוני הבקשה.(מספקת פונקציות Middleware שמוסיפות לתוך אובייקט הבקשה (req) את המידע המפורש מהגוף של הבקשה בפורמטים שונים כמו JSON, URL-encoded, או Form-data. זה מאפשר לך לגשת בקלות למידע שנשלח מהלקוח בבקשות POST, PUT, PATCH, וכדומה.)



**mongoose** היא ספרייה עבור Node.js המהווה כלי נוח לעבודה עם MongoDB, מסד נתונים NoSQL פופולרי. Mongoose מספקת שכבת ממשק אובייקט-מסד נתונים (ODM - Object Data Modeling) שמסייעת למפתחים לתקשר עם MongoDB בצורה פשוטה ומסודרת. (mongoose מספקת גישה נוחה ומסודרת לעבודה עם MongoDB על ידי יצירת מודלים של נתונים וסקימות שמקלות על ניהול ומניפולציה של הנתונים. היא מסייעת בהפשטת התקשורת עם מסד הנתונים ומביאה עימה כלים נוחים לאימות ולניהול נתונים).

**API שהשתמשנו:**

**Search News**: To search for news articles based on keywords:

https://api.currentsapi.services/v1/search?keywords={query}&apiKey={API\_KEY}&page={page}

**Latest News**: To fetch the latest news:

https://api.currentsapi.services/v1/latest-news?apiKey={API\_KEY}&page={page}

**DB שהשתמשנו :**

'mongodb+srv://MarwaHamoud:Marwa2304@clusterweb.hnfso7d.mongodb.net/pro20'

**קטעי קוד מיוחדים שלקחנו ממקורות שונים (כולל רפרנסים):**

login/sign up /connect the server with the client

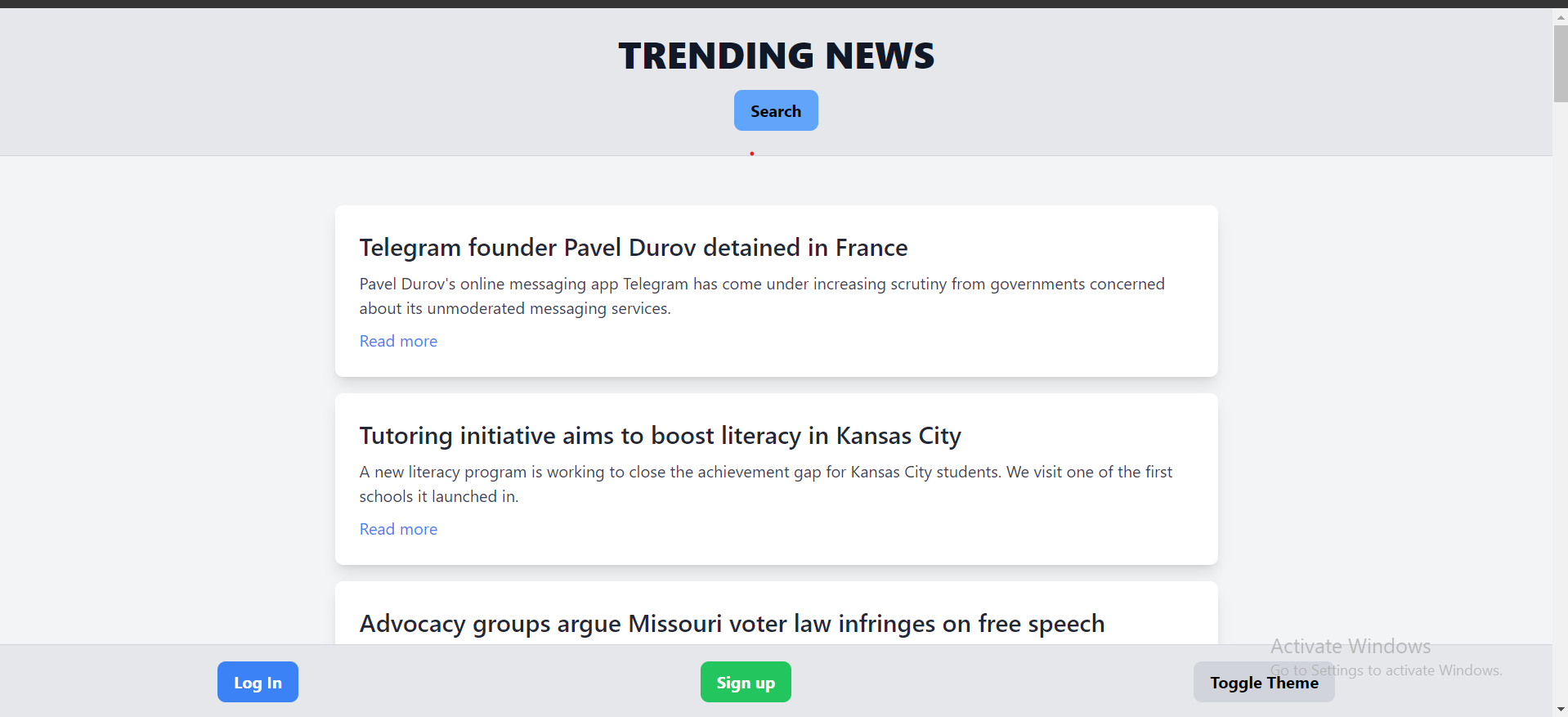
לקחנו אותם מ <https://github.com/MohammedDabbah> ושפרנו אותם והוספנו עיצוב דרך tailwind והתאמנו אותם לרציניו .

**שימוש בAI :**

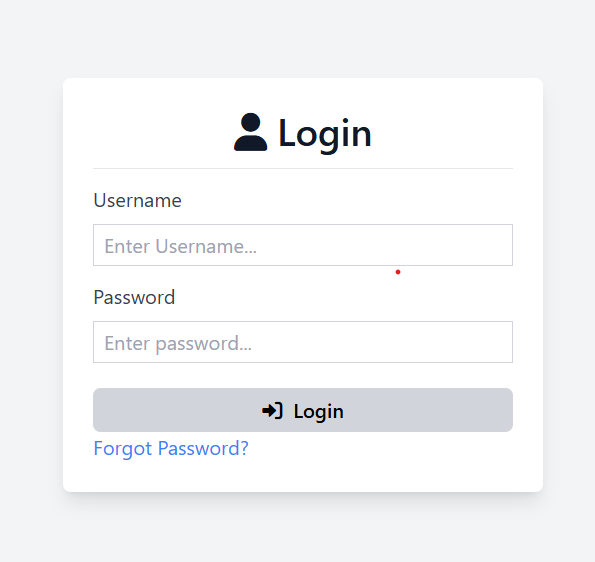
השתמשנו ב chat gpt כדי לשפר את הקוד שלנו מבחינת עיצוב , פונקציונאליות , לפתור כל מיני ERROR , וגם לקחנו קטעי קוד ……

**תיק משתמש :**

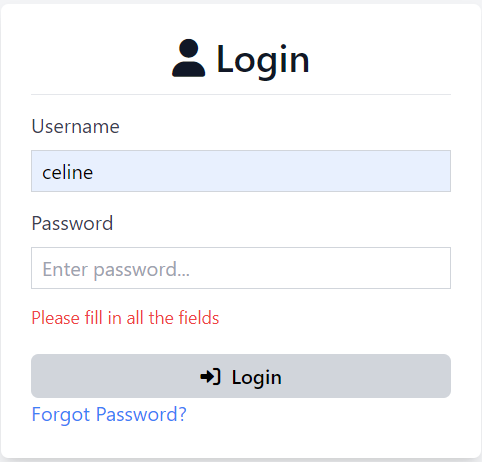
1. מסך ראשי: בעת הכניסה לאתר שלנו זה יהיה מסך המוצג למשתמשת יש לו 4 אפשרויות : להירשם כמשתמש חדש ,התחברות כמשתמש קיים , חיפוש חדשה מסויימת וכן אופציה לשינוי mood light/dark .



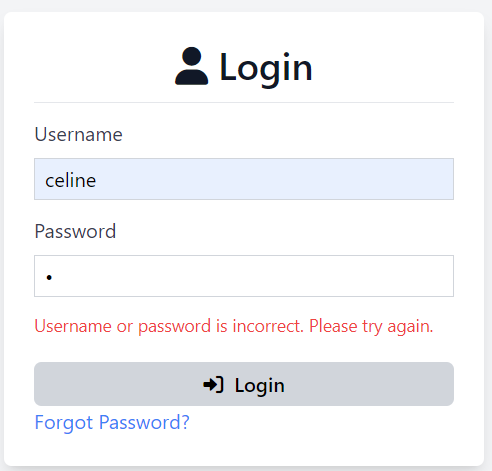
**2.login :**  יופיע המסך לאחר לחיצה על כפתור ה Log In , כאשר באחריות המשתמש להכניס פרטים תקינים אחרת לא יעבור למסך הפרופיל , וגם יש לו אופציה לאיפוס הסיסמה .



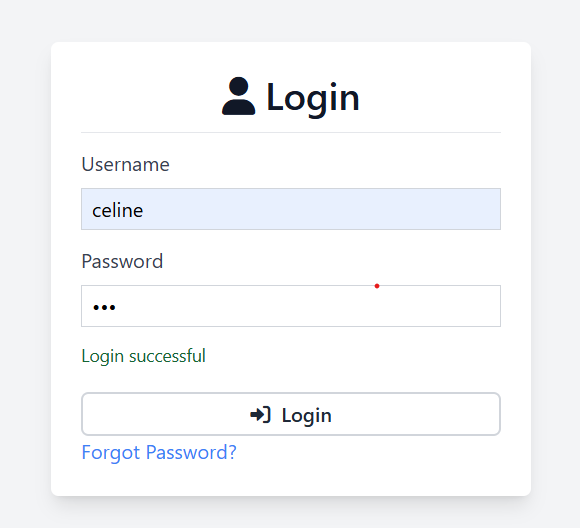
**\* הודעה שתוצג כאשר אחד מהפרטים חסרים :**



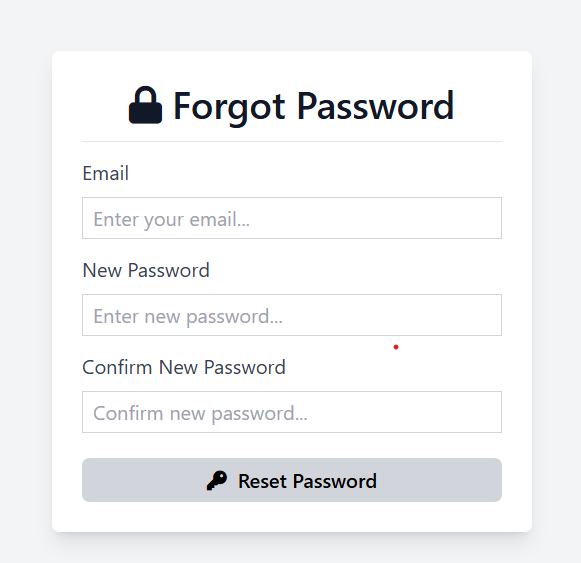
כאשר המשתמש מכניס ואחד הפרטים שגוי :



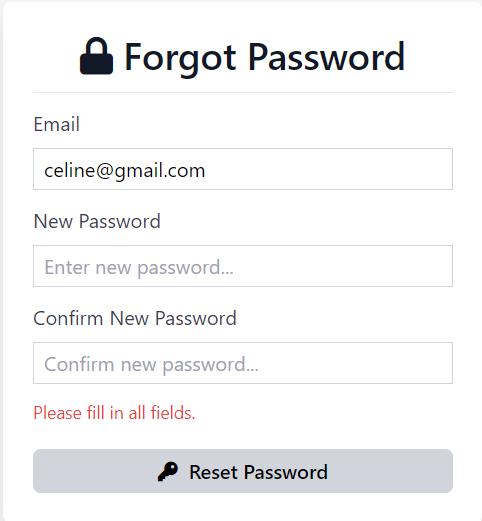
**\* הודעה שתוצג כאשר המשתמש כתב הכל תקין מוצג הודעת הצלחה :**

****

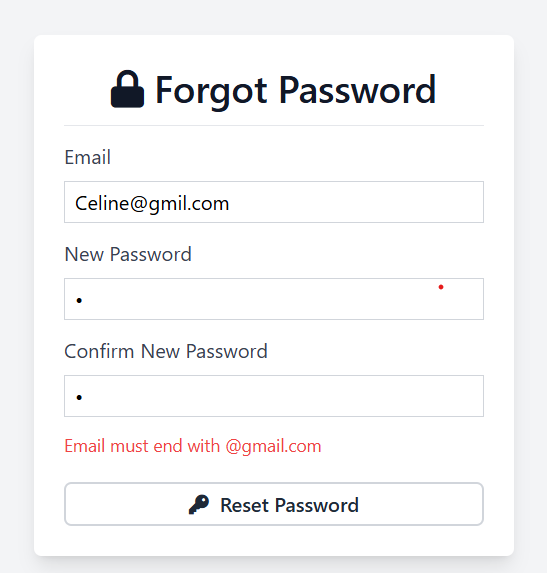
**3.forgot password :** יופיע המסך לאחר לחיצה על כפתור ה Forgot Password במסך ה Login , יש למשתמש אפשרות לאפס את הסיסמה לאחת חדשה, על המשתמש למלא את כל באופן חוקי

.****

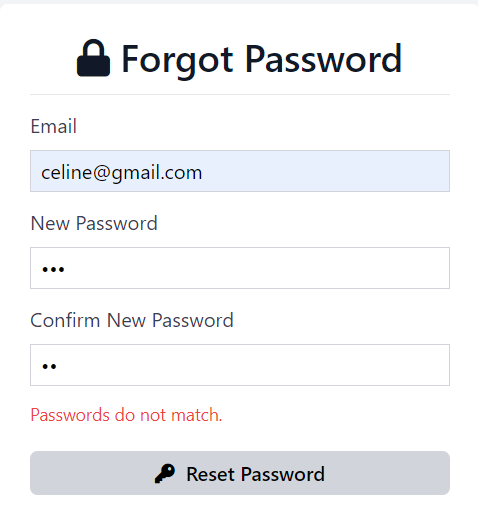
**מילוי חלקי לשדות:** תופיע הודעה שצריך למלא את כל השדות



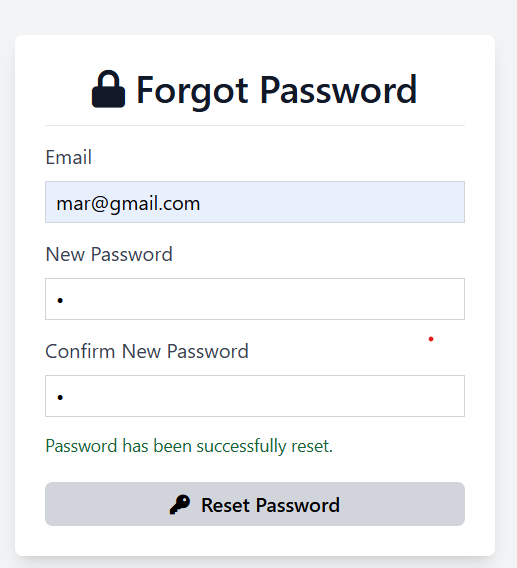
**מילוי לא חוקי של אחד השדות תופיע הודעת שגיאה : דוגמה להודעת שגיאה של מילוי אימייל :**



**כאשר מזין סיסמאות שונות : תופיע הודעה מתאימה**

****

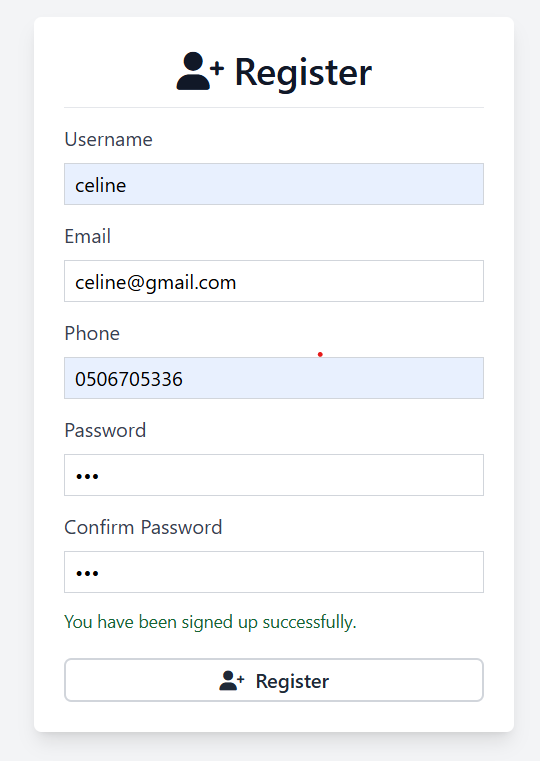
**\* כאשר מזין את הפרטים תקין מוצג הודעת הצלחה :**

****

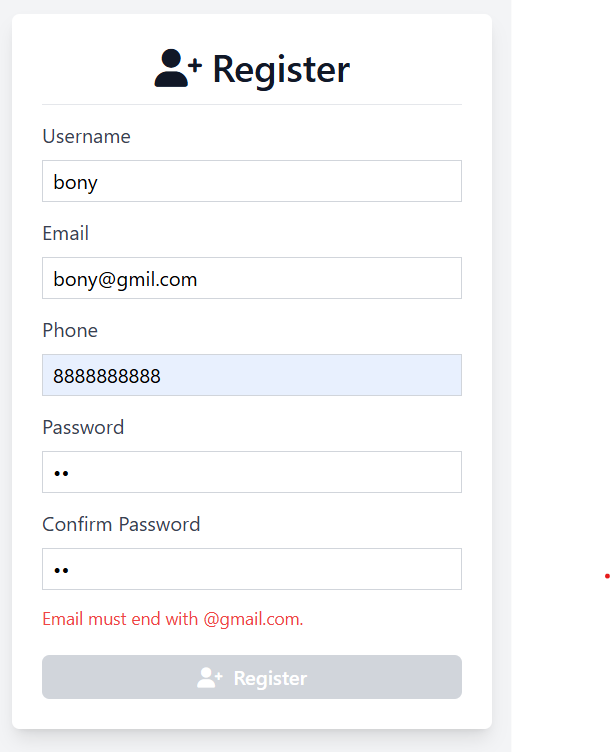
**4. מסך ההרשמה :** יופיע המסך לאחר לחיצה על כפתור ה sign up

על המשתמש למלא את כל השדות באופן תקין אחרת לא נרשם במערכת ואם כן הוא הצליח להירשם יופיע לו הודעה בהתאם:

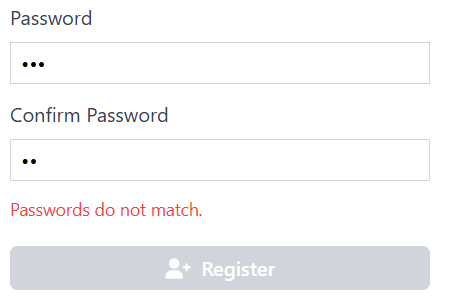
**מקרה הצלחה :**



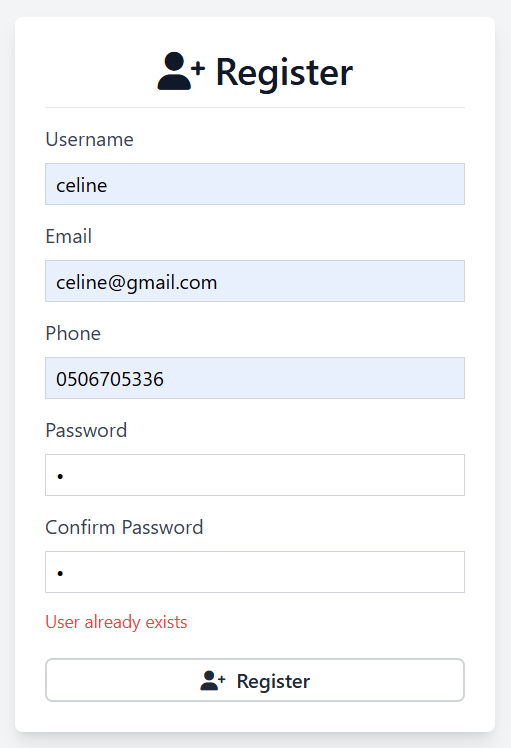
**\*מילוי לא חוקי של אחד השדות תופיע הודעת שגיאה : דוגמה להודעת שגיאה של מילוי אימייל :**



\***סיסמאות לא זהות בשתי השדות שקשורים לסיסמה :**

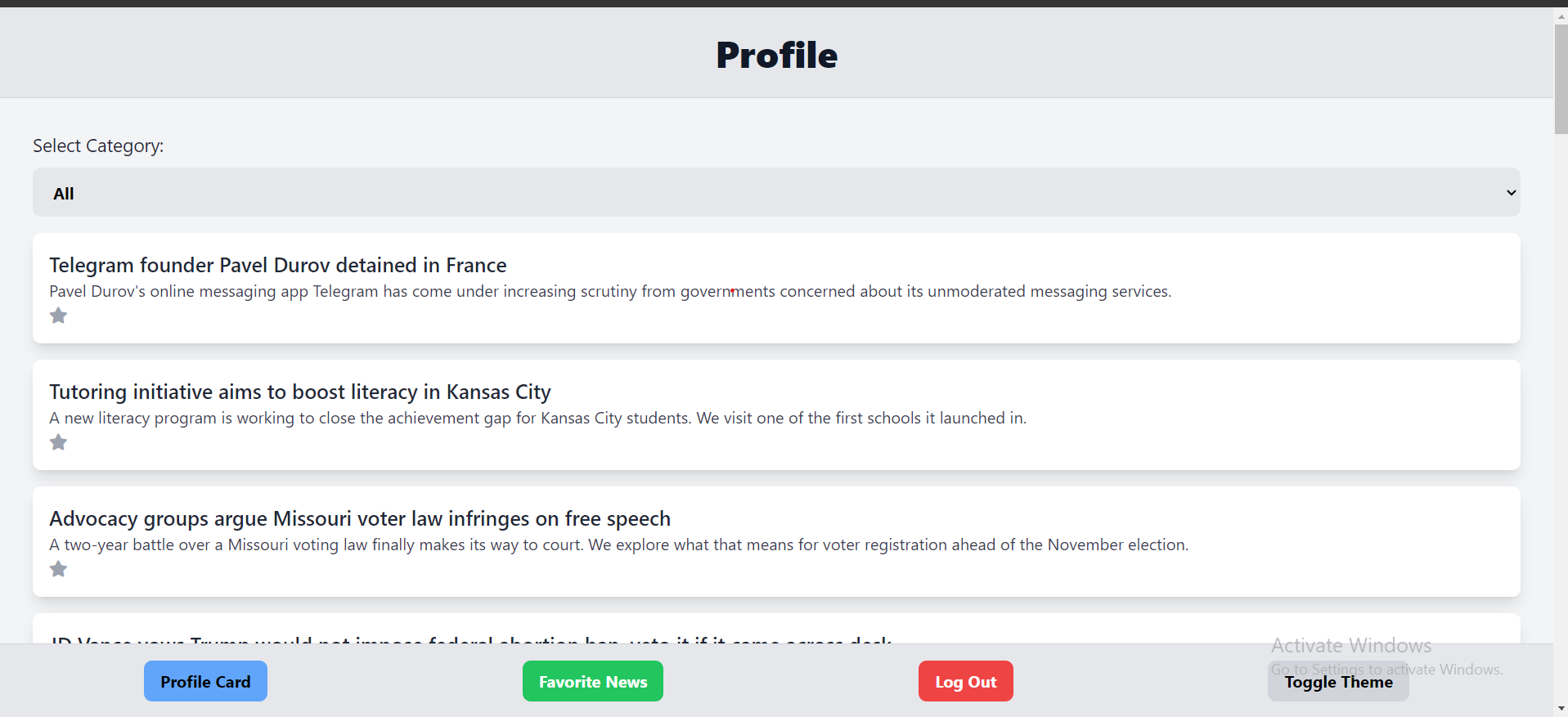


\* ה username כבר קיים :

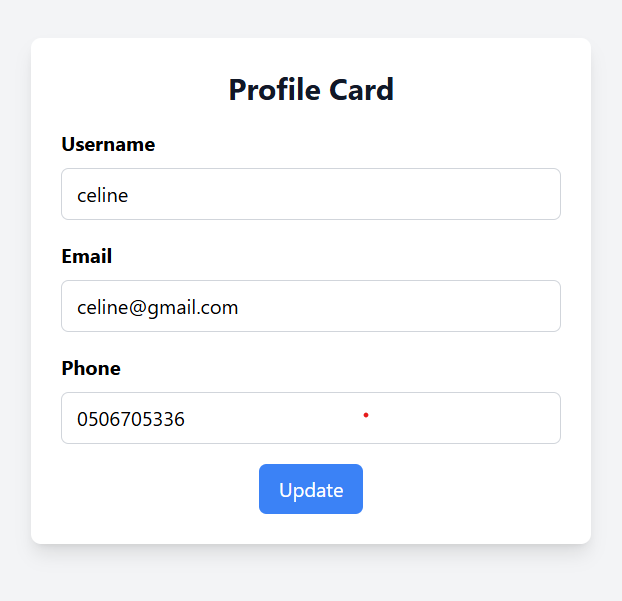


**5. מסך פרופיל משתמש :**

במסך הראשי של המשתמש יש לו הרבה אפשרויות שינוי mood dark/light שזה כפתור toggle theme בנוסף יש לו אופציה להיכנס לפרופיל שלו ולעדכן פרטים ובנסוף מתחת לכל חדשה יש כפתור כוכב בעת לחיצה עליו החדשה הזו תהפוך ל favorite והוא יכול לראות את כל החדשות המועדפים במסך favorite news ויכול לעשות התנתקות בלחיצה על LOGOUT **.**

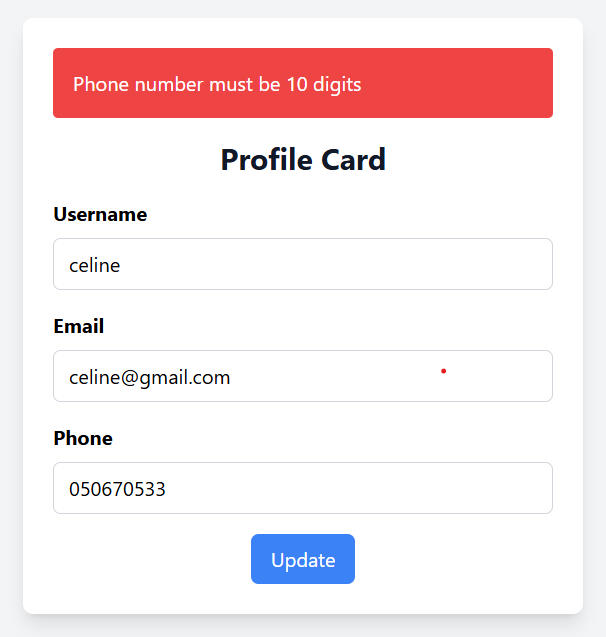


**6. profile card :** מסך שמציג את הפרטים של המשתמש : שם , אימייל, מספר טלפון .

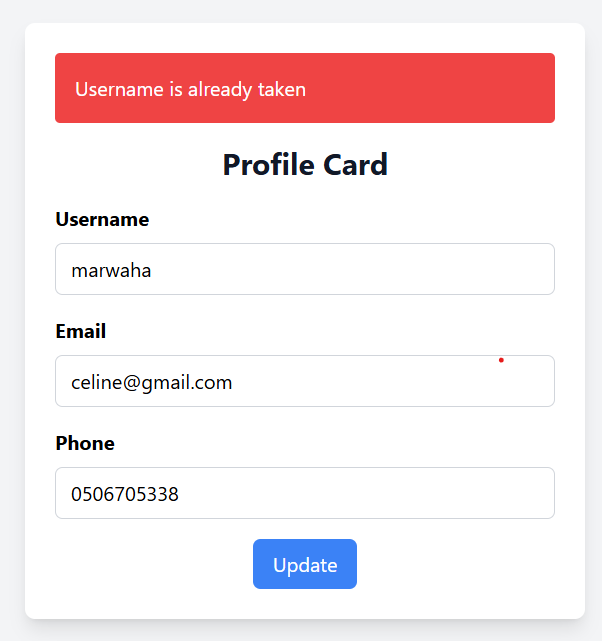


**\*כאשר המשתמש מעדכן אחד הפרטים לוחץ על UPDATE ומופיע הודעה בהתאם לחוקיות של השדות :**

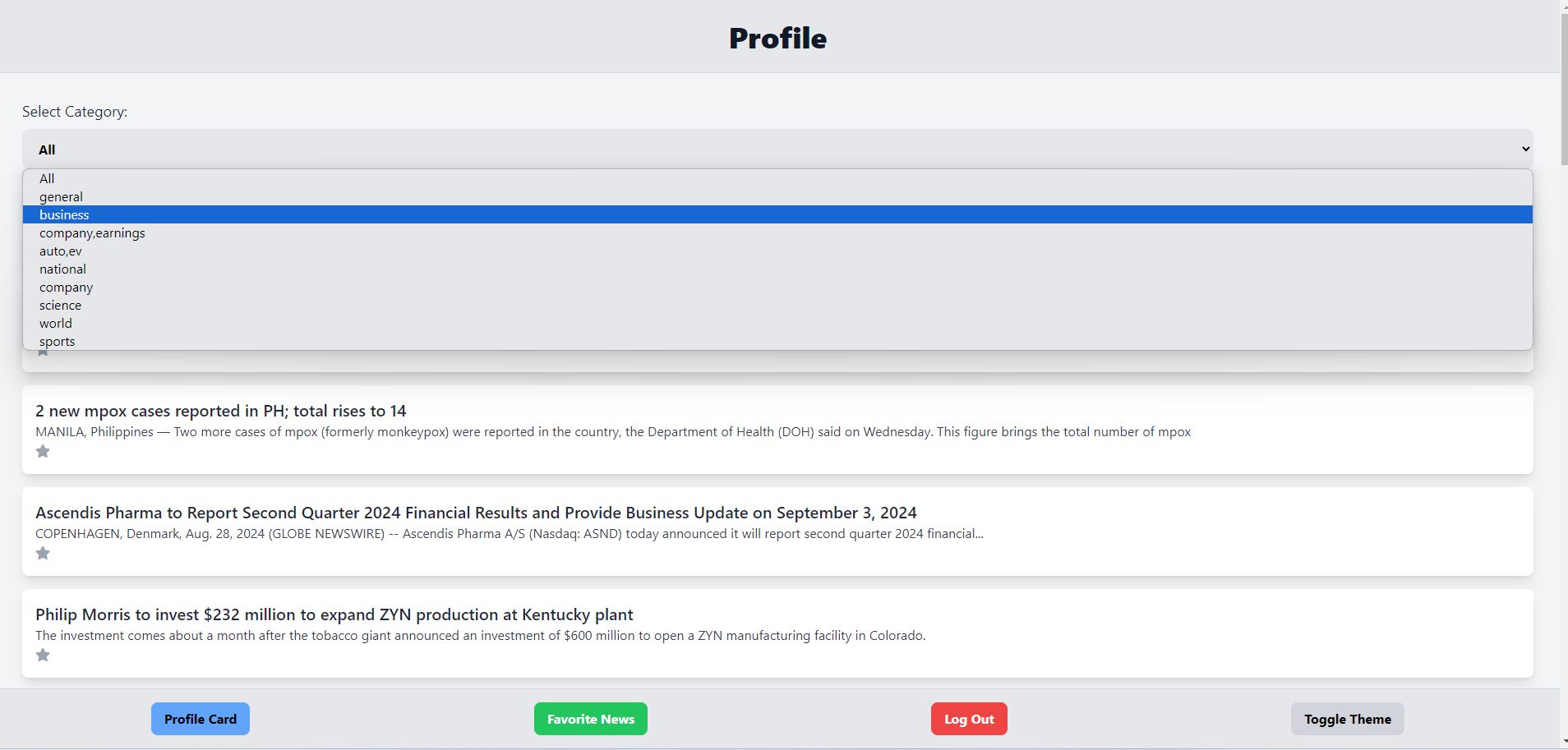
\* מספר טלפון לא תקין -



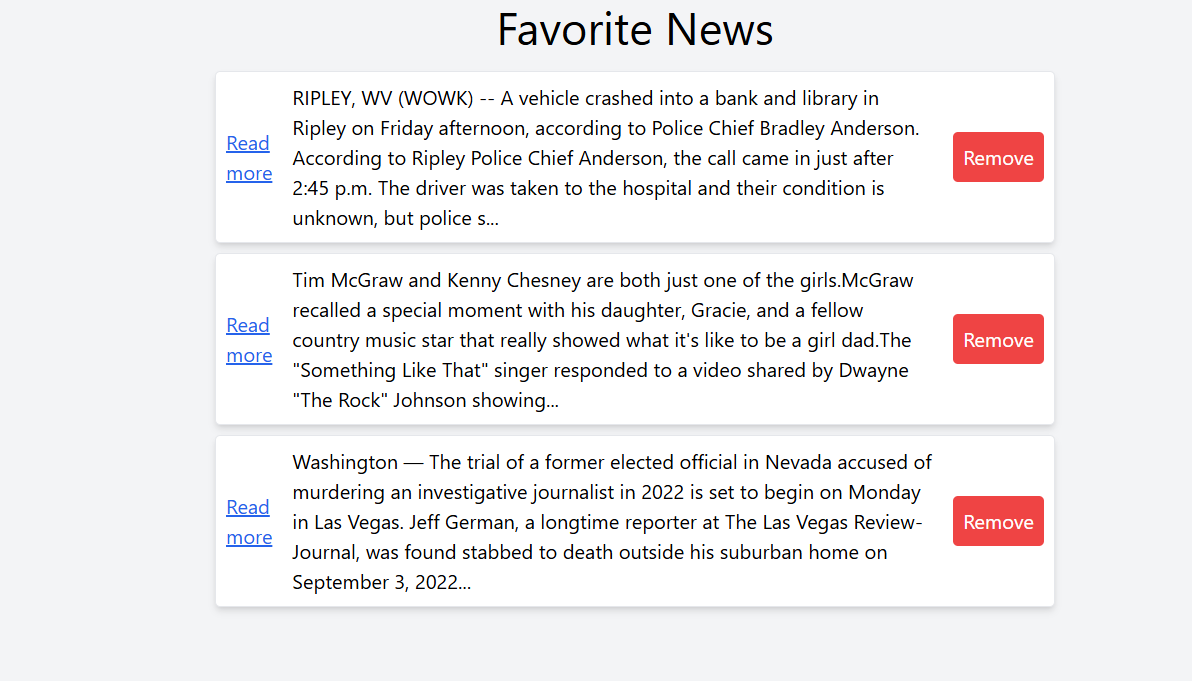
\* שם משתמש כבר קיים -



**7. יש אופציה שהמשתמש יכול לבחור קטגוריה מסוימת :**



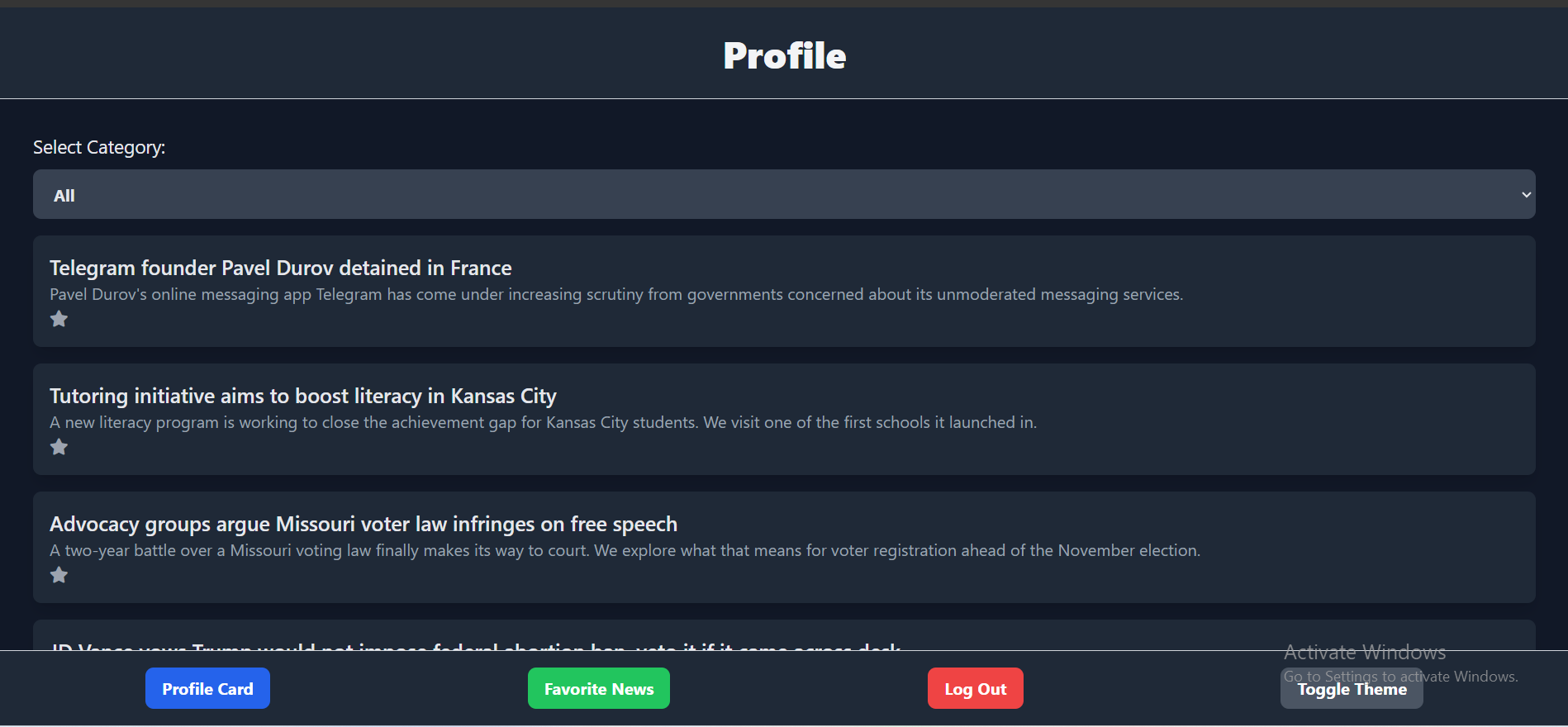
**8. מסך favorite news -** כאשר מציג את כל הקישורים ותיאור קצר לכל החדשות שהמשתמש סימן אותם בכוכב, יש אפשרות שהוא יסיר אחת החדשות מרשימת ה- favorite באמצעות כפתור remove .



9. המשתמש יכול לבחור את התצוגה של המסך שתהיה DARK/LIGHT דרך לחיצה על כפתור

Toggle Theme :

LIGHT :



יש להגיש את כל התוצרים בסביבת MTW :

* GIt repository - קישור (הריפו שפתחתם לצוות)
* Git Pages - קישור
* MoreThanWallet.com Gallery
* **הגשה בתיבת ההגשה במוודל – "הגשת פרויקט**"

להזכירכם , יש לבצע פריסה (deployment) ב - morethanwallet.com App Gallery

תוך שימוש ב- vercel.

הסבר על הגשה זו ניתן למצוא ב:

https://www.morethanwallet.com/appStore/gettingStarted

בנוסף, יש לוודא כי תיקיית ה- GIT ציבורית וכוללת את כל הקבצים של הפרויקט, כולל המסמכים, המצגת וכן הוראות הרצה.

**הנחיות הגשה:**

1.באחריותכם לוודא שהגשתם את כל התוצרים כנדרש ושהם הגיעו ליעדם.

לא תתאפשר בדיקה מחודשת של העבודה עקב טעויות בעת ההגשה!

2 .יש להגיש את התרגיל בקבוצות שהוגדרו בקורס.

3 .יש להגיש את התוצר הסופי של מסמך הפרויקט בפורמט WORD שייכלל בריפו של הפרויקט.

4.ניתן להפנות שאלות במייל לצוות הקורס, נא לשלוח שאלות לכלל הצוות . נא לציין את שם הקורס.