Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Colegiul Universității Tehnice a Moldovei

**Raport**

Lucrarea de practica

la „Programarea Procedurala”

**Elaborat de:  
 Grigoriev Artur, gr.PAPP-241**

# Ziua 1

## Analiza temei „X și Zero”

Jocul „X și Zero” reprezintă un concept simplu, dar atractiv, care a rezistat testului timpului, continuând să fie apreciat de utilizatorii de toate vârstele. Acesta constă în plasarea alternativă a simbolurilor „X” și „0” pe o grilă formată din trei rânduri și trei coloane, cu scopul de a realiza o linie verticală, orizontală sau diagonală compusă din simboluri identice. Deși regulile jocului par la prima vedere simple și ușor de înțeles, succesul în acest joc depinde în mod semnificativ de abilitățile strategice ale jucătorilor, determinându-i să anticipeze mișcările adversarului și să își adapteze propriile mutări în consecință.

În contextul actual, în care atenția utilizatorilor este limitată și competiția între aplicații este acerbă, dezvoltarea unui produs care să aibă succes implică înțelegerea aprofundată a mecanismelor ce atrag și mențin atenția utilizatorilor. În acest sens, tema „X și Zero” are ca obiectiv principal nu doar crearea unui simplu joc digital, ci și oferirea unei experiențe atractive și stimulante, în care utilizatorii să fie încurajați să se implice activ, folosindu-și logica și abilitățile cognitive într-un context ludic. Astfel, abordarea temei trebuie să depășească simpla redare a regulilor clasice și să ofere elemente care să îmbunătățească semnificativ experiența generală, inclusiv interfața intuitivă, o grafică plăcută și posibilitatea dezvoltării unor strategii variate care să prevină monotonia.

Totuși, simplitatea mecanică a jocului aduce cu sine și o provocare majoră, anume faptul că repetarea constantă a aceluiași model de joc poate conduce rapid la plictiseală și pierderea interesului din partea utilizatorului. Astfel, o implementare reușită a acestei teme trebuie să ia în considerare necesitatea introducerii unor elemente suplimentare care să evite monotonizarea experienței și să mențină implicarea jucătorului pe termen lung.

## Înțelegerea contextului și problemei abordate

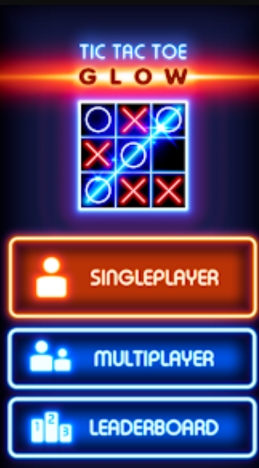
Jocul „X și Zero” poate fi încadrat într-un context digital modern dominat de numeroase aplicații care încearcă să capteze atenția utilizatorului și să-l păstreze implicat cât mai mult timp. În acest mediu competitiv, o simplă implementare a regulilor clasice nu mai este suficientă pentru a genera o audiență stabilă și loială. În acest sens, o problemă importantă identificată în cazul jocului „X și Zero” este caracterul repetitiv al gameplay-ului care, în lipsa unor mecanisme suplimentare, conduce rapid la plictiseală și, ulterior, la abandonarea aplicației.

Utilizatorii actuali caută experiențe variate și captivante, motiv pentru care o aplicație bazată strict pe regulile simple ale jocului poate deveni rapid neinteresantă. Provocarea majoră este deci aceea de a evita instalarea sentimentului de monotonie generat de repetarea constantă a aceluiași set limitat de acțiuni. Pentru a combate acest fenomen, dezvoltatorii trebuie să creeze mecanisme adiționale, cum ar fi niveluri diferite de dificultate, provocări tematice sau diverse moduri de joc care să îmbogățească experiența de bază și să determine utilizatorii să se reîntoarcă periodic.

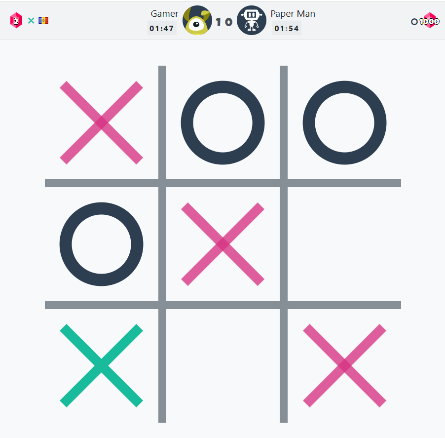
Mai mult decât atât, înțelegerea profundă a problemei înseamnă recunoașterea faptului că repetitivitatea este strâns legată de lipsa unor recompense sau stimulente adecvate. Astfel, introducerea unor obiective clare, recompense simbolice sau elemente de progresie în joc poate să ajute semnificativ la prevenirea plictiselii și să asigure o experiență plăcută și atractivă pe termen lung.

## Analiză Comparativă a Platformelor Existente

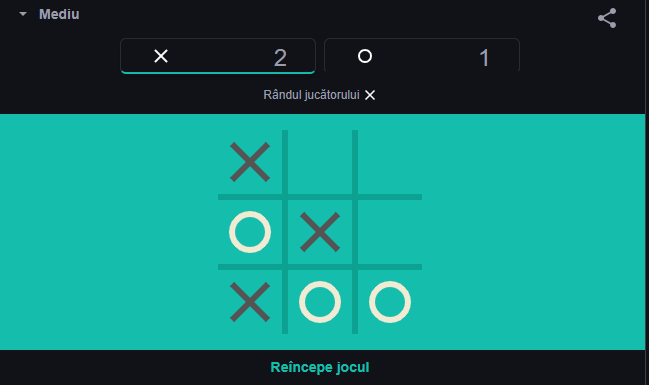
Pentru a înțelege mai bine oportunitățile și capcanele implementării temei „X și Zero”, este necesară o analiză detaliată a unor aplicații deja existente, identificând atât punctele lor forte, cât și slăbiciunile lor semnificative, în special cele legate de repetitivitatea gameplay-ului.



„**Tic Tac Toe Glow**” este o reinterpretare modernă și atrăgătoare a clasicului joc „X și Zero”, remarcându-se printr-o interfață grafică vibrantă și efecte luminoase neon, care sporesc atractivitatea vizuală și implicarea utilizatorilor. Aplicația oferă o diversitate semnificativă, incluzând atât moduri variate de joc, precum confruntarea cu un adversar AI sau cu alt jucător pe același dispozitiv, cât și multiple niveluri de dificultate potrivite tuturor tipurilor de jucători. În plus, aceasta propune grile extinse (5x5, 7x7, 9x9, 11x11) și un mod inovativ, numit „**Battle**”, care aduce o nouă dimensiune strategică jocului prin sisteme speciale de punctaj și confruntări personalizate. Astfel, „**Tic Tac Toe Glow**” depășește limita monotoniei tipice jocurilor din această categorie, stimulând rejucabilitatea și fidelizarea utilizatorilor.



A doua aplicație, **„Tic Tac Toe Online” (BYRIL)**, se remarcă prin claritatea regulilor și prin simplitatea interfeței. Deși este atractivă prin aspectul său minimalist și prin clasamentele existente, jocul nu oferă suficiente elemente care să evite instalarea monotoniei. Gameplay-ul devine repetitiv rapid, întrucât fiecare partidă este similară celei precedente, fără elemente distincte sau surprize care să susțină interesul utilizatorilor în timp. Lipsa recompenselor, provocărilor tematice sau elementelor de progresie contribuie și ea la diminuarea implicării pe termen lung.



Cea de-a treia platformă analizată este **„Google Tic Tac Toe”**, integrată direct în motorul de căutare Google. Acest joc beneficiază de o accesibilitate remarcabilă și de o interfață extrem de simplă și intuitivă, aspect care îl face atractiv pentru utilizatorii ocazionali. Cu toate acestea, tocmai această simplitate extremă generează o problemă majoră: gameplay-ul devine aproape imediat repetitiv, experiența limitându-se strict la aceleași acțiuni repetitive, fără variație sau recompense care să încurajeze implicarea continuă a utilizatorilor.

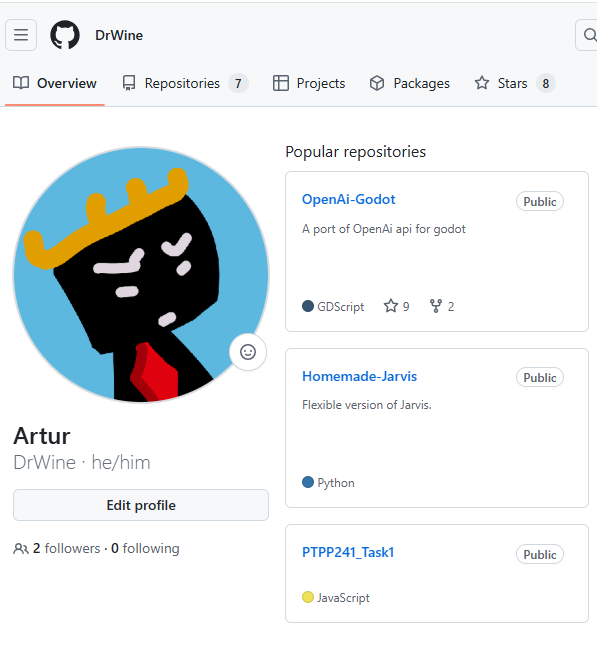
## Concluzie

Din analiza comparativă a acestor platforme rezultă clar că repetitivitatea gameplay-ului rămâne o provocare semnificativă, ce afectează negativ implicarea utilizatorilor în jocurile simple precum „**X și Zero**”. În timp ce aplicații precum „**Tic Tac Toe Online” (BYRIL)** și „**Google Tic Tac Toe**” suferă din cauza monotoniei rapide cauzate de lipsa elementelor diversificate, platforma „**Tic Tac Toe Glow**” se evidențiază pozitiv prin integrarea mai multor moduri de joc, grile extinse și mecanisme inovatoare care stimulează implicarea utilizatorilor și fidelizarea acestora pe termen lung. Prin urmare, dezvoltarea unei noi aplicații care să abordeze tema clasică „**X și Zero**” trebuie să preia exemplele pozitive oferite de „**Tic Tac Toe Glow**” și să continue în această direcție, punând accent pe diversificarea gameplay-ului, introducerea de obiective progresive și recompense virtuale, pentru a preveni monotonia și a susține interesul jucătorilor în mod constant și durabil.

## 

# Ziua 2 - Studierea Git-ului, Github

## Crearea cont de github



În cadrul activității dedicate familiarizării cu platforma GitHub, studentul a creat un cont personal pe site-ul oficial github.com, urmând pașii standard de înregistrare: completarea adresei de email, alegerea unui nume de utilizator, stabilirea unei parole și confirmarea contului prin e-mail. După autentificare, s-a explorat interfața platformei, evidențiindu-se elemente precum dashboard-ul, secțiunea de repository-uri și zona de profil. De asemenea, studentul a înțeles diferența dintre un repository public și unul privat, precum și importanța unui fișier README pentru descrierea proiectelor. Această etapă a avut ca scop nu doar familiarizarea cu platforma, ci și pregătirea pentru utilizarea practică a GitHub în procesul de dezvoltare colaborativă.

## Ce este Git, care sunt principiile de funcționare:

Git este un sistem de control al versiunilor care permite urmărirea modificărilor în fișierele unui proiect, facilitând munca în echipă și gestionarea eficientă a codului. Principiul principal al Git constă în păstrarea unui istoric complet al modificărilor, permițând revenirea la versiuni anterioare sau combinarea codului scris de mai mulți dezvoltatori.

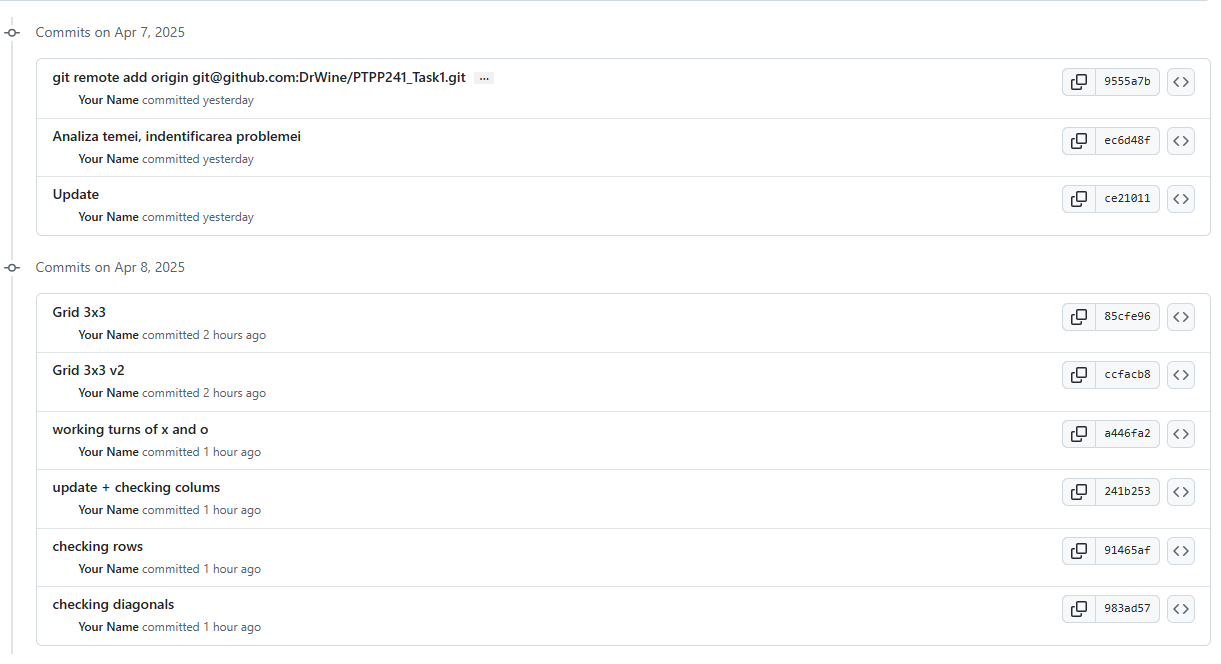
## Studierea comenzilor Git:

Pentru utilizarea eficientă a Git, este necesară învățarea comenzilor de bază precum git init, git add, git commit, git status și git log. Acestea permit inițializarea unui repository, adăugarea modificărilor și salvarea lor într-un istoric versionat. Înțelegerea acestor comenzi este esențială pentru gestionarea corectă a proiectelor.

## Conectarea dispozitivului local la contul GitHub:

Pentru a sincroniza codul local cu un repository de pe GitHub, este necesară conectarea între cele două prin comenzi precum git remote add origin și git push. Această legătură permite trimiterea modificărilor locale în spațiul online, asigurând o colaborare facilă și un backup permanent al codului.

## Concluzia



În cadrul lecției, studentul a aplicat cunoștințele despre Git și GitHub prin realizarea mai multor commituri care reflectă etapele dezvoltării aplicației mele. Au fost inițializate repository-ul și conexiunea cu GitHub, apoi au fost implementate funcționalități precum grila 3x3, alternarea tururilor și verificarea condițiilor de câștig pe linii, coloane și diagonale. Aceste commituri demonstrează o înțelegere practică a comenzilor Git și o organizare logică a progresului proiectulu

# Ziua 3 – Crearea unei pagini web.

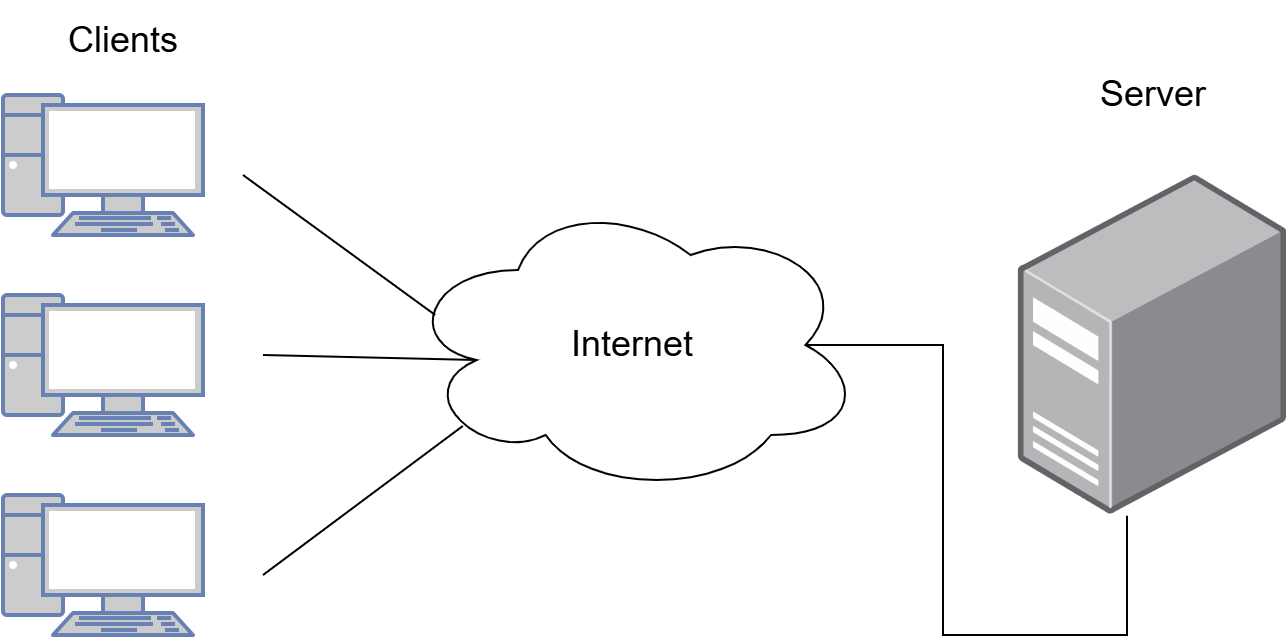


Fig. 1.0 Interacțiunea dintre client si server.

## Ce este html?

### **Interacțiunea dintre client și server:**

#### **Ce este un client, ce este un server și cum interacționează:**

Clientul este dispozitivul sau aplicația (de exemplu, un browser web) care trimite cereri pentru a obține date sau servicii. Serverul este calculatorul sau sistemul care răspunde la aceste cereri, oferind informații precum pagini web, fișiere sau rezultate din baze de date. Interacțiunea client-server funcționează astfel: clientul trimite o cerere către server, iar serverul procesează cererea și trimite înapoi un răspuns. Această comunicare poate fi vizualizată într-o schemă logică creată cu [draw.io](https://app.diagrams.net/?src=about), Fig 1.0.

#### **Ce este HTTP și HTTPS și de ce avem nevoie de ele:**

HTTP (HyperText Transfer Protocol) este protocolul standard folosit pentru comunicarea între client și server pe internet, permițând încărcarea de pagini web. HTTPS este versiunea securizată a acestuia, care criptează informațiile transmise pentru a preveni interceptarea datelor. Avem nevoie de ele pentru ca utilizatorii să poată accesa în siguranță informații de pe site-uri web. Ele stabilesc reguli clare pentru cum sunt formulate cererile și cum răspund serverele.

#### **Ce este un protocol și ce rol joacă TCP și UDP:**

Un protocol este un set de reguli care definește modul în care două sau mai multe dispozitive comunică într-o rețea. TCP (Transmission Control Protocol) este un protocol fiabil care asigură că toate datele sunt livrate în ordine și fără pierderi – ideal pentru aplicații web. UDP (User Datagram Protocol) este mai rapid, dar nu garantează livrarea completă a datelor – fiind preferat în aplicații unde viteza e mai importantă decât acuratețea, cum ar fi streamingul video sau jocurile online.

#### **Ce este un port și ce porturi folosesc aceste protocoale:**

Un port este o poartă virtuală prin care comunicațiile de rețea sunt direcționate către aplicația corectă de pe un dispozitiv. Protocoalele folosesc porturi standard pentru a stabili conexiuni: HTTP folosește portul 80, iar HTTPS folosește portul 443. TCP și UDP pot utiliza orice port disponibil (0–65535), dar pentru HTTP și HTTPS se folosesc în mod tradițional porturi TCP 80 și 443. UDP este adesea folosit pe porturi precum 53 (pentru DNS), 67-68 (pentru DHCP) sau 123 (pentru NTP).

### **Arhitectura aplicației web-based:**

#### **Ce este o aplicație web-based și care sunt componentele ei principale:**

O aplicație web-based este un software accesibil printr-un browser, fără a necesita instalare locală. Aceasta este compusă din trei componente principale: frontend (interfața utilizatorului), backend (logica și procesarea datelor) și baza de date (stocarea informațiilor). Toate aceste părți colaborează pentru a oferi utilizatorului o experiență completă.

#### **Care sunt avantajele unei arhitecturi web:**

Arhitectura web oferă acces ușor de pe orice dispozitiv cu internet, fără instalări suplimentare. Actualizările sunt simple și rapide, iar întreținerea aplicației se face centralizat, pe server. De asemenea, permite scalarea ușoară și integrarea cu diverse servicii externe.

#### **Diferența dintre frontend și backend într-o aplicație web:**

Frontend-ul este partea aplicației pe care o vede și o folosește utilizatorul – include designul și interacțiunile vizuale, realizate cu HTML, CSS și JavaScript. Backend-ul este partea invizibilă care rulează pe server și gestionează logica aplicației, conexiunile la bazele de date și răspunsurile la cererile frontend-ului.

#### **Ce este o arhitectură monolitică și ce este una pe microservicii:**

Arhitectura monolitică presupune că toate componentele aplicației sunt într-un singur bloc de cod. Este mai simplă de implementat inițial, dar dificil de extins sau întreținut. Arhitectura pe microservicii împarte aplicația în servicii mici și independente, fiecare cu rolul său, fiind mai flexibilă și ușor de scalat.

### **Limbajul HTML (directive uzuale, ancore, imagini, culori fundal, meta):**

HTML este limbajul de bază pentru crearea și structurarea paginilor web, folosind directive standard precum titluri, paragrafe și liste. Ancora (<a>) permite conexiunea între pagini, iar imaginea (<img>) îmbogățește vizual conținutul. Culorile și fundalul pot fi controlate prin CSS pentru o prezentare atractivă, iar etichetele meta furnizează informații utile motoarelor de căutare.

### **Formularele în HTML, tabelele, cadrele (frames):**

Formularele HTML permit utilizatorilor să introducă și să trimită informații către server prin elemente precum câmpuri text și butoane de trimitere. Tabelele sunt utilizate pentru organizarea datelor în mod clar și structurat. Cadrele (frames) permit afișarea simultană a mai multor documente HTML, dar utilizarea acestora este descurajată în prezent din motive legate de accesibilitate și optimizare SEO.

# ****Ziua 4 CSS****

## ****Ce este CSS?****

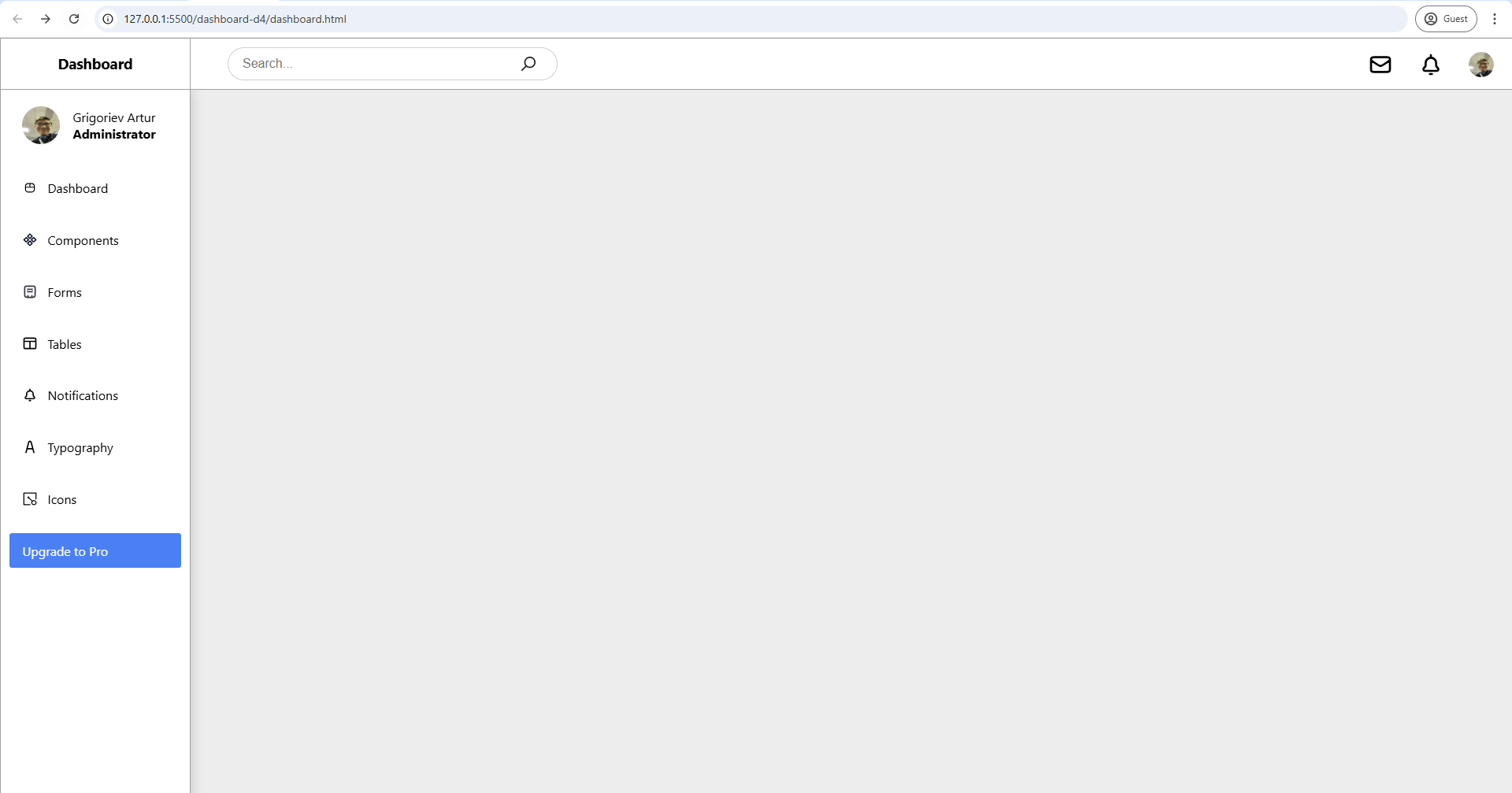
**a. Sintaxa CSS**  
Sintaxa CSS este modul în care sunt scrise regulile de stilizare pentru paginile web. Fiecare regulă CSS este compusă dintr-un „selector” și un „bloc de declarații”. Selectorul identifică elementul HTML care va fi stilizat, iar blocul de declarații conține una sau mai multe perechi de proprietăți și valori, separate prin punct și virgulă. De exemplu, p { color: red; font-size: 16px; } aplică un text roșu și o dimensiune de 16px tuturor paragrafelor.

**b. CSS text (font, weight, color)**  
CSS oferă o gamă largă de proprietăți pentru formatarea textului. Printre cele mai comune se numără font-family pentru alegerea fontului, font-weight pentru grosimea textului (de exemplu bold sau normal), și color pentru culoarea textului. Acestea permit designerilor web să personalizeze aspectul textului pentru a se potrivi cu identitatea vizuală a site-ului.

**c. CSS table**  
Stilizarea tabelelor în CSS permite un control detaliat asupra aspectului rândurilor, celulelor și bordurilor. Proprietăți precum border, padding, text-align și background-color sunt frecvent utilizate pentru a face tabelele mai lizibile și mai atractive vizual. De asemenea, se pot controla marginile dintre celule cu border-collapse și spațierea celulelor cu border-spacing.

**d. CSS border**  
Proprietatea border este folosită pentru a adăuga margini în jurul elementelor HTML. Se pot seta dimensiunea, stilul (solid, dashed, dotted etc.) și culoarea bordurii. De exemplu, border: 2px solid black; va aplica o margine neagră solidă de 2px. Este esențială pentru delimitarea clară a elementelor în designul interfeței.

**e. CSS list**  
CSS permite stilizarea listelor (ordonate și neordonate) prin proprietăți precum list-style-type, list-style-position, și list-style-image. Acestea controlează felul în care apar marcatorii listelor (buline, cifre, imagini personalizate) și poziționarea lor. Astfel, se pot crea liste care să se integreze armonios în estetica generală a paginii.



În lucrul cu CSS, GPT m-a ajutat să lucrez mai rapid și mai eficient, mai ales în taskuri repetitive sau tehnice. De exemplu, atunci când am avut nevoie să convertesc valori precum 1.4rem în pixeli sau 15vw în rem, am obținut imediat rezultatul fără să pierd timp cu calcule sau verificări externe. În structura dashboard-ului, alinierea div-urilor a fost optimizată rapid cu recomandări clare — cum ar fi folosirea lui display: flex, împreună cu justify-content: center și align-items: center, pentru a centra elemente ca #corner.

Un alt exemplu a fost eliminarea redundanțelor în stiluri. Când am observat linii CSS repetitive, cum ar fi:



GPT m-a ajutat să le identific și să le reorganizez eficient în clase reutilizabile. Nu a fost vorba de „cum să fac”, ci de a face mai repede și mai curat. Practic, GPT a devenit un instrument de accelerare a deciziilor și un filtru instant de soluții, fără să mai pierd timp pe documentație sau încercări inutile.

# Ziua 5

# 

**Ce este JavaScript?**  
JavaScript este un limbaj de programare care permite adăugarea de interactivitate pe paginile web. Este folosit pentru a dinamiza conținutul care altfel ar fi static.

**Unde se folosește?**  
JavaScript se folosește în principal în browser, dar și pe server cu platforme precum Node.js. Este peste tot unde ai nevoie de logică în web.

**Cum se folosește (browser, consola, fișiere .js)?**  
Codul JavaScript se poate scrie direct în browser, în consola de dezvoltator, sau în fișiere cu extensia .js care sunt apoi legate în pagina HTML.

**Cum scriem primele linii de cod?**  
Primele linii pot fi scrise direct în consola browserului. E de ajuns un console.log() pentru a afișa text.

**Crearea unui folder pentru proiect**  
Organizarea începe cu un folder dedicat unde se salvează toate fișierele (HTML, CSS, JS). Un spațiu curat și logic e esențial.

**Crearea fișierelor (index.html, style.css, scripts.js)**  
Se creează cele 3 fișiere: index.html pentru structură, style.css pentru stil, scripts.js pentru logică. E o convenție de bază.

**Interconectarea fișierelor**  
Fișierele se leagă între ele prin HTML: CSS-ul cu <link> și JavaScript-ul cu <script>. Totul trebuie să fie conectat ca să funcționeze corect.

# ****Ziua 6****

**Structura HTML și Interfața Utilizator (UI):**  
A fost definită structura fundamentală a paginii de autentificare, punându-se accent pe separarea clară a celor două secțiuni – Login și Register – prin tab-uri interactive. Fiecare formular a fost conceput cu câmpuri relevante și elemente auxiliare pentru feedback-ul utilizatorului, cum ar fi etichete, mesaje de validare și un sistem de comutare între stări și pași. Înregistrarea a fost împărțită într-un proces în trei etape, fiecare cu propriile formulare și controale de navigare. S-a urmărit asigurarea unei experiențe intuitive pentru utilizator, deși totul rămâne la nivel client-side.

**Stilizare CSS:**  
Elementele HTML au fost completate de o componentă vizuală coerentă, implementată în CSS, ce conferă aplicației un aspect modern și profesionist. S-au adăugat stilizări pentru toate componentele interactive, inclusiv tab-uri, câmpuri, butoane și indicatori de progres, utilizând o paletă clar definită și o tipografie uniformă. S-au prevăzut și ajustări minime pentru ecrane mici, însă fără o abordare responsive completă. Efortul a fost direcționat spre aspect vizual și claritate, nu neapărat spre performanță sau accesibilitate avansată.

**Logica JavaScript (auth.js):**  
S-a implementat o logică JavaScript destul de amplă pentru controlul interfeței și validarea datelor introduse de utilizatori. Deși sunt prezente funcții de validare în timp real și logica de comutare între tab-uri și pași, totul este executat local, fără vreo comunicare reală cu un backend. Tăria parolei este evaluată dinamic, iar feedback-ul este livrat instant utilizatorului. Este important de menționat că, în absența unei protecții reale sau criptări, toate aceste funcționalități sunt exclusiv de test și nu pot fi utilizate într-un mediu de producție.

**Logica de Login:**  
Mecanismul de autentificare a fost realizat printr-o simulare bazată pe cookie-uri, o soluție complet nesigură dar acceptabilă pentru debugging. Codul validează datele introduse și compară credențialele cu o listă fictivă stocată local. În caz de succes, se setează un cookie de status și se simulează o redirecționare. Deși funcțional, acest sistem este lipsit de orice standard de securitate și nu ar trebui sub nicio formă expus utilizatorilor reali. E un exercițiu de interfață, nu un proces de autentificare viabil.

**Logica de Înregistrare:**  
Fluxul de înregistrare este împărțit logic și validat secvențial, cu funcționalități specifice fiecărui pas. Cu toate acestea, salvarea datelor se face local, în cookie-uri, fără criptare și fără protecție împotriva duplicatelor sau atacurilor. Adăugarea unui nou cont în cookie este tratată ca o reușită, fără nicio verificare reală. E o implementare pur demonstrativă, care expune toate limitele abordărilor client-side și subliniază urgența migrării către o soluție server-side reală.

# Ziua 7

**1. Crearea unor funcții clare pentru fiecare acțiune:**  
Am dezvoltat funcții bine definite pentru acțiuni specifice în aplicație. De exemplu, funcția CreateSwitch se ocupă exclusiv de gestionarea stării unui switch (pornit/oprit), fără a încorpora logică inutilă sau responsabilități suplimentare, menținând astfel claritatea și scopul codului.

document.addEventListener('DOMContentLoaded', function () {

    const debugLoginStatus = document.cookie

        .split('; ')

        .find(row => row.startsWith('debug\_login\_status='))

        ?.split('=')[1];

    const path = window.location.pathname;

    console.log(path);

    if (path !== '/auth.html' && debugLoginStatus !== 'loggedIn') {

        window.location.href = 'http://localhost:5500/auth.html';

    }

    console.log(debugLoginStatus);

});

**2. Separarea logicii în pași simpli:**  
Codul este structurat logic în pași clari și ușor de urmărit. Evenimentul DOMContentLoaded tratează verificarea stării de autentificare și redirecționarea, în timp ce funcțiile auxiliare precum getCookie și deleteCookie sunt izolate pentru a simplifica fiecare operațiune individuală.

function getCookie(name) {

    const cookieArr = document.cookie.split(";");

    for (let cookie of cookieArr) {

      cookie = cookie.trim();

      if (cookie.indexOf(name + "=") === 0) {

        return decodeURIComponent(cookie.substring(name.length + 1));

      }

    }

    return null;

  }

function deleteCookie(cookieName) {

    document.cookie = cookieName + "=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";

  }

**3. Scrierea unui cod reutilizabil:**  
Am evitat duplicarea logicii prin crearea de funcții reutilizabile precum getCookie, deleteCookie și CreateSwitch. Acestea pot fi apelate ori de câte ori este nevoie, asigurând astfel consistență în comportament și facilitând întreținerea codului pe termen lung.

function CreateSwitch(selfId, btnIdOn, btnIdOff, inputId){

    let btn1 = document.getElementById(btnIdOn);

    let self = document.getElementById(selfId);

    let btn2 = document.getElementById(btnIdOff);

    let input = document.getElementById(inputId);

    if (selfId === btnIdOn){

      input.value = 'on';

      btn1.classList.add('btn-primary');

      btn2.classList.remove('btn-primary');

    } else {

      input.value = 'off';

      btn2.classList.add('btn-primary');

      btn1.classList.remove('btn-primary');

    }

  }

# ****Ziua 8****

**Învață localStorage.setItem(), getItem()**  
localStorage.setItem() și localStorage.getItem() sunt metode fundamentale pentru a salva și a prelua date în browser. Acestea permit stocarea de informații care persistă și după închiderea paginii, fiind utile pentru salvarea preferințelor sau datelor introduse de utilizator.

**Folosește JSON.stringify() și JSON.parse() pentru array-uri**  
Pentru a salva array-uri sau obiecte complexe în localStorage, e obligatoriu să folosești JSON.stringify() pentru a le converti în string. La extragere, JSON.parse() reface obiectul sau array-ul original. Fără acest pas, datele complexe nu pot fi manipulate corect.

**Creează funcții pentru salvarea și extragerea datelor din localStorage**  
În loc să scrii de fiecare dată localStorage.setItem și localStorage.getItem, creezi funcții precum saveData() și getData() pentru a centraliza logica. În codul tău, ele ajută la gestionarea conturilor și a stării de login, păstrând codul clar, reutilizabil și ușor de întreținut. Dacă schimbi metoda de stocare, modifici doar aceste funcții, nu tot proiectul.

  const saveData = (key, value) => {

      localStorage.setItem(key, JSON.stringify(value));

  };

  const getData = (key) => {

      let data = localStorage.getItem(key);

      try {

          return data ? JSON.parse(data) : null;

      } catch (e) {

          return null;

      }

  };