**Thobor**

Elev, Indrumător,

         Drăguțu Matei prof.  Adriana Ciocănel

**Documentație non-tehnică**

Thobor este singura echipă de robotică din Tecuci de la Colegiul Național “Calistrat Hogaș” și concurează la competiția BRD FIRST TECH CHALLENGE (FTC) unde s-a calificat în fiecare an la etapa națională și a obținut premii importante.

Site-ul echipei, disponibil pe domeniul thobor.ro, este structurat atât drept un site de prezentare, cât și o platformă pentru membrii cu scopul de a le spori productivitatea în cadrul sezonului. Astfel, utilizatorii ce vor doar să afle informații despre echipă, au la dispoziție paginile din navbar (home, blog, despre, shop, apps, team, sponsors, simulator și robotul din sezonul actual).

**Pagina “Home” conține:**

* un scurt istoric în cifre al echipei (anii de experiență, numărul participărilor laetapa națională, membrii activi și alumni)
* o scurtă descriere a echipei
* premiile obținute la etapele regionale și naționale la care a participat echipa

**Pagina “Blog” conține:**

* cele mai importante activități organizate de Thobor sau la care a participat echipa
* fiecare postare conține o descriere, un carusel de poze și 2 butoane ce trimit utilizatorul către postarea de pe facebook, respectiv de pe Instagram încurajând userul să le aprecieze cu scopul de a ne crește reach-ul pe rețele de socializare

**Pagina “Despre” conține:**

* informații detaliate despre echipă și colegiul la care învață membrii
* informații despre educația STEM pe care o promovează echipa (Educația STEM = Science, Technology, Engineering și Mathematics)

**Pagina “Shop” conține:**

* materialele promoționale pe care Thobor le oferă publicului.
* se pot filtra și sorta în funcție de preferințe
* fiecare material are o descriere scurtă, lungă, informații tehnice, poze, preț, o secțiune de reviews ce calculează ratingul în funcție de numărul steluțelor din fiecare review și o listă cu produse din categorie

**Pagina “Apps” conține:**

* toate aplicațiile dezvolatate de departamentul de programare al echipei (2 scavanger hunt-uri și o aplicație pentru android care scanează codul QR și pe ecranul telefonului apare robotul 3D)

**Pagina “Team” conține:**

* fiecare generație care a făcut parte din echipă are un carusel în care apar membrii din anul respectiv

**Pagina “Sponsors” conține:**

* logo-urile sponsorilor care ne-au ajutat pe parcusul sezoanelor financiar sau cu materialele necesare

**Pagina “Simulator” conține:**

* un simulator care replică jocul sezonului actual folosind conceptul robotului echipei Thobor

**Pagina “Robotul din sezonul 6” conține:**

* descrierea tehnică a robotului din ultimul sezon
* când se dă scroll, fiecare mecanism important este vizualizat + informații despre el

Pentru membrii echipei Thobor a fost creată o pagină de administrator unde poate fi controlat conținutul dinamic al site-ului.

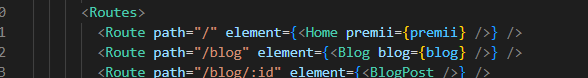
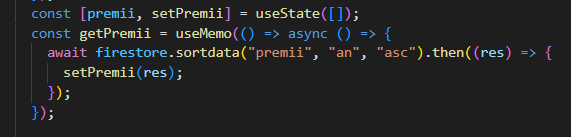
De asemenea, pentru a spori productivitatea, a fost adăugat un CRM custom unde userii cu rolul de admin, mentor, alumni și lider de departament pot adăuga taskuri pe care ceilalți membrii le pot prelua și încărca rezolvarea.

Doar userii cu rolul de admin, mentor, alumni și lider de departament pot crea sedințe online în care share-uiesc ecranul, codează împreună sau creează scheme organizatorice pe tablă.

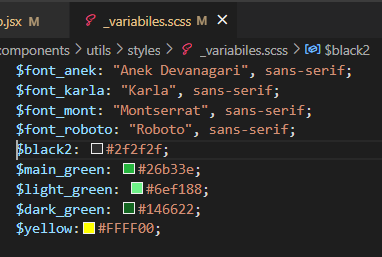
**Documentație tehnică**

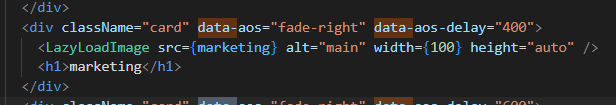
Pentru realizarea site-ului am folosit react.js și firebase. Am optat pentru o bază de date non-sql întrucât fiecare postare are un număr inegal de poze și paragrafe. Un principiu universal în cadrul site-ului este manipularea locală a conținutului care funcționează astfel:

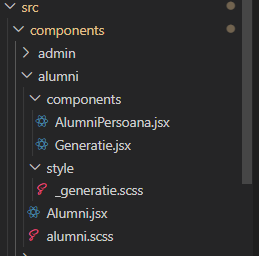
* datele sunt preluate din baza de date in fișierul “App.jsx” și parse-uite în funcție de componente
* pe site e afișată fiecare postare care se stochează într-un array local
* când este efectuată o operație CRUD (Create, Read, Update, Delete) se face un API call către baza de date, dar concomitent se modifică și array-ul local, astfel nefiind necesar un API adițional cu scopul de a updata datele afișate pe site
* când se accesează pagina din nou, datele se actualizează cu modificările făcute în baza de date, astfel, nu se va face o diferență



**Design-ul siteului** este fluid și intuitiv, fiind optimizat și scalabil pentru diferite tipuri de ecrane (telefon, tabletă, laptop, PC, etc). Pentru styling am optat pentru scss, care, o data transpuns în css, asigură compatibilitatea între browsere. Am folosit o paletă de câteva culori (#2f2f2f, #26b33e, #6ef188, #146622, #FFF00, #dc3545) pentru a crea un UI și UX plăcut, printr-o temă dark. Animațiile sunt realizate cu libraria “aos” creând diferite modele prin adaugarea atributelor data-aos-offset și data-aos-delay.



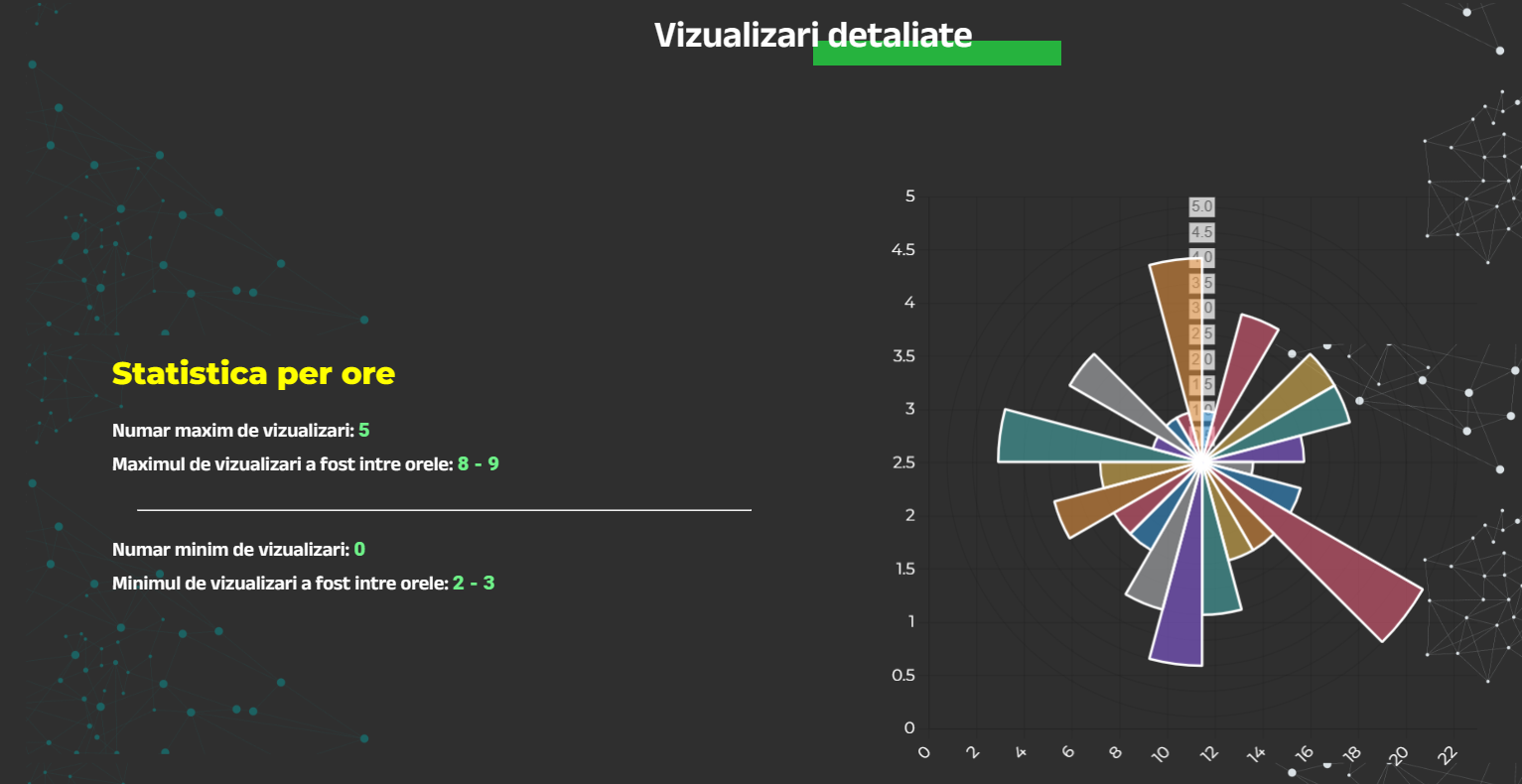


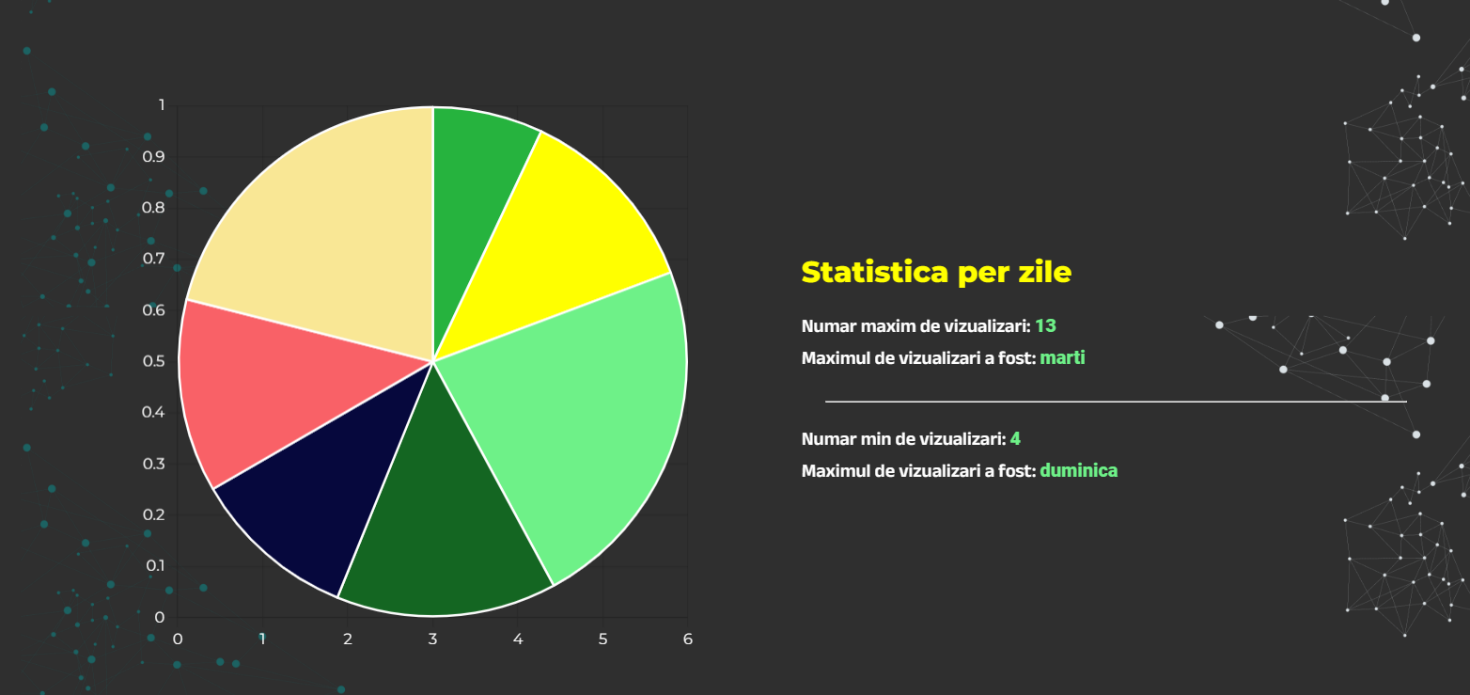
Site-ul este structurat în foldere destinate fiecărei pagini ce conține un fișier .jsx main al componentei, un fișier de styling și un folder cu subcomponentele necesare. Stylingul site-ului este dinamic: pentru fiecare componentă se importă în fișierul .scss, “\_base.scss” care conține styling-ul general valabil (h1…h6, span, p, a, etc), dar și fișiere adiacente (navbar, footer, butoanele principale, etc).

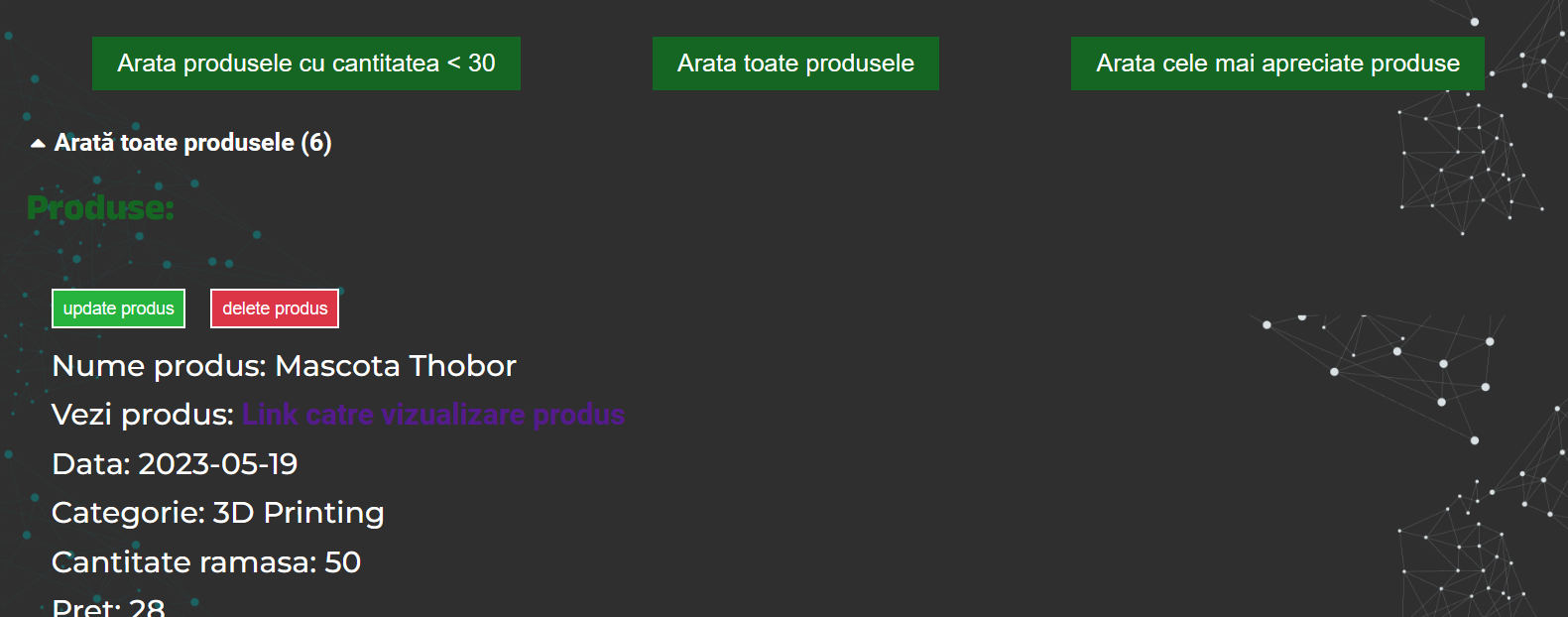
**Accesul pe pagina** **de admin** se face prin următorul proces:

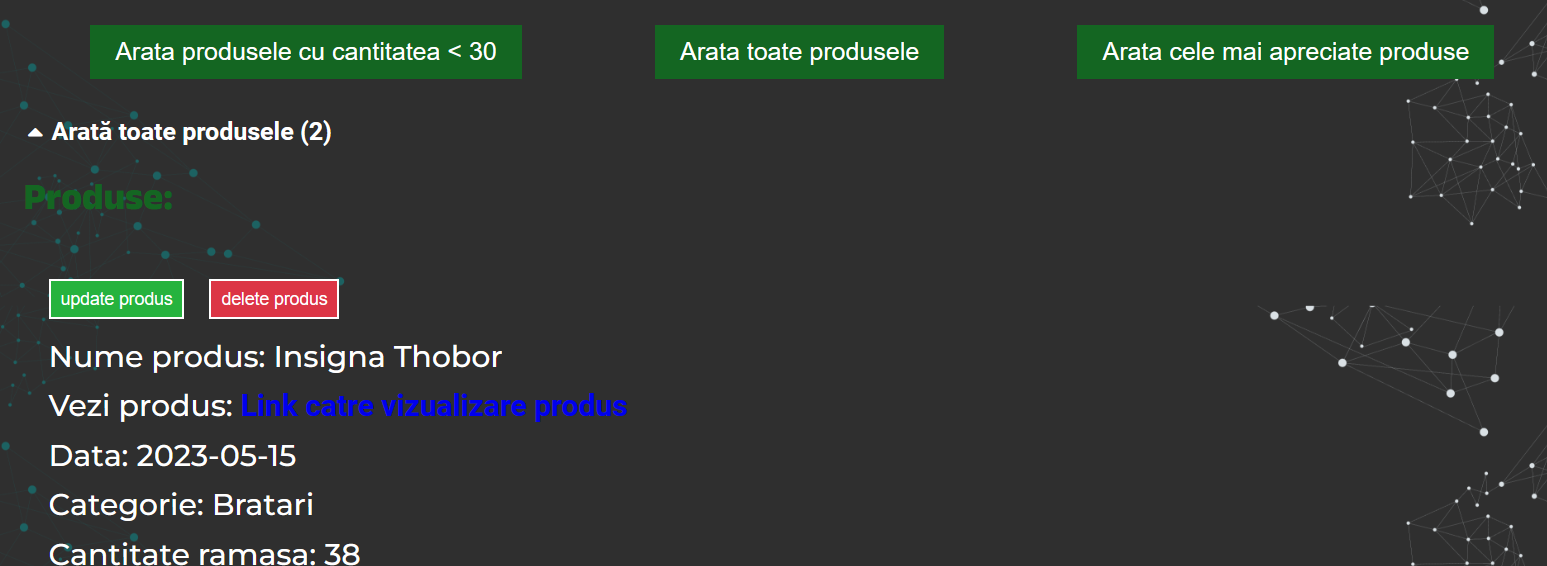
* un user cu rolul de admin, mentor, alumni sau lider de departament adaugă emailul membrilor și departamentul din care fac parte (programare, mecanică, proiectare, marketing sau caiet tehnic)
* userii se loghează cu contul de google, iar daca emailul lor regasește în baza de date, o sa primească accesul pe pagina de admin, unde, în funcție de rolul lor, o să aibă interdicții la diferite pagini

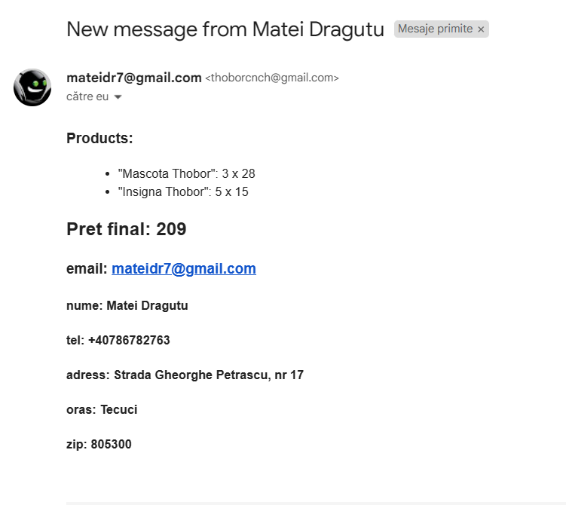
Pagina de admin conține o secțiune pentru fiecare pagină a site-ului în care se controlează conținutul cu ajutorul operațiilor CRUD.

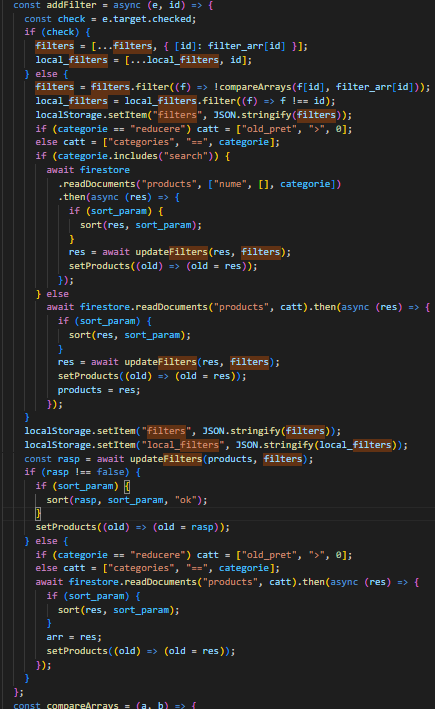
**Postările de pe blog** ne ajută să știm orele și zilele optime pentru a posta, astfel, am creat pentru fiecare postare o pagină specială pentru a afișa statisticile postării. Am folosit libraria “react-chartjs-2” pentru a crea graficele vizuale, iar pentru o interpretare mai ușoară a lor, am folosit algoritmul de vector de frecvență ce contorizează vizualizările în funcție de zile, ore și luni. Numărul de vizualizări este contorizat în funcție de ip-ul userului care vizualizează postarea și este aflat prin realizarea unui API la “https://api.ipify.org /?format=json” folosind libraria “axios”. Daca ip este deja în documentul din firebase al postării, nu se mai ia în calcul, postarea fiind considerată deja vizualizată.

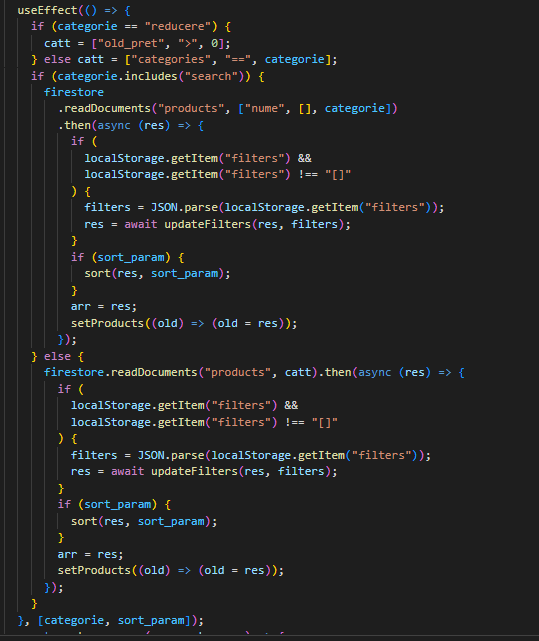


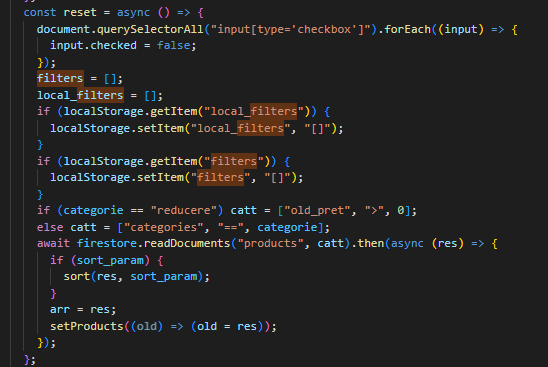
**Shopul** este controlat de pe pagina de admin unde sunt vizualizate toate produsele și pot fi adăugate, updatate și șterse. Pentru un control mai usor, au fost adăugate 2 filtre speciale: “arată produsele cu cantitatea < 30” – ca să știm ce produse trebuiesc updatate – și “arata cele mai apreciate produse” – ne indică ce produse sunt cele favorite în functie de ratingul lor oferit de cumparători -.



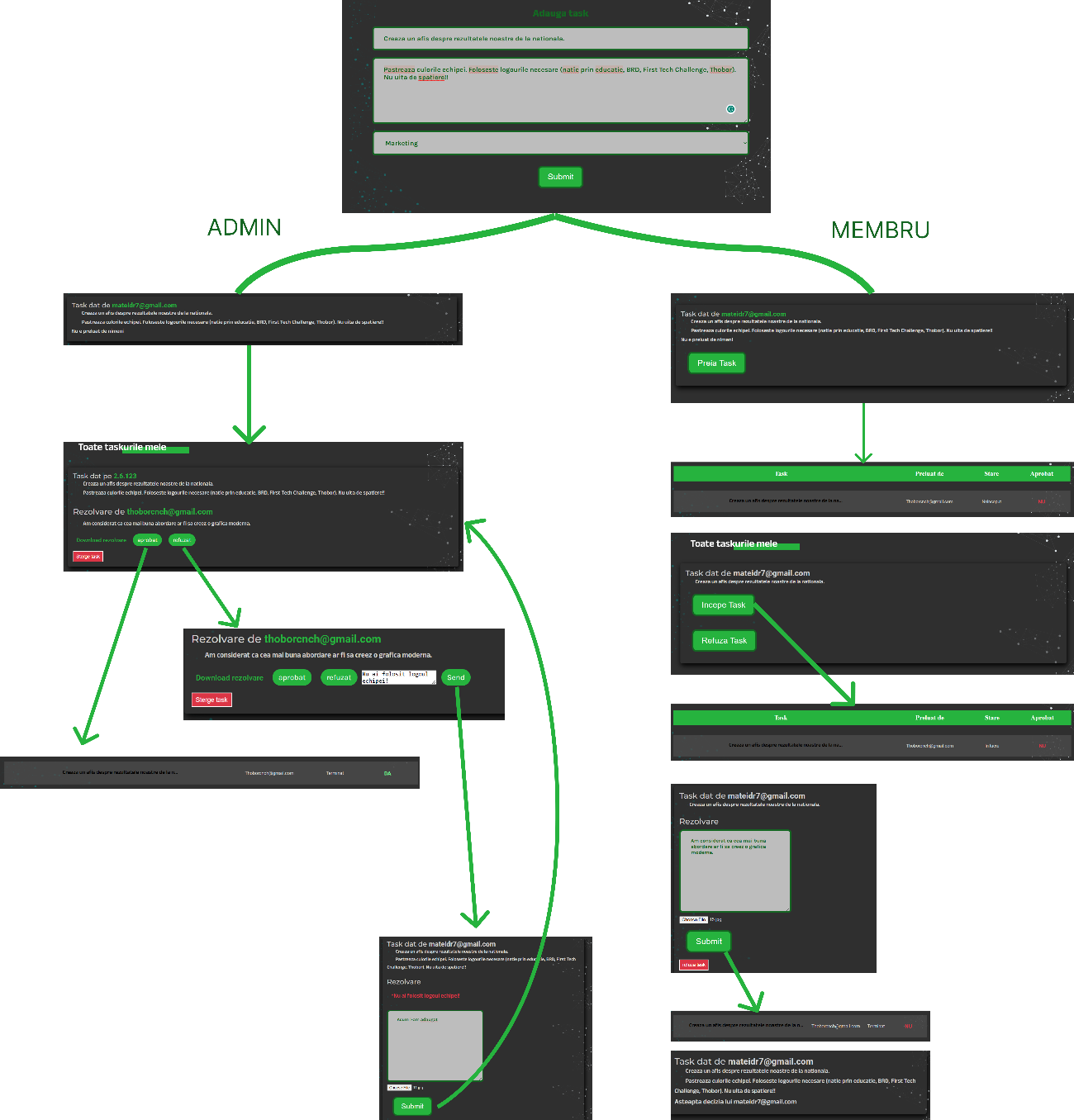
Dupa ce user-ul a ales ce materiale și câte vrea să cumpere, o să completeze detaliile de livrare și o să-si plaseze comanda. User-ul o să primească un mail de confirmare al comenzii, iar echipa Thobor o să primească un mail cu toate detaliile: fiecare produs cu cantitatea dorită și informațiile scrise de comparător. Trimiterea emailurilor se face folosind libraria “emailjs”.

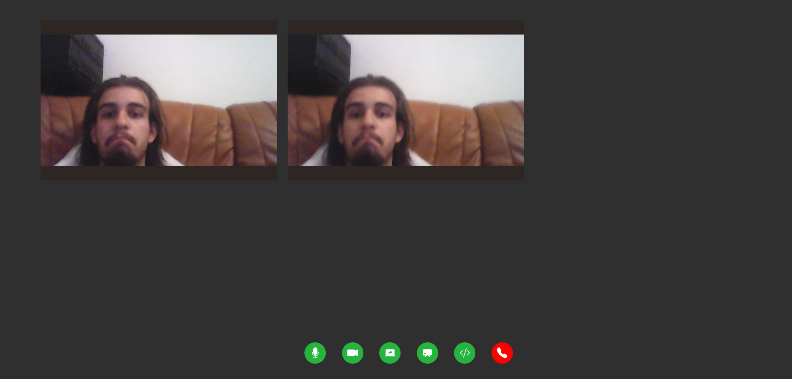
**** Filtrarea produselor se face printr-un array filtrat local, pentru a nu încărca pagina cu requesturi multiple. Filtrele se salvează în localstorage astfel încât, în cazul în care se dă refresh paginii, filtrele rămân active.

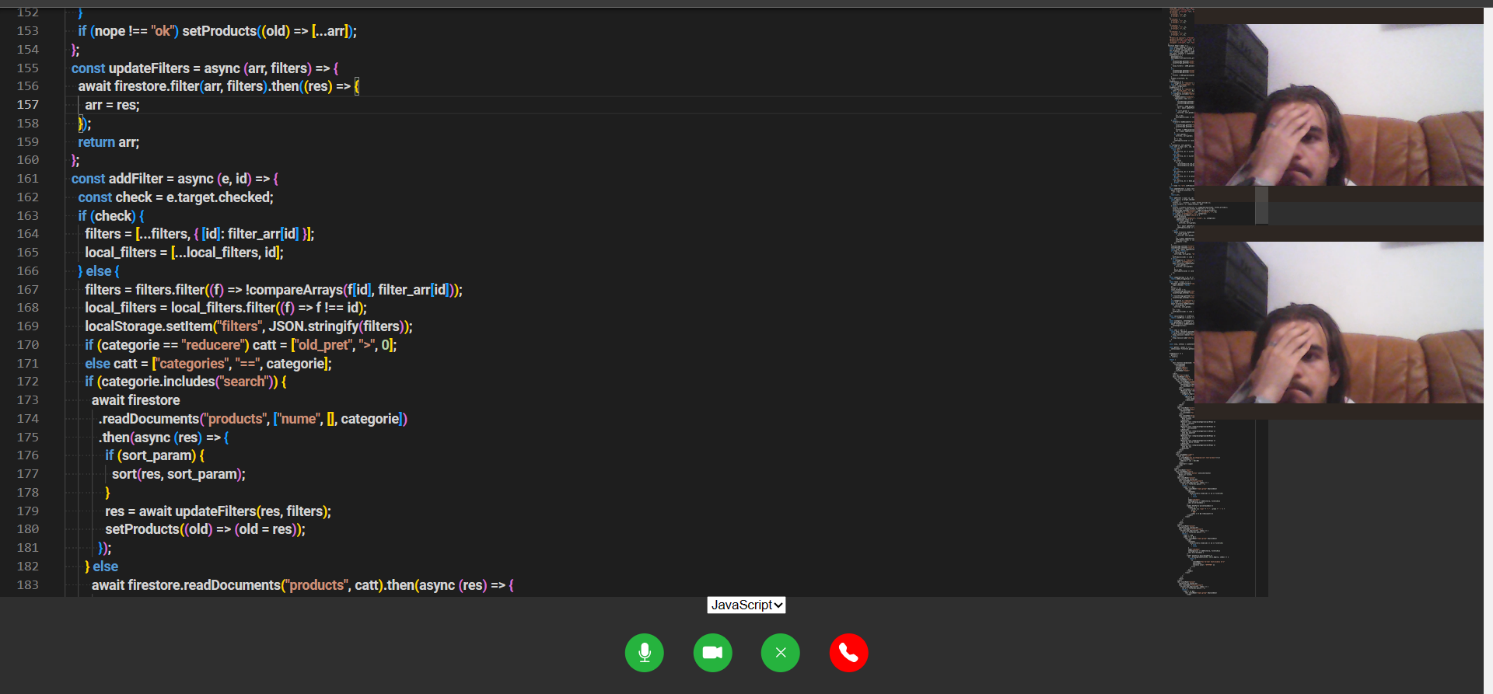


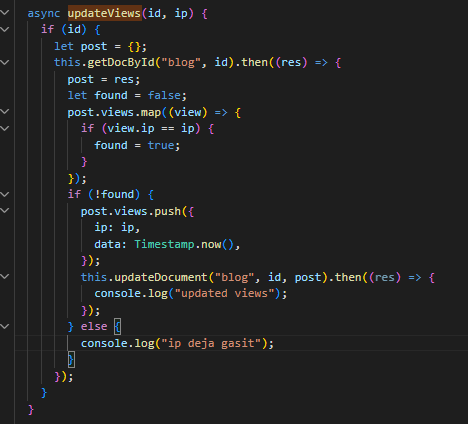
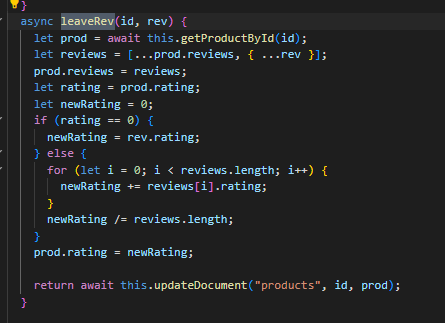
****

**În secțiunea “Tasks”** este implementat un sistem de CRM. Userii cu rolul de admin, mentor, alumni și lider de departament postează taskuri destinate membrilor echipei și pot vizualiza toate taskurile date, indiferent de departamentul selectat. Membrii o să vadă doar task-urile date pe departamentele lor. Rezolvarea poate fi aprobată sau refuzată de cel ce a creat task-ul oferind un feedback, iar fiecare modificare se updatează în tabelul cu task-urile preluate. Inițial, când un task este preluat, starea acestuia este “neinceput” și când este apăsat butonul “începe task” starea se updatează în “în lucru”, concomitent vor apărea 2 câmpuri în care membrul explică rezolvarea și poate urca și un fișier dacă este cazul. Când trimite o rezolvare, starea task-ului devine “terminat”, iar userul ce a postat task-ul îl aprobă, caz în care membrului ii va apărea mesajul “task aprobat de ….” și în tabel va aparea “DA” în coloana “Aprobat”, iar în caz contrar, membrul va primi observația scrisă de user, starea task-ului va redeveni “în lucru”, iar acesta o să trebuiasca să retrimită o rezolvare.

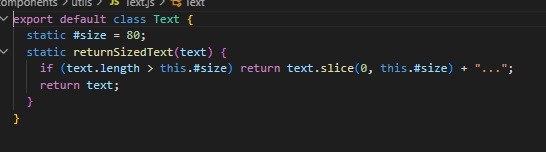


**Secțiunea “Ședințe”** este disponibilă doar userilor cu rolul de admin, mentor, alumni și lider de departament în care pot să creeze ședințe pentru întreaga echipa. Restul membrilor pot să intre la ședința dacă primesc un link ce conține id-ul acesteia. Orice persoană din cadrul unei ședințe poate să facă share screen, să deschidă un editor special în care codează toată lumea alegând unul din cele 4 limbaje de programare (javascript, java, python și C++) și să deschidă o tablă pe care să facă scheme împreună cu ceilalți. Crearea id-ului unic pentru sedința se face cu libraria “uuid”. Conectarea între membrii se face printr-un server Node.js folosind libraria “socket.io-client” și “socket.io”, iar pentru fiecare acțiune se emite un semnal între frontend și backend, astfel, manipulându-se DOM-ul la fiecare acțiune. Editorul de cod este oferit de libraria “monaco-editor/react”.



**Pentru conexiunea cu baza de date firebase**, am folosit conceptul de OOP creând o clasă denumită “Firestore”. În constructor se face conexiunea prin pasarea unor variabile din fișierul .env pentru o securitate sporită. De asemenea, se realizează și o instanță a logării folosind clasa “GoogleAuthProvider”. Clasa Firestore() conține funcții destinate operațiilor CRUD, dar și a queries ample. Spre exemplu, funcția “filter” care filtrează produsele pentru pagina “Shop” în funcție de ce filtre precizez in variabila “filterss” ce este un array de 3 elemente: [“field-ul după care se sortează”, “semnul de comparație”, “valoarea de comparație"]. Funcția “updateViews” are 2 argumente: id și ip. Verifică daca ip-ul se regasește în vizualizările din postarea cu id-ul pasat folosind falsa ipoteză. În caz că nu se gasește, vizualizarile postarii sunt modificate. Funcția “leaveRev” adaugă un nou review la postarea al carei id este pasat ca argument și concomitent face media rating-urilor dacă s-au mai găsit și alte reviews folosind media artimetică. În caz contrar, media este ratingul review-ului pasat, acesta fiind primul.

**Clasele “Placeholder()” și “Text()”** conțin funcții de tip static pentru a ușura modificarea datelor în diferite situații. Spre exemplu, clasa “Text()” are o functie “returnSizedText()” prin care se pasează un text lung și aceasta returnează primele 80 de caractere și “…” pentru a marca faptul că textul inițial este mai mare și pentru a-l citi pe tot, userul trebuie să acceseze postarea. De asemenea, funcția “roundit()” din clasa “Placeholder()” returnează numărul pasat în argument rotunjit la un număr exact de zecimale.





**Site-ul a fost împărțit în 3 domenii principale:** thobor.ro, sumulator.netlify.app și robotsez6.netlify.app. Scopul acestei structuri este cel de a creea o experiență cât mai fluidă pentru utilizatori întrucât simulatorul și galeria robotului conțin fișiere mari ce afectează performanța site-ului principal, thobor.ro. Motiv pentru care, simulatorul și galeria sunt nu valabile pentru telefoane sau tablete, condiție stabilită prin libraria “react-device-detect”. De asemenea, pentru a asigura o performanță sporită, imaginile de pe site sunt randate folosind “<LazyLoadImage/>” din libraria “react-lazy-load-image-component”. Scorul de 100 masurat pe pagespeed.web.dev la secțiunea SEO a fost atins întrucât am folosit toate tagurile meta necesare (description, keywords, etc).

**Securitatea site-ului** este realizată în mai multe etape: accesul la pagina “admin” este interzisă userilor neadăugați în baza de date; informațiile sensibile (keys pentru firebase, pentru serviciu de email, etc) sunt scrise într-un fișier .env care nu este publicat pe github și pentru producție, în secțiunea “environment variabiles” de la netlify, hostul ales pentru frontend; baza de date are regulile setate astfel încât metodele HTTPS să fie în parametrii optimi și am setat singurul domeniu de unde se pot face request-uri către baza de date, thobor.ro; codul sursă nu se poate vizualiza în “developer tools/sources” datorită faptului că am rescris comanda de build astfel:

* “set \”GENERATE\_SOURCEMAP=false\” && react-scripts build”.

**Simulatorul** reprezintă un joc ce încearcă să imite cu acuratețe modul de joc al ultimului sezon al competiției BRD First Tech Challenge.

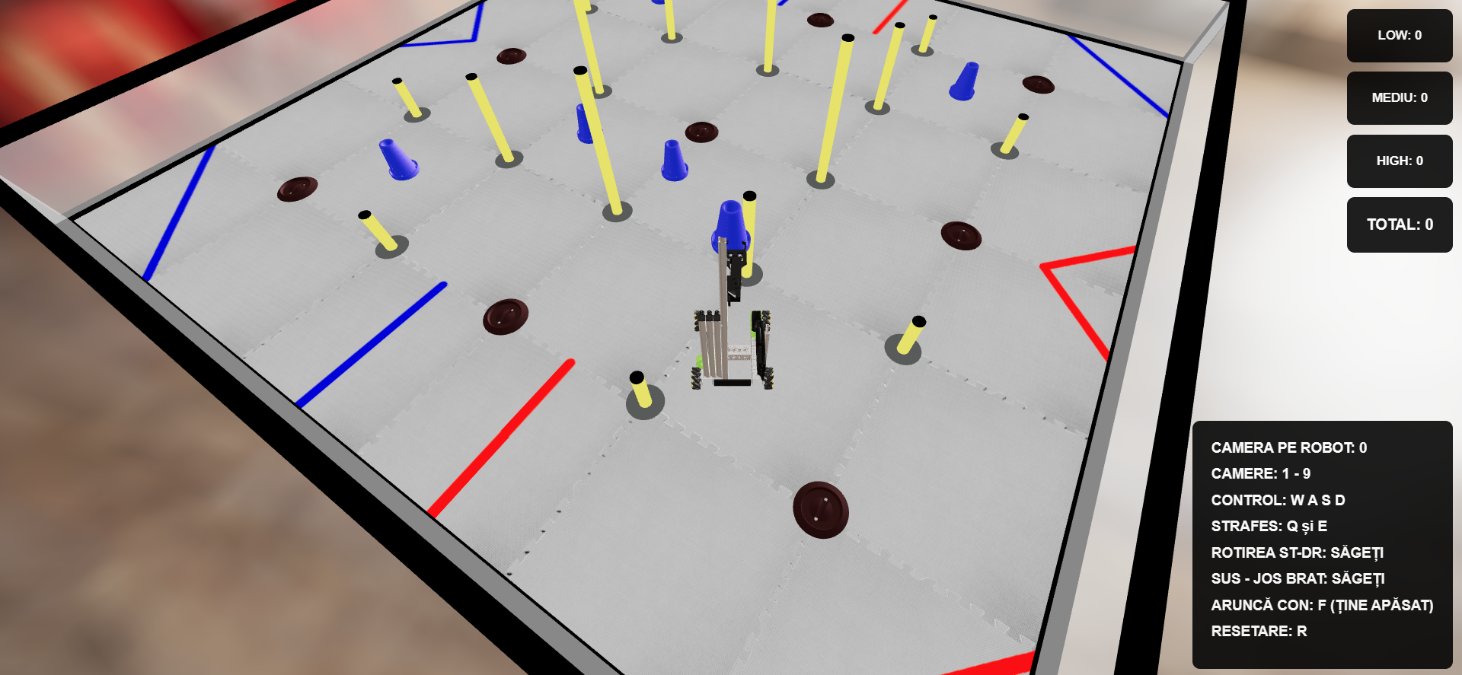
În realizarea acestuia au fost folosite librariile “three.js” și “three-fiber” pentru a calcula partea grafică 3d și a face randarea acesteia, “three-drei” pentru a adăuga hook-uri astfel încât procesul de implementare a claselor să fie simplificat, “cannon” pentru a calcula scena fizică și coliziunile dintre corpuri și socket.io-client pentru a face conexiunea la un server node ce transmite datele primite tuturor clientilor. Three.js a fost principala alegere pentru flexibilitatea de care dă dovadă în construirea scenelor și personalizarea acestora.

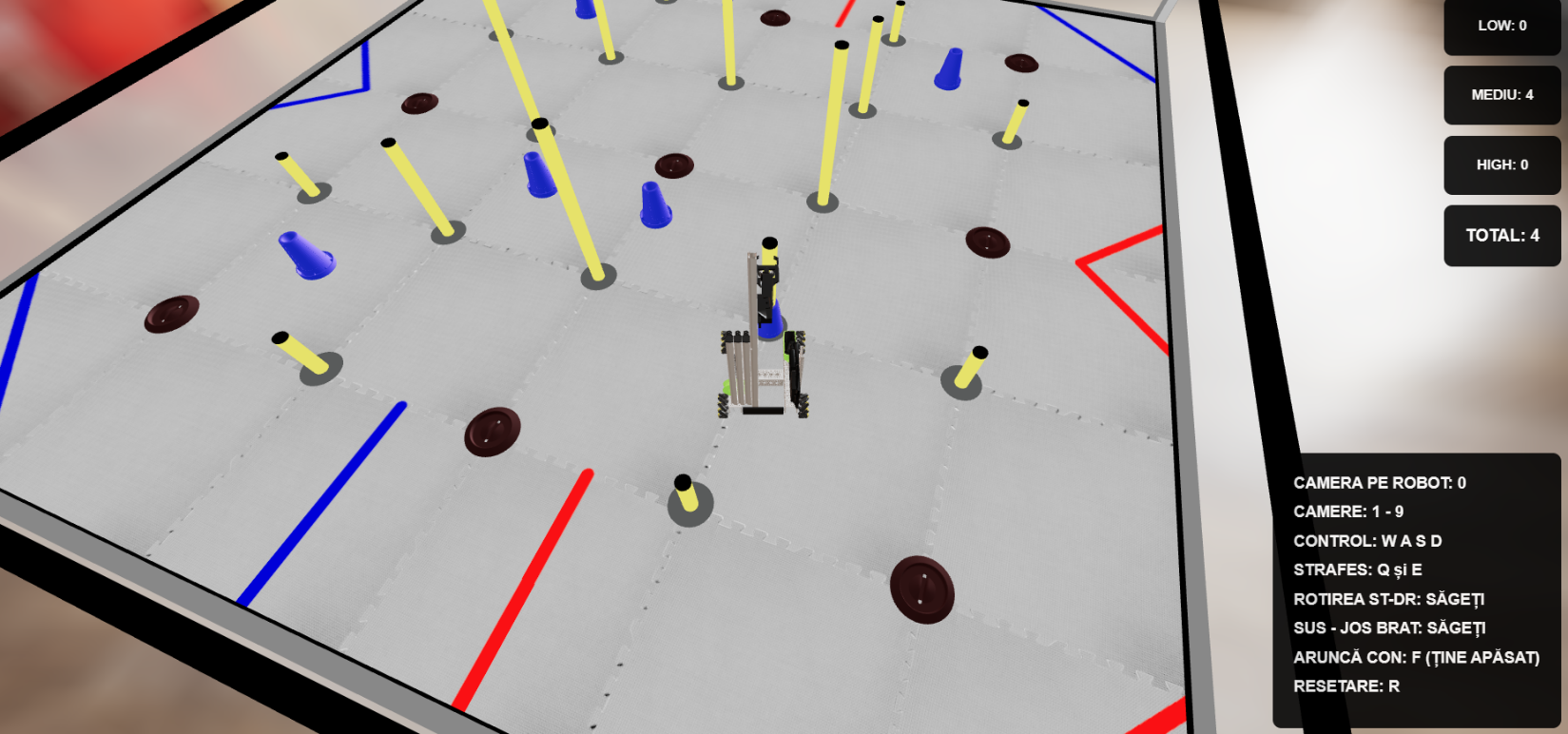
Din punct de vedere logistic, simulatorul se împarte în 3 secțiuni care acționează pentru a pune în mișcare câte o parte a simulatorului. Prima secțiune este “environment” care returnează elemente precum fundalul scenei, lumina ambientală, camera folosită pe robot și alte efecte ce sunt adăugate în scenă. A doua secțiune este “field” și conține componetele ce construiesc terenul de joc: pereți, junctions, conuri, podea, grounds. Pentru fiecare corp este importat un model 3d al obiectului realizat în Blender sau este generat pe baza dimensiunilor oferite componentei, apoi este generat în spațiul fizic un corp cu aceeași formă pentru a simula coliziunea cu celelalte corpuri. Componenta conului, pentru a putea fi manipulate cu ajutorul brațului robotului, verifică distanța de la con la braț, iar în cazul în care acesta se află la o distanță mai mică de 2 unități își va seta poziția la cea a brațului cu ajutorului API-ului corpului fizic .A treia secțiune “teleop” are în vedere funcționalitatea robotului și conține componente și hooks precum useControls, useWheels, Lokione (robotul), Brat. Principalele componente auxiliare sunt useControls care aplică forțe asupra robotului în funcție de tastele apăsate pentru a putea fi controlat și useWheels care generează 4 roti cu ajutorul funcției useCompoundBody() și a obiectului ce stochează configurația roților pentru a le atașa de robot. Componența Lokione (robotul) generează un vehicul în spațiu fizic pentru a simula coliziunile și controlul acestuia și un alt obiect fizic, brațul lui, pentru a putea controla conurile (acesta este teleportat constant în fața robotului la înălțimea specificată din taste). Pe lângă obiectele fizice, este generat și un socket prin care sunt transmise datele către server pentru a fi împărtășite cu ceilalți jucători. Socketul este pasat și la alte componete pentru a putea transmite diferite date precum mișcările robotului din apăsările tastelor. Socketul este însoțit de un webWorker ce primește prin intermediul funcției “postMessage()” date pe care le trimite la server dintr-un thread diferit (poziția robotului și a conurilor), acest lucru este necesar pentru a evita pierderile în performanță atunci când este trimis un flux mare de date ce se procesează greu.

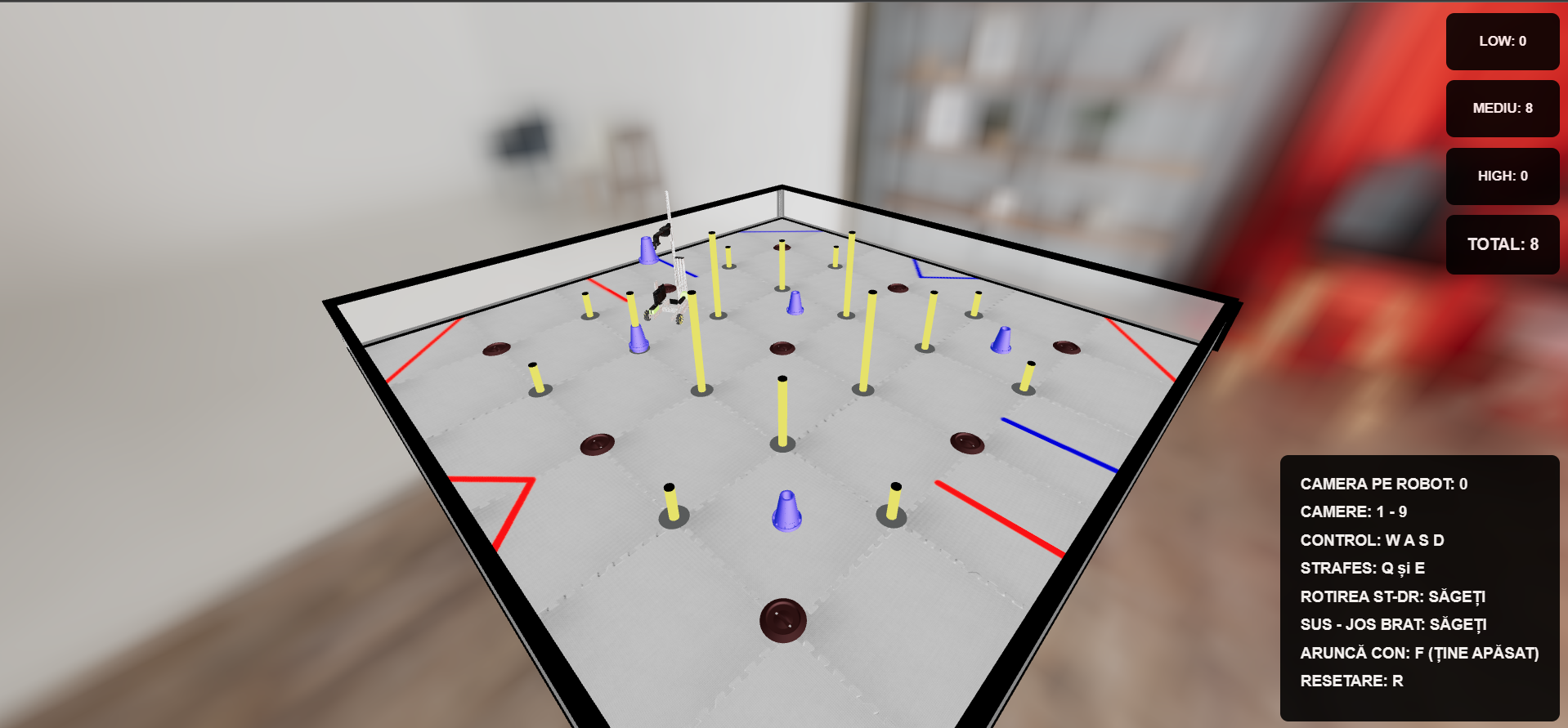
Toate datele importante ale componentelor (poziția robotului, poziția brațului, poziția conurilor, conurile marcate pentru punctaj, etc.) sunt salvate într-o clasă separată “ExternalData”, din care fiecare componentă poate retrage aceste informații sau să salveze informații noi.

Pentru a distinge între mai mulți jucători în server este utilizat un cod unic generat pentru fiecare browser și salvat în localStorage cu cheia “horia\_id”. Codul reprezintă momentul în care jucătorul intra pe pagina “/multi” a simulatorului. Timpul este reprezentat drept epoch time la momentul generării. Pe langă codul unic sunt utilizate și două valori salvate în sessionStorage ce determină dacă jocul se desfășoara în modul multiplayer și dacă jucătorul este gazda camerei.

Serverul de webSocket al simulatorului rulează prin intermediul Node.js și salvează datele primite(cod unic al jucătorilor, poziții ale robotului) și le trimite la toți clienții.

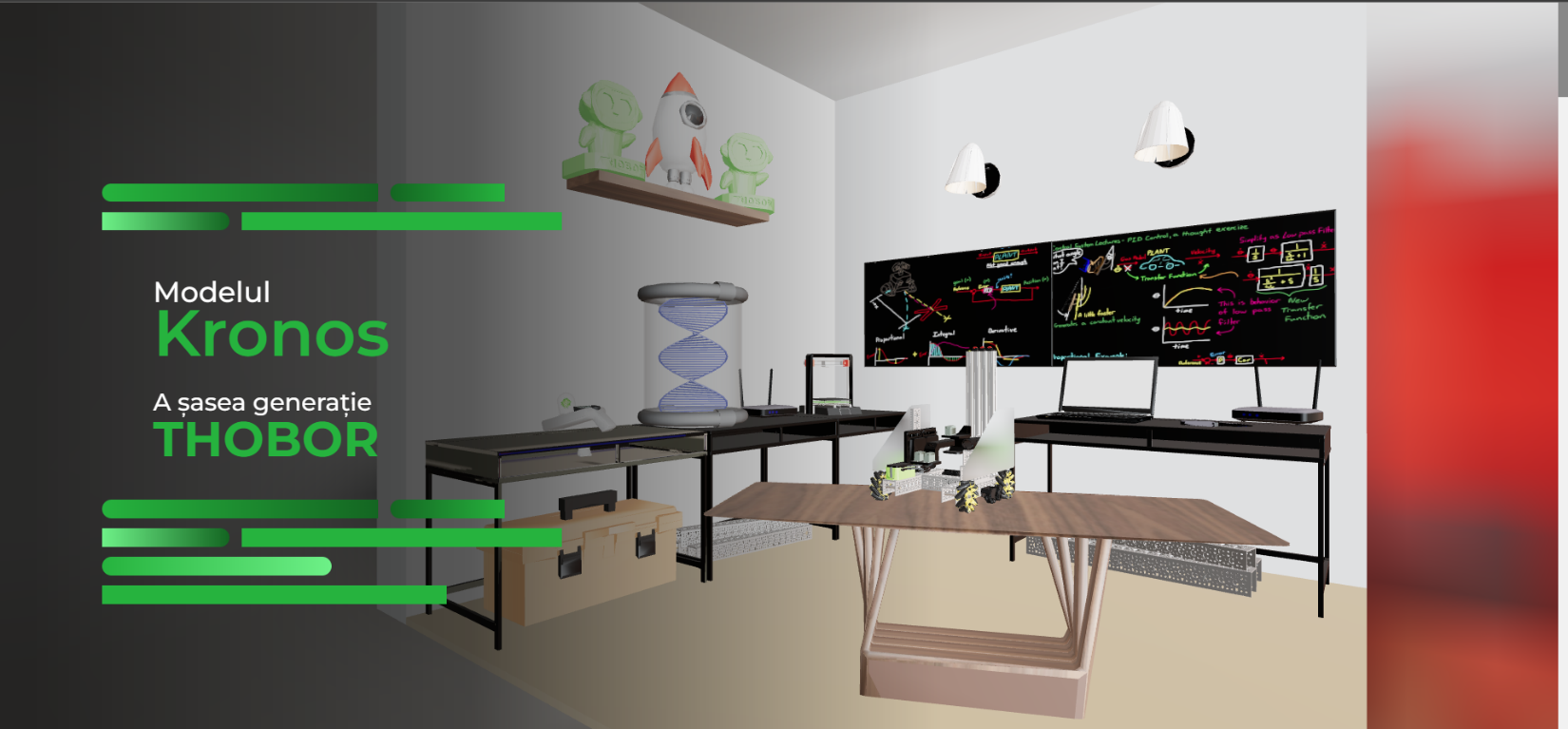
Toate secțiunile simulatorului sunt conectate în fișierul “Assembly” pentru a simplifica procesul de integrare a simulatorului într-o pagină.

****

****

**Pagina destinată robotului** din acest sezon oferă posibilitatea de a-i vizualiza sistemele importante, iar scrollând, camera se aproprie de fiecare, apărând o descriere individuală. Trecerea smooth a camerei se face prin funcția “lerp” în care precizez coordonatele camerei depinzând de scroll-ul pagininii.

Pagina se împarte în două componente. Prima componentă conține textul cu informațiile despre robot, împărțit pe clase și stilizat. A doua componentă asigură scena 3d cu robotul și se împarte în trei parți: prima parte importă modelul 3d al robotului realizat în Blender, a doua parte adaugă camera în care robotul este așezat, iar a treia parte oferă lumina ambintală, fundalul scenei și asigură mișcările camerei în funcție de nivelul de scroll la care se află pagina.



****