# Capitolul 5. Testarea aplicației

Testarea reprezintă o componentă fundamentală în procesul de dezvoltare software, având rolul de a asigura calitatea, stabilitatea și funcționarea corectă a aplicației. Indiferent de cât de bine este proiectat sau dezvoltat un sistem, testarea este esențială pentru a detecta erorile, pentru a valida comportamentul aplicației în diverse scenarii și pentru a garanta o experiență coerentă și sigură pentru utilizatorii finali.

În general, testarea se împarte în două mari categorii:

* **Testarea manuală**, în care un tester uman verifică funcționalitățile aplicației, urmând scenarii prestabilite sau explorând liber comportamentul sistemului;
* **Testarea automată**, în care teste predefinite sunt executate automat prin intermediul unor framework-uri specializate (ex: JUnit, PyTest, Jasmine, PHPUnit), permițând validarea repetitivă, rapidă și fiabilă a componentelor aplicației.

Ambele tipuri de testare sunt complementare. Testarea manuală oferă flexibilitate și detectează mai ușor problemele de interfață sau de logică neprevăzută, în timp ce testarea automată asigură o verificare rapidă și constantă a funcționalităților critice, fiind ideală pentru procesele de integrare continuă și livrare rapidă.

În cadrul acestei lucrări, testarea a vizat atât partea de frontend (interfața utilizatorului), cât și backend-ul (API-ul și logica de generare a orarului), pentru a asigura buna funcționare a aplicației de la introducerea datelor până la exportul orarului final.

**5.1 Testarea manuală**

Testarea manuală a fost utilizată pentru a verifica dacă aplicația funcționează corect din perspectiva utilizatorului final. Aceasta presupune parcurgerea interfeței, introducerea datelor și observarea reacțiilor aplicației în diverse scenarii. Au fost testate componentele critice precum autentificarea, adăugarea datelor, generarea orarului, exportul și reutilizarea orarelor salvate.

**✅ Testul 1 – Autentificare**

**Scenariul 1 – Date corecte**

* Pasul 1: Se deschide pagina de autentificare.
* Pasul 2: Se introduce admin / parola123.
* Pasul 3: Se apasă pe „Autentificare”.
* ✅ Rezultat: Se accesează pagina principală (dashboard).

**Scenariul 2 – Parolă greșită**

* Pasul 1: Se introduce admin / gresit.
* ✅ Rezultat: Se afișează mesajul: „Date de autentificare incorecte”.

**Scenariul 3 – Câmpuri goale**

* Pasul 1: Se lasă necompletat user/parolă.
* ✅ Rezultat: Se afișează mesajul: „Completați toate câmpurile”.

**Scenariul 4 – Acces direct la pagină securizată**

* Pasul 1: Se încearcă accesarea /dashboard fără autentificare.
* ✅ Rezultat: Redirecționare către login.

**✅ Testul 2 – Gestionarea profesorilor**

**Scenariul 1 – Adăugare validă**

* Pasul 1: Se completează toate câmpurile: nume, discipline, tipuri, nivel.
* ✅ Rezultat: Profesorul apare în listă, cu mesaj de confirmare.

**Scenariul 2 – Lipsă nume**

* Pasul 1: Se lasă câmpul „Nume” gol.
* ✅ Rezultat: Mesaj de eroare vizibil: „Numele este obligatoriu”.

**Scenariul 3 – Mai multe discipline**

* Pasul 1: Se adaugă „Programare, Baze de date”.
* ✅ Rezultat: Disciplinele apar separate și corect afișate.

**✅ Testul 3 – Gestionarea sălilor**

**Scenariul 1 – Adăugare sală**

* Pasul 1: Se adaugă „GC01”, tip „Curs”.
* ✅ Rezultat: Sala apare în lista corespunzătoare.

**Scenariul 2 – Cod duplicat**

* Pasul 1: Se reintroduce „GC01”.
* ✅ Rezultat: Mesaj: „Sala există deja”.

**Scenariul 3 – Resetare săli**

* Pasul 1: Se șterg toate sălile și se introduc din nou.
* ✅ Rezultat: Lista este actualizată.

**✅ Testul 4 – Generarea orarului**

**Scenariul 1 – Generare completă**

* Pasul 1: Se introduc profesori, săli, grupe, reguli.
* Pasul 2: Se accesează pagina de generare.
* Pasul 3: Se apasă „Generează orar cu AI”.
* ✅ Rezultat: Orarul este generat pentru toate grupele și zilele.

**Scenariul 2 – Lipsă săli**

* Pasul 1: Nu se introduc săli.
* ✅ Rezultat: Butonul de generare nu este disponibil.

**Scenariul 3 – Lipsă regulă selectată**

* Pasul 1: Se accesează pagina fără selectarea unei reguli.
* ✅ Rezultat: Apare Swal cu mesajul: „Regulă neselectată”.

**✅ Testul 5 – Exportul orarului**

**Scenariul 1 – Export PDF**

* Pasul 1: Se generează un orar complet.
* Pasul 2: Se apasă pe „Exportă PDF”.
* ✅ Rezultat: Se descarcă fișierul orar.pdf.

**Scenariul 2 – Export Excel**

* Pasul 1: Se apasă „Exportă Excel”.
* ✅ Rezultat: Se descarcă orar.xlsx.

**Scenariul 3 – Fără orar generat**

* Pasul 1: Se accesează pagina fără a genera orar.
* ✅ Rezultat: Butonul de export nu apare deloc.

**Scenariul 4 – Orar incomplet/invalid**

* Pasul 1: Se generează orar cu date lipsă.
* ✅ Rezultat: Exportul nu este disponibil.

**✅ Testul 6 – Salvarea și încărcarea orarelor anterioare**

**Scenariul 1 – Salvare implicită**

* Pasul 1: Se generează un orar.
* ✅ Rezultat: Orarul apare în lista „Orare salvate anterior”.

**Scenariul 2 – Încărcare orar existent**

* Pasul 1: Se apasă „Încarcă” pe un orar din listă.
* ✅ Rezultat: Orarul se afișează în aplicație.

**Scenariul 3 – Editare orar**

* Pasul 1: Se apasă „Editează” pe un orar salvat.
* ✅ Rezultat: Datele pot fi modificate și salvate din nou.

**Scenariul 4 – Ștergere orar**

* Pasul 1: Se apasă „Șterge” și se confirmă.
* ✅ Rezultat: Orarul este eliminat din listă.

**Scenariul 5 – Căutare orar**

* Pasul 1: Se caută după nume.
* ✅ Rezultat: Lista este filtrată instant.

## 5.2. Testarea automată

Testarea automată reprezintă un proces esențial în dezvoltarea aplicațiilor software moderne, asigurând validarea funcționalităților sistemului fără intervenție umană. În cadrul acestui proiect, testarea automată a fost utilizată pentru a verifica corectitudinea componentelor backend și frontend ale aplicației web pentru generarea orarelor universitare, contribuind astfel la creșterea fiabilității și mentenanței codului.

Scopul principal al testării automate este de a detecta rapid erorile și regresiile, facilitând dezvoltarea iterativă și introducerea de noi funcționalități fără riscul deteriorării celor existente. În plus, testele automate oferă un grad ridicat de încredere în comportamentul aplicației în scenarii diverse.

Pentru acest proiect, testele automate au fost structurate în două mari categorii:

* **Testarea backendului**: realizată cu ajutorul bibliotecii pytest în Python, a vizat testarea rutelor API, interacțiunii cu baza de date, integritatea datelor și răspunsurile corecte la solicitările HTTP.
* **Testarea frontendului**: realizată cu ajutorul bibliotecii Jest și a utilitarului React Testing Library, a urmărit verificarea componentelor vizuale, interacțiunilor utilizatorului, validarea formularelor și afișarea corectă a datelor în interfață.

Testarea automată a permis dezvoltarea controlată a aplicației și a contribuit la menținerea calității codului pe parcursul întregului ciclu de dezvoltare. În continuare, vor fi prezentate testele relevante implementate pentru componentele backend și frontend, împreună cu explicațiile aferente.

### 5.2.1 Testarea backend

Testarea backendului a fost realizată cu ajutorul frameworkului pytest, o bibliotecă Python eficientă și extensibilă, folosită pentru a valida corectitudinea logicii aplicației. Testele au fost organizate pe fișiere separate, în funcție de funcționalitățile testate, fiecare acoperind o zonă esențială a aplicației:

* rutare și API (endpoints)
* interacțiuni cu baza de date
* reguli de generare și validare a orarului

### 5.2.1 Testarea funcționalității de generare orar – test\_api\_generator.py

Fișierul test\_api\_generator.py conține testele pentru rutele care gestionează procesul de generare automată a orarului în aplicație. Aceste rute fac parte din blueprint-ul generator\_bp, responsabil cu afișarea formularului de generare și cu logica de procesare a datelor introduse de utilizator.

Pentru testare a fost utilizat modulul pytest, împreună cu mecanismul test\_client oferit de Flask, care permite simularea cererilor HTTP în mod controlat, fără a porni serverul real.

Primul test din fișier verifică dacă pagina /genereaza\_orar\_propriu este afișată corect atunci când este accesată printr-o cerere GET. Se asigură astfel că interfața utilizatorului este încărcată cu succes și conține formularul necesar generării orarului. În plus, se verifică prezența titlului corespunzător („Orar Generat Automat”) în conținutul HTML returnat.

Al doilea test simulează o cerere POST validă către ruta /genereaza\_algoritm\_propriu, trimițând un set complet de date: nivelul de studii, anul și grupele selectate. Se verifică dacă serverul răspunde cu succes (status 200) și dacă răspunsul conține un obiect orar sub forma unui dicționar. Acesta este considerat **testul esențial** pentru verificarea funcționalității principale a aplicației — generarea propriu-zisă a orarului.

Ultimul test introduce o situație de tip edge-case: se trimite o cerere POST cu un payload incomplet (fără nivel, an sau grupe). Testul validează că aplicația detectează această situație și returnează un răspuns cu cod de eroare 400 Bad Request, împreună cu un mesaj de eroare.

Prin aceste teste, se validează atât **comportamentul pozitiv (happy path)**, cât și gestionarea erorilor pentru cazuri invalide, asigurând astfel o robustețe minimă necesară pentru procesul de generare a orarului.

📌 Toate testele din test\_api\_generator.py au fost executate cu succes, confirmând funcționarea corectă a rutelor de generare a orarului în condiții variate.

**5.2.2 Testarea gestionării profesorilor – test\_api\_profesori.py**

Fișierul test\_api\_profesori.py conține testele automate dedicate operațiilor de gestionare a profesorilor în aplicație: adăugare, listare, actualizare și ștergere. Aceste funcționalități sunt esențiale pentru corecta funcționare a aplicației, întrucât fiecare profesor trebuie să aibă asociate discipline, niveluri și intervale orare de disponibilitate.

Testele au fost implementate folosind biblioteca pytest, împreună cu clientul de testare al Flask, care permite simularea de cereri HTTP către rutele definite, fără a porni un server real. Pentru a se evita poluarea bazei de date, testele includ și pași de curățare (cleanup), precum ștergerea profesorilor creați temporar.

Testul principal din acest fișier validează fluxul complet de **adăugare și ștergere a unui profesor**. Se trimite o cerere POST către ruta /adauga\_profesor, cu un payload ce include numele profesorului, disponibilitatea sa și disciplina predată. După confirmarea adăugării cu succes, se verifică existența profesorului în lista returnată de /toti\_profesorii. În final, profesorul este șters printr-o cerere DELETE, testând astfel funcționalitatea completă a ciclului de viață al unui profesor.

Un alt test important simulează **actualizarea unui profesor existent**. Pentru aceasta, un profesor este inserat direct în baza de date, după care este modificat printr-o cerere PUT care actualizează datele sale (nume, orar de disponibilitate și discipline). Se verifică dacă modificările sunt reflectate corect, iar la final, profesorul este șters pentru a nu lăsa date reziduale.

Prin aceste teste, este validată funcționarea corectă a tuturor operațiilor CRUD pentru profesori, atât în cazul unor date valide, cât și în scenarii reale ce presupun modificarea structurii asociate profesorilor (ex: schimbarea disciplinei, a nivelului sau a tipului de activitate).

📌 **Toate testele din test\_api\_profesori.py au rulat cu succes**, confirmând că operațiile esențiale pentru gestionarea profesorilor funcționează corespunzător și că datele din backend sunt tratate corect și coerent.

📸 *(Se recomandă atașarea unei capturi de ecran cu testele bifate verde în editorul de cod.)*

**5.2.3 Testarea interacțiunii directe cu baza de date – test\_database.py**

Fișierul test\_database.py conține o serie de teste care verifică funcționarea corectă a conexiunii la baza de date și integritatea operațiilor SQL de bază, utilizând clasa personalizată Database. Aceste teste nu implică API-ul Flask, ci interacționează direct cu baza de date MySQL, simulând scenarii reale de acces și modificare a datelor.

Primul test (test\_show\_tables) verifică dacă metoda care listează tabelele din baza de date funcționează corect și dacă există tabela esențială profesori. Acest lucru asigură că structura bazei de date este completă și că aplicația poate accesa datele necesare.

Al doilea test (test\_select\_profesori) execută o interogare simplă de tip SELECT pe tabela profesori și verifică dacă răspunsul este o listă validă. Dacă există date în tabelă, se asigură că structura fiecărui rezultat include coloanele esențiale, precum id.

Cel mai complex test din fișier (test\_insert\_and\_delete\_profesor) simulează un ciclu complet de inserare și ștergere a unui profesor de test:

* Se inserează manual un profesor în baza de date cu nume și un orar fictiv.
* Se verifică dacă profesorul a fost adăugat cu succes printr-o interogare filtrată.
* Apoi, profesorul este șters.
* În final, se confirmă că datele au fost eliminate complet și nu mai există în tabelă.

Aceste teste sunt utile pentru a valida funcționalitatea de bază a conexiunii și manipulării datelor și pentru a garanta că aplicația poate lucra cu baza de date în condiții normale, fără erori sau pierderi de informații.

📌 **Toate testele din test\_database.py au fost executate cu succes**, demonstrând că structura bazei de date este corectă și că operațiile SQL fundamentale (inserare, interogare, ștergere) funcționează conform așteptărilor.

📸 *(Este recomandată atașarea unei capturi de ecran cu testele bifate în consola de rulare pentru a demonstra succesul execuției.)*

**5.2.4 Testarea generatorului propriu de orar – test\_orar\_generator.py**

Fișierul test\_orar\_generator.py conține testele pentru clasa OrarGenerator, care implementează logica personalizată de generare a orarului săptămânal. Aceste teste validează inițializarea corectă a generatorului, aplicarea criteriilor personalizate, precum și respectarea constrângerilor impuse asupra structurii orarului.

Pentru a izola testele de o conexiune reală la baza de date, clasa OrarGenerator a fost testată folosind tehnici de simulare (mocking), cu ajutorul bibliotecii unittest.mock. Astfel, toate dependențele de conexiuni SQL au fost înlocuite cu obiecte simulate, permițând testarea logicii interne într-un mediu controlat.

Un prim test important a verificat **inițializarea corectă a generatorului**, inclusiv setările implicite privind zilele, intervalele orare și criteriile prestabilite (de exemplu, pauza de miercuri între 14:00–16:00 și maximul de 8 ore pe zi).

Un alt test a evaluat metoda de **extragere a nivelului și anului** pe baza denumirii grupei (ex: „LM2a” trebuie să returneze „Master”, anul II). Acesta este un pas critic pentru alocarea corectă a intervalelor orare.

Au fost testate și funcționalitățile de **actualizare a criteriilor** dinamic, pornind de la datele transmise din interfață. Testul a verificat dacă modificările asupra orelor de master, pauzelor forțate sau numărului maxim de ore se reflectă corect în obiectul generatorului.

Pentru testarea propriu-zisă a metodei genereaza\_orar, au fost simulate cazuri în care grupelor li se alocă activități fictive, respectând regulile impuse. De exemplu, s-a testat dacă în ziua de miercuri la ora setată ca pauză, orarul generează automat o activitate de tip „Pauză forțată”.

De asemenea, un test specific a evaluat dacă se respectă **limitarea la maximum 9 cursuri diferite** într-un orar complet. S-au simulat apeluri repetate către funcția de generare, fiecare cu denumiri de activități unice, iar testul a verificat că numărul total de cursuri nu depășește limita impusă.

Prin aceste teste, s-a validat că generatorul personalizat respectă cerințele funcționale, poate fi configurat flexibil și răspunde corect la reguli impuse de utilizator.

📌 **Toate testele din test\_orar\_generator.py au fost executate cu succes**, confirmând funcționalitatea logicii personalizate de generare a orarului și comportamentul previzibil în scenarii complexe.

📸 *(Se recomandă atașarea unei capturi de ecran cu execuția testelor bifate verde în terminal sau editor.)*

**5.2.5 Testarea validării orarului generat – test\_validator\_orar.py**

Fișierul test\_validator\_orar.py include testele automate pentru componenta de validare a orarului generat, implementată în clasa ValidatorOrar. Scopul acestei componente este de a verifica dacă orarul respectă reguli structurale importante, în special sincronizarea activităților comune între grupele unui an.

Validarea se concentrează în special asupra **cursurilor comune**, care trebuie să apară în același interval orar, în aceeași zi și în aceeași sală pentru toate grupele implicate. Aceasta este o cerință academică fundamentală, întrucât cursurile sunt predate simultan tuturor studenților dintr-un an și nu pot fi distribuite aleatoriu în orar.

Un prim test acoperă cazul **valid**: două grupe din același an (de exemplu, LI1a și LI1b) au cursul de „BD” programat exact în același interval (Luni 08:00–10:00), cu același profesor și în aceeași sală. Testul rulează fără erori și validează faptul că orarul este sincronizat corespunzător.

Al doilea test acoperă un caz **invalid**: aceeași activitate (ex: „Algebra”) este programată în zile diferite pentru două grupe (una Luni, alta Marți), deși ar trebui să fie comună. În acest caz, funcția de validare semnalează eroarea de sincronizare, afișând un mesaj specific în consola de ieșire (stdout), indicând neconcordanțele între grupele verificate.

Prin aceste teste, este asigurată consistența logică a orarului generat, mai ales în ceea ce privește respectarea regulilor de organizare academică. Astfel, se previn scenarii în care cursurile comune ar fi distribuite neuniform, ceea ce ar duce la conflicte reale în programarea activităților.

📌 **Ambele teste din test\_validator\_orar.py au fost executate cu succes**, confirmând că sistemul de validare detectează atât sincronizările corecte, cât și erorile evidente de aliniere între grupele unui an.

📸 *(Este recomandată atașarea unei capturi de ecran cu execuția testelor din acest fișier.)*

5.3 Testarea frontend

**Testarea componentelor frontend**

Testarea automată a componentelor frontend are un rol esențial în asigurarea funcționării corecte a interfeței utilizator. Într-o aplicație modernă construită cu React, testele frontend validează dacă elementele vizuale sunt afișate corect, dacă interacțiunile utilizatorului produc rezultatele așteptate și dacă starea aplicației se actualizează corespunzător în urma acestor interacțiuni.

Pentru acest proiect, testarea frontend a fost realizată folosind biblioteca **Jest**, împreună cu **React Testing Library**, două dintre cele mai populare unelte pentru testarea componentelor React. Acestea permit simularea comportamentului real al utilizatorului (de exemplu: apăsarea butoanelor, completarea câmpurilor, selectarea opțiunilor din formulare), precum și validarea conținutului vizual afișat în pagină.

Testele au fost organizate pe fișiere separate, câte unul pentru fiecare componentă importantă din aplicație, cum ar fi:

* pagina principală (Home.test.jsx),
* formularul de introducere a profesorilor (Profesori.test.jsx),
* formularul de setare reguli (SetareReguli.test.jsx),
* pagina de generare și afișare a orarului (GeneratedTimetable.test.jsx).

În cadrul fiecărui test, s-au verificat:

* prezența în DOM a elementelor cheie (titluri, butoane, formulare),
* comportamentul interactiv al utilizatorului,
* reacția componentelor la date de intrare și acțiuni,
* eventuale mesaje de eroare sau confirmare.

📌 Prin această suită de teste, se asigură că aplicația răspunde corect la acțiunile utilizatorului, iar modificările aduse interfeței nu introduc erori de afișare sau funcționare.

**5.3.1 Testarea componentei Home.jsx**

Componenta Home reprezintă pagina principală a aplicației și servește ca punct de intrare pentru utilizatori. Aceasta afișează diferit conținutul în funcție de starea de autentificare a utilizatorului – fie că este autentificat, fie că nu.

Pentru a verifica funcționalitatea acestei componente, au fost implementate teste automate folosind **Jest** și **React Testing Library**, împreună cu un sistem de mock pentru hook-ul useHomeLogic, care controlează logica de autentificare. Astfel, s-au putut simula ambele scenarii relevante pentru aplicație: utilizator neautentificat și utilizator autentificat.

În primul set de teste s-a validat comportamentul componentei în cazul în care **utilizatorul nu este autentificat**. Testele au verificat:

* prezența titlului aplicației în navbar;
* afișarea butonului de autentificare;
* mesajul principal care invită utilizatorul să se logheze;
* faptul că butonul „Începe generarea orarului” este dezactivat și este însoțit de un mesaj informativ care indică necesitatea autentificării.

Al doilea set de teste a vizat comportamentul componentei în cazul în care **utilizatorul este autentificat**. Aici s-a verificat:

* afișarea mesajului de bun venit cu numele utilizatorului;
* prezența butonului „Logout” și a linkului „Orarul meu”;
* activarea butonului „Începe generarea orarului”, care devine un link funcțional;
* afișarea unui mesaj care indică faptul că utilizatorul va fi redirecționat către platforma de generare a orarului.

Prin aceste teste s-a validat că interfața reacționează dinamic și corect în funcție de starea aplicației, iar experiența utilizatorului este adaptată în mod corespunzător. Testele contribuie la menținerea unei interfețe consistente și sigure din punct de vedere funcțional.

📌 **Toate testele din Home.test.jsx au fost executate cu succes**, confirmând funcționarea corectă a componentei în ambele situații (autentificat/neautentificat) și afișarea conținutului așteptat.

📸 *(Este recomandată includerea unei capturi de ecran cu testele bifate verde.)*

**5.3.2 Testarea componentei Profesori.jsx**

Componenta Profesori este responsabilă pentru gestionarea cadrelor didactice din aplicație. Aceasta permite introducerea de noi profesori, completarea disciplinei, setarea nivelului, tipului de activitate (curs/seminar/laborator), precum și a disponibilității săptămânale. Fiind una dintre cele mai complexe interfețe ale aplicației, a fost esențială acoperirea sa prin teste automate.

Testele au fost realizate folosind **Jest** și **React Testing Library**, cu simularea comportamentului logicii aplicației prin mock-uri ale hook-ului useProfesoriLogic. Acest lucru a permis testarea interfeței independent de backend, concentrându-se strict pe comportamentul vizual și interactiv.

Un prim test a verificat că pagina conține titlul principal și formularul de completare pentru un nou profesor, inclusiv câmpul „Nume complet” și butonul de salvare. Această verificare confirmă că utilizatorul poate începe procesul de adăugare în mod vizibil și accesibil.

Un alt test a simulat situația în care lista profesorilor este goală, confirmând că aplicația afișează mesajul „Niciun profesor găsit”. Astfel, utilizatorul primește un feedback clar într-un caz frecvent întâlnit – inițierea aplicației fără date.

S-a testat și comportamentul butonului „Salvează profesor”, unde s-a verificat că la apăsare se declanșează funcția adaugaProfesor. Acest test este crucial pentru a garanta că formularul este funcțional și că logica asociată este invocată la momentul corect.

De asemenea, testele au confirmat prezența funcționalității de adăugare a mai multor discipline, prin verificarea butonului „Adaugă altă disciplină”, cât și a structurii tabelare de selecție a disponibilității pe zile și intervale orare. Aceste elemente sunt esențiale pentru configurarea corectă a profesorilor, conform cerințelor reale de orar.

Prin această suită de teste, s-a validat funcționalitatea vizuală și interactivă a componentei, asigurându-se că toate secțiunile importante sunt afișate și funcționează corect, indiferent de starea aplicației sau de prezența datelor.

📌 **Toate testele din Profesori.test.jsx au fost executate cu succes**, confirmând stabilitatea și funcționarea completă a interfeței de gestionare a profesorilor.

📸 *(Se recomandă atașarea unei capturi de ecran cu testele bifate verde pentru această componentă.)*

**5.3.3 Testarea componentei SetareReguli.jsx**

Componenta SetareReguli joacă un rol important în procesul de generare a orarului, oferind utilizatorului posibilitatea de a configura și salva reguli personalizate care influențează direct structura orarului final. Această pagină permite scrierea regulilor în format text (JSON), denumirea acestora și salvarea într-o bază de date, pentru reutilizare ulterioară.

Testele automate pentru această componentă au fost implementate cu ajutorul bibliotecilor **Jest** și **React Testing Library**, folosind mock-uri pentru hook-ul useSetariReguli. Această abordare a permis testarea exclusivă a comportamentului vizual și interactiv al componentei, fără a depinde de backend sau de starea globală a aplicației.

Primul test a validat afișarea titlului paginii și a descrierii informative, prin care utilizatorul este ghidat să definească regulile de generare. Această verificare confirmă că interfața transmite corect scopul funcționalității și este intuitivă din perspectiva utilizatorului.

Un alt test a confirmat prezența câmpului pentru introducerea denumirii regulii, care are un placeholder sugestiv („Ex: Reguli orare Licență și Master”). Acest element este esențial pentru salvarea ulterioară și organizarea regulilor în sistem.

De asemenea, a fost testată și funcționalitatea butonului „Salvează”, prin simularea unui click și verificarea faptului că funcția salveazaReguli este apelată. Această verificare este crucială pentru a garanta că regulile pot fi persistate corect în baza de date.

Un ultim test a evaluat comportamentul aplicației în cazul în care nu există reguli salvate. Interfața afișează un mesaj informativ – „Nu există reguli salvate” – care ajută utilizatorul să înțeleagă că trebuie să creeze o regulă nouă pentru a putea continua.

Prin această serie de teste, a fost validată funcționalitatea critică a paginii SetareReguli, care are impact direct asupra modului în care se generează orarul. Testele contribuie la asigurarea unei experiențe de utilizare coerente și funcționale.

📌 **Toate testele din SetareReguli.test.jsx au fost executate cu succes**, confirmând funcționarea corectă a componentei în toate scenariile de utilizare testate.

📸 *(Este recomandată includerea unei capturi de ecran cu execuția testelor în interfața editorului.)*

**5.3.4 Testarea componentei GeneratedTimetable.jsx**

Componenta GeneratedTimetable este una dintre cele mai complexe din aplicație, fiind responsabilă pentru afișarea, generarea, validarea și gestionarea orarelor universitare. Aceasta permite utilizatorului să genereze automat orare fie cu ajutorul inteligenței artificiale (OpenAI), fie printr-un algoritm clasic propriu, să vizualizeze orarele salvate anterior, să le editeze, șteargă, filtreze și să le exporte.

Pentru a valida funcționalitatea acestei componente critice, s-au implementat teste automate folosind **Jest** și **React Testing Library**, cu mock-uri pentru toate hook-urile logice implicate în procesul de generare și manipulare a orarului. Această abordare a permis testarea izolată a interfeței, fără dependențe reale de rețea sau baze de date.

Un prim test a verificat afișarea corectă a elementelor de bază ale interfeței: titlul paginii „📅 Orar Generat” și butoanele principale pentru generarea orarului cu AI și prin algoritm clasic. Prezența acestor butoane este esențială pentru navigarea funcțională în aplicație.

Ulterior, s-a testat reacția componentelor la acțiunile utilizatorului. Prin simularea unui click pe butonul „Generează orar cu AI”, s-a confirmat că funcția genereazaOrar este apelată corect. În mod similar, click-ul pe butonul „Generează clasic” declanșează funcția genereazaOrarClasic, validând ambele ramuri ale funcționalității de generare.

S-au testat și elementele auxiliare de interfață, precum:

* butonul „🔄 Reîncarcă” care permite resetarea componentei,
* butonul „🔙 Înapoi” care facilitează întoarcerea la pașii anteriori,
* afișarea orarelor salvate anterior, cu opțiuni de editare, ștergere și reîncărcare,
* inputul de căutare pentru filtrarea rapidă a orarelor după denumire.

Aceste elemente contribuie la o experiență de utilizare fluentă și eficientă, iar testele automate confirmă că ele sunt prezente și funcționează corect în interfață.

Prin această suită de teste, s-a asigurat că pagina GeneratedTimetable răspunde corect atât la interacțiunile directe ale utilizatorului, cât și la stările interne ale aplicației, oferind un comportament robust și previzibil în procesul de generare a orarului.

📌 **Toate testele din GeneratedTimetable.test.jsx au fost executate cu succes**, confirmând funcționalitatea completă și corectă a componentei în scenariile testate.

📸 *(Se recomandă includerea unei capturi de ecran cu testele bifate verde din consola sau din IDE.)*