



**UNIVERSITATEA
TEHNICĂ
DIN CLUJ-NAPOCA**

**DOCUMENTATIE
PROIECT IS**

2025/2026

Aaron Husleag

SCURTA DESCRIERE A PROIECTULUI

Am propus sa fac proiectul **Window Configurator Mark1** care este o aplicatie web moderna dezvoltata pentru configurarea personalizata a ferestrelor. De ce Mark 1 este pentru ca am inceput proiectul asta pentru o firma si dupa multe modificari s-a ajuns sa fac alta aplicatie intreaga care este numita Mark2.

Aplicatia permite utilizatorilor sa aleaga diverse optiuni pentru ferestrele lor printr-un proces pas cu pas intuitiv, incluzand mai multe optiuni unele rezervate la unele tipuri de ferestre ca de exemplu:

- Tipul de fereastra (Fereastra simpla, Fereastra culisanta, Usa Fereastra, Fereastra fixa)
- Materialul (PVC, Aluminiu, Lemn)
- Dimensiunile (latime si inaltime)
- Culoare (exterior si interior)
- Numarul de panouri
- Tipul de deschidere (pentru anumite tipuri de ferestre)

Dupa finalizarea configuratiei, utilizatorii pot genera un PDF cu devis-ul complet al ferestrei configurate. Aplicatia este construita folosind **Next.js (include react si node.js)** pentru frontend si ofera o experienta utilizator moderna si responsiva datorita multimei de clase CSS disponibile.

TEHNOLOGIILE FOLOSITE

Principala tehnologie folosita : **Next.js**

- Framework React pentru aplicatii web moderne
- Justificare: Next.js a fost ales pentru mai multe motive. In primul rand, este o tehnologie SEO-friendly din ce stiu, ceea ce permite indexarea mai buna de catre motoarele de cautare ca Google. In al doilea rand, am ales sa invat Next.js pentru ca este o tehnologie foarte cautata pe piata muncii si reprezinta un skill important pentru dezvoltatorii web. De asemenea, Next.js ofera routing automat bazat pe structura de fisiere si optimizari de performanta integrate. Un avantaj suplimentar este ca aplicatiile Next.js pot fi hostate gratuit pe Vercel, ceea ce este ideal pentru un portfolio viitor si pentru prezentarea proiectelor.

React (inclus in next.js)

- Biblioteca JavaScript pentru construirea interfetei utilizatorului
- Justificare: React este popular pentru aplicatii web interactive. Permite crearea de componente reutilizabile, gestionarea eficienta a starii aplicatiei, si ofera un ecosistem vast de biblioteci si resurse. React 19 aduce imbunatatiri de performanta si noi feature-uri pentru gestionarea starii.

TypeScript 5

- Suprapunere de tipuri pentru JavaScript
- Justificare: TypeScript a fost esential pentru mentinerea calitatii codului intr-un proiect cu multiple componente si logica complexa. Ajuta la detectarea erorilor in timpul dezvoltarii, ofera autocompletare mai buna in IDE, si face codul mai usor de inteles si mentinut, mai ales pentru gestionarea configuratiei ferestrei care are multiple proprietati tipizate. Component-based architecture

Tailwind CSS 4

- Framework CSS utility-first pentru stilizare
- Justificare: Tailwind CSS permite dezvoltare rapida a interfetei fara a scrie CSS custom pentru fiecare componenta. Clasele utility permit stilizare directa in JSX, reducand timpul de dezvoltare si mentinand consistenta designului.

Node.js

- Mediu de rulare JavaScript
- Justificare: Node.js este necesar pentru rularea Next.js si pentru tooling-ul de dezvoltare. Permite rularea JavaScript in server-side.

npm

- Manager de pachete
- Justificare: npm este managerul standard de pachete pentru Node.js si permite instalarea si gestionarea tuturor dependintelor proiectului intr-un mod eficient si reproducibil.

CERINTE FUNCTIONALE

Aplicatia ofera un proces pas cu pas pentru configurarea personalizata a ferestrelor, permitand utilizatorilor sa aleaga diverse optiuni si sa genereze un devis complet in format PDF.

3.1 Configurare Tip Fereastra

Primul pas in procesul de configurare implica selectarea tipului de fereastra dorit. Sistemul ofera patru optiuni principale: Fereastra simpla, Fereastra culisanta, Usa Fereastra si Fereastra fixa. Aceasta alegere este fundamentala deoarece influenteaza disponibilitatea optiunilor ulterioare, precum numarul de panouri si tipurile de deschidere disponibile. Fiecare tip de fereastra este prezentat vizual cu o imagine reprezentativa, facilitand procesul de alegere pentru utilizator.

3.2 Configurare Material

Utilizatorii pot selecta materialul din care va fi realizata fereastra din trei optiuni disponibile: PVC, Aluminiu sau Lemn. Fiecare material are caracteristici specifice si influenteaza atat aspectul estetic cat si performantele termice ale ferestrei. Selectia materialului este prezentata prin imagini clare care permit utilizatorului sa vizualizeze diferenta intre optiuni. In aplicatia vanduta m am gandit sa planificat implementarea unei interfeete administrative care permite vanzatorului sa gestioneze optiunile de materiale dintr-o baza de date. Aceasta solutie ar transforma aplicatia intr-un produs mai polivalent, oferind posibilitatea personalizarii ofertei fara intervenire tehnica, adaptandu-se astfel la cerintele specifice ale fiecaruia.

3.3 Configurare Dimensiuni si Culori

Utilizatorii pot seta dimensiunile ferestrei prin slider-uri interactive: latimea (100-300 cm, implicit 200 cm) si inaltimea (50-200 cm, implicit 125 cm). Dimensiunile influenteaza calculul pretului final bazat pe suprafata. Pentru culori, aplicatia ofera selectie separata pentru partea exterioara si interioara din cinci optiuni standard (Alb Ral, Gri Antracit Ral, Gri Tip Ral, Gri Deschis Ral, Negru Grafit Ral), fiecare reprezentata vizual printr-un card colorat. Culorile non-standard pot implica un cost suplimentar.

3.1.4 Trimitere Email cu Variabile de Mediu

Pentru trimitera devis-ului prin email, aplicatia utilizeaza Nodemailer cu configuratie SMTP prin variabile de mediu pentru securitate. Credentialele SMTP (adresa email si parola aplicatiei) sunt stocate in variabile de mediu (SMTP_USER si SMTP_PASS), evitand expunerea acestora in cod. La trimitera formularului, aplicatia genereaza un PDF cu configuratia ferestrei si il ataseaza la email-ul trimis catre client. Variabilele de mediu permit configurarea mediul local si mediu de productie, asigurand securitate.

CERINTE FUNCTIONALE

3.1.5 Configurare Numar Panouri

Pentru anumite tipuri de ferestre (Fereastra simpla si Fereastra culisanta), utilizatorii pot selecta numarul de panouri (1, 2 sau 3 panouri), optiune care nu este disponibila pentru Usa Fereastra sau Fereastra fixa. Numarul de panouri selectat determina optiunile de deschidere disponibile si influenteaza calculul pretului final prin aplicarea unui multiplicator. Dupa selectarea numarului de panouri, utilizatorii pot alege tipul de deschidere, optiunile variind in functie de tipul ferestrei si numarul de panouri (de la patru optiuni pentru o fereastra simpla cu un panou, pana la douasprezece optiuni pentru o fereastra simpla cu trei panouri). Fiecare optiune este ilustrata cu un desen schematic care arata modul exact de functionare.

3.6 Validare Configuratie si Navigare Intre Pasi

Sistemul implementeaza o validare continua care asigura completarea tuturor campurilor obligatorii inainte de avansarea la pasul urmator. Daca utilizatorul incercă sa acceseze un pas ulterior fara sa fi completat pasii anteriori, sistemul il redirectioneaza automat catre primul pas incomplet, prevenind astfel generarea de devis-uri incomplete. Aplicatia permite navigarea bidirectionala: utilizatorii pot avansa doar daca pasul curent este completat corect, dar pot reveni la orice pas anterior pentru modificari. Butoanele de navigare sunt disponibile pe fiecare pagina, cu exceptia primului pas unde butonul "Inapoi" este dezactivat, oferind flexibilitate pentru experimentarea cu diferite combinatii de optiuni fara a reinncepe procesul.

3.7 Vizualizare Configuratie Finala

Dupa finalizarea tuturor pasilor de configurare, utilizatorul este redirectionat catre pagina finala unde poate vizualiza un rezumat complet al configuratiei. Aceasta pagina afiseaza toate alegerile facute: tipul ferestrei, materialul, dimensiunile, culorile, numarul de panouri si tipul de deschidere (daca este aplicabil). In plus, pagina include o reprezentare vizuala a ferestrei configurate, care poate fi fie imaginea tipului de fereastra (pentru ferestre fixe sau usi-fereastra), fie imaginea tipului de deschidere selectat (pentru ferestrele cu panouri multiple).

3.8 Generare PDF

Dupa vizualizarea configuratiei finale, utilizatorii pot genera un document PDF care contine devis-ul complet. PDF-ul este generat dinamic folosind biblioteca jsPDF si include toate detaliile configuratiei: tipul ferestrei, materialul, dimensiunile exacte, culorile selectate, numarul de panouri si tipul de deschidere. Documentul este formatat profesional cu un header personalizat si toate informatiile organizate clar. PDF-ul este descarcat automat pe dispozitivul utilizatorului cu numele "devis-fenetre.pdf".

CERINTE NON FUNCTIONALE

Cerintele non-functiionale definesc caracteristicile calitative ale aplicatiei Window Configurator Mark1, precum securitatea, usabilitatea si mentenabilitatea. Aceste cerinte sunt esentiale pentru asigurarea unei experiente utilizator optimale si pentru garantarea calitatii produsului software in conditii reale de utilizare. In plus, aplicatia a fost conceputa cu accent pe reutilizabilitate, permitand modificarea eficienta a stilului si tematicii pentru a putea fi revanduta catre alte intreprinderi fara necesitatea unei reimplementari complete.

4.1 Usabilitate si Experienta Utilizator

Interfata aplicatiei trebuie sa fie intuitiva si usor de folosit, permitand utilizatorilor sa navigheze prin procesul de configurare fara necesitatea instructiuni. Procesul de configurare trebuie sa fie clar si ghidat pas cu pas, cu indicatii vizuale clare pentru fiecare etapa.

4.2 Responsivitate si Compatibilitate

Aplicatia trebuie sa functioneze corect pe multiple platforme si dispozitive. Layout-ul trebuie sa se adapteze automat la diferite dimensiuni de ecran de desktop-uri mari si mici.

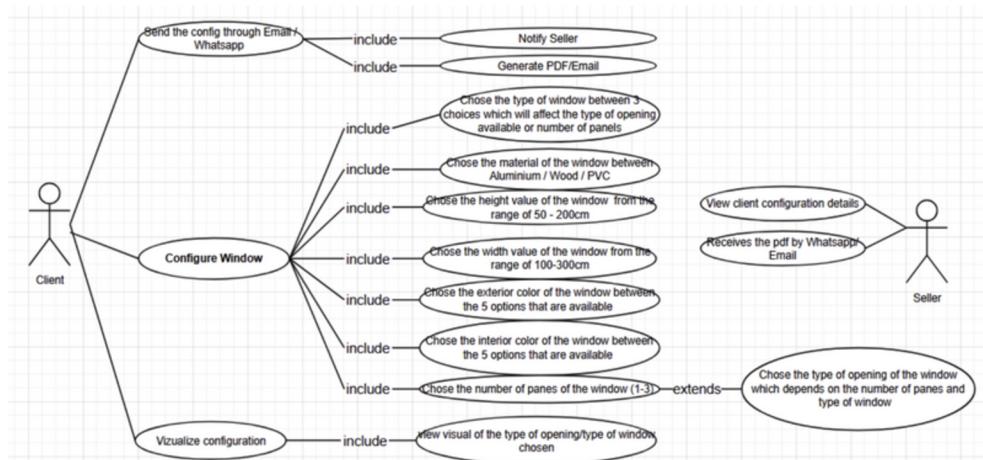
4.3 Mantinabilitate si Extensibilitate

Codul aplicatiei trebuie sa fie bine structurat, facilitand intretinerea si dezvoltarea ulterioara. Componentele trebuie sa fie reutilizabile si modulare, permitand modificari si extinderi fara a afecta functionalitatile existente. Arhitectura aplicatiei trebuie sa permita adaugarea de noi functionalitati (new steps) sau modificarea celor existente cu un impact minim asupra codului deja implementat, respectand principiul Open/Closed din SOLID. Aplicatia a fost proiectata cu accent pe reutilizabilitate, structura modulara si separarea intre logica de business si prezentare permitand modificarea eficienta a stilului, culorilor si tematicii pentru adaptarea la identitatea vizuala a diferitelor intreprinderi. Aceasta abordare faciliteaza revanzarea aplicatiei catre alte companii fara necesitatea unei reimplementari complete, reducand semnificativ timpul si costurile de personalizare.

4.5 Accesibilitate

Aplicatia trebuie sa fie accesibila pentru utilizatori cu diferite nevoi. Suportul pentru navigare prin tastatura permite utilizatorilor care nu pot folosi mouse-ul sa acceseze toate functionalitatatile aplicatiei. Contrastul adevarat pentru text asigura lizibilitatea clara.

DIAGRAMA USE CASE



Aaron Husleag

DESIGN PATTERNS

Aplicatia Window Configurator Mark1 utilizeaza mai multe design patterns pentru a rezolva probleme specifice de arhitectura si pentru a asigura cod de calitate, mentenabil si extensibil. Aceste patterns au fost alese pentru a gestiona complexitatea aplicatiei si pentru a facilita evolutia viitoare a produsului.

6.1 Singleton Pattern – Gestionarea Configuratiei Unice

- 6.1.1 Problema Identificata

In aplicatie, configuratia ferestrei trebuie sa fie accesibila din multiple componente si pagini diferite (step-1 pana la step-8 si pagina finala), dar trebuie sa existe o singura instanta a acestei configuratii pe parcursul intregii sesiuni de configurare. Fara un pattern adevarat, s-ar putea crea multiple instante ale configuratiei in diferite componente, ceea ce ar duce la inconsistente de date si pierderea starii utilizatorului. De exemplu, daca utilizatorul selecteaza tipul de fereastra in step-1, aceasta informatie trebuie sa fie disponibila in toate paginile ulterioare fara a fi pierduta sau duplicata.

- 6.1.2 Rezolvarea Problemei prin Singleton Pattern

Singleton Pattern asigura ca exista **o singura instanta a clasei WindowConfiguration** in intreaga aplicatie. Aceasta instanta este exportata ca un obiect constant si accesibila global prin import, garantand ca toate componentele folosesc aceeași configuratie si ca starea este pastrata consistenta pe parcursul intregului proces de configurare. Pattern-ul previne crearea accidentală a multiple instantane si asigura ca modificari facute intr-o pagina sunt imediat vizibile in toate celelalte pagini.

- 6.1.3 Implementare

Implementarea Singleton Pattern se bazeaza pe urmatoarea structura:

1) Clasa WindowConfiguration (--> lib/models/WindowConfiguration.ts)

Clasa contine toate proprietatile configuratiei ferestrei (type, material, width, height, colorExterior, colorInterior, numberOfPanels, openingType) ca atribute private, cu getters si setters pentru acces controlat. Constructorul initializeaza valorile implice pentru width (200 cm) si height (125), asigurand o configurare valida chiar daca utilizatorul nu modifica aceste valori. Metoda isComplete() verifica daca toate campurile obligatorii sunt completate, avand in vedere ca anumite tipuri de ferestre (Usa Fereastra, Fereastra fixa) nu necesita toate campurile.

2) Instanta Unica (--> lib/config.ts)

```
lib > TS config.ts > ...
1  import { WindowConfiguration } from '@lib/models/WindowConfiguration'
2
3  export const config = new WindowConfiguration() // config instance
```

Singleton Pattern ofera multiple avantaje pentru aplicatie. Consistenta, Accesibilitatea. Initializarea valorilor implice in constructor asigura ca aplicatia functioneaza corect chiar daca utilizatorul nu completeaza toate campurile. In plus, pattern-ul faciliteaza validarea centralizata a configuratiei prin metoda isComplete(), care poate fi apelata ori-unde.

DESIGN PATTERNS

3) Utilizare in Componente

```
app > configurator > step-1 > page.tsx > Step1Page
10  export default function Step1Page() {
11
12      setSelectedOptionIndex(index)
13      config.type = WindowTypeOptions[index].label
14
15  }
```

6.2 Strategy Pattern – Selectia Dinamica a Opsiunilor de Deschidere

- 6.2.1 Problema Identificata

Opsiunile de tip de deschidere pentru ferestre variază în funcție de două criterii: tipul ferestrei (simplă sau culisantă) și numărul de panouri (1, 2 sau 3). De exemplu, o fereastră simplă cu un panou are patru opțiuni de deschidere, în timp ce o fereastră simplă cu trei panouri are douăzeci și patru opțiuni diferite. Fără un pattern adecvat, ar **trebui să se scrie logica conditională complexă și repetitivă în fiecare componentă** care necesită aceste opțiuni, ceea ce ar duce la cod duplicat și dificil de întreținut. Adăugarea unei noi combinații tip/panouri sau modificarea opțiunilor existente ar necesita modificări în multiple locuri din cod.

- 6.2.2 Rezolvarea Problemei prin Strategy Pattern

Strategy Pattern permite selectarea algoritmului corect (setul de opțiuni) în timpul execuției, bazat pe parametrii de intrare. Fiecare strategie (set de opțiuni) este encapsulată într-o structură de date separată, iar funcția `getOpeningTypeOptions` selectează strategia corectă bazându-se pe tipul ferestrei și numărul de panouri. Această abordare permite adăugarea de noi strategii fără modificarea codului existent și simplifică testarea fiecărei strategie independent.

- 6.2.3 Implementare

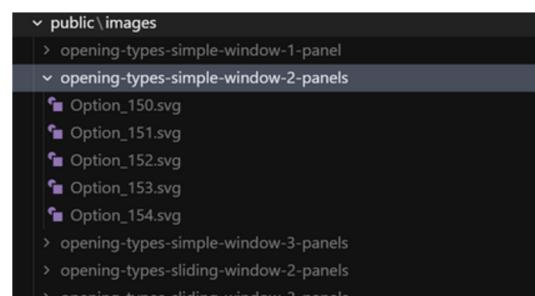
Implementarea Strategy Pattern pentru sistemul de opțiuni de deschidere se bazează pe următoarea structură:

1) Strategiile Concrete (Seturi de Opțiuni)

Fiecare set de opțiuni este definit ca o constantă separată în `data/opening-options.ts`:

Fiecare opțiune este salvată în format SVG, acesta fiind formatul ideal pentru imaginilor simple.

```
data > TS opening-options.ts > SimpleWindow2PanelsOptions > img
Review Next
1 const SimpleWindow1PanelOptions = [
2     { label: 'Option 1', img: '/images/opening-types-simple-window-1-panel/Option_144.svg' },
3     { label: 'Option 2', img: '/images/opening-types-simple-window-1-panel/Option_145.svg' },
4     { label: 'Option 3', img: '/images/opening-types-simple-window-1-panel/Option_146.svg' },
5     { label: 'Option 4', img: '/images/opening-types-simple-window-1-panel/Option_147.svg' },
6 ]
7
8 const SimpleWindow2PanelsOptions = [
9     { label: 'Option 1', img: '/images/opening-types-simple-window-2-panels/Option_150.svg' },
10    { label: 'Option 2', img: '/images/opening-types-simple-window-2-panels/Option_151.svg' },
11    { label: 'Option 3', img: '/images/opening-types-simple-window-2-panels/Option_152.svg' },
12    { label: 'Option 4', img: '/images/opening-types-simple-window-2-panels/Option_153.svg' },
13    { label: 'Option 5', img: '/images/opening-types-simple-window-2-panels/Option_154.svg' },
14 ]
15
16 const SimpleWindow3PanelsOptions = [
17     { label: 'Option 1', img: '/images/opening-types-simple-window-3-panels/Option_155.svg' },
18     { label: 'Option 2', img: '/images/opening-types-simple-window-3-panels/Option_156.svg' },
19     { label: 'Option 3', img: '/images/opening-types-simple-window-3-panels/Option_157.svg' },
20     { label: 'Option 4', img: '/images/opening-types-simple-window-3-panels/Option_158.svg' },
```



DESIGN PATTERNS

2) Functia de Selectie a Strategiei

Functia getOpeningTypeOptions selecteaza strategia corecta bazandu-se pe tipul ferestrei si numarul de panouri:

```
51 | export function getOpeningTypeOptions(windowType?: string, numberOfPanels?: string): OpeningOption[] {  
52 |   if (windowType === 'Fereastra simpla' && numberOfPanels === '1 panou') {return SimpleWindow1PanelOptions}  
53 |   if (windowType === 'Fereastra simpla' && numberOfPanels === '2 panouri') {return SimpleWindow2PanelsOptions}  
54 |   if (windowType === 'Fereastra simpla' && numberOfPanels === '3 panouri') {return SimpleWindow3PanelsOptions}  
55 |   if (windowType === 'Fereastra culisanta' && numberOfPanels === '2 panouri') {return SlidingWindow2PanelsOptions}  
56 |   if (windowType === 'Fereastra culisanta' && numberOfPanels === '3 panouri') {return SlidingWindow3PanelsOptions}  
57 |  
58 | }
```

3) Utilizare in Componente

In pagina step-8, strategia este selectata dinamic:

```
app > configurator > step-8 > page.tsx > Step8Page  
12  export default function Step8Page() {  
32  |   const openingOptions = useMemo(() => {  
33  |     return getOpeningTypeOptions(config.type, config.numberOfPanels)  
34  |   }, [config.type, config.numberOfPanels])
```

- 6.2.4 Avantaje si Beneficii

Strategy Pattern ofera extensibilitate (adaugarea de noi combinatii necesita doar o noua constanta si o conditie), testare izolata a fiecarei strategii, si flexibilitate runtime pentru oferirea de optiuni contextuale utilizatorilor.

6.3 Component Pattern - Componente React Reutilizabile

- 6.3.1 Problema Identificata

Interfata necesita afisarea si gestionarea selectiei optiunilor in multiple locuri (step-1 pana la step-8). Fara componentizare, codul ar fi duplicat in fiecare pagina, cu aceeasi logica de afisare, selectie si stilizare, facand aplicatia dificil de intretinut.

- 6.3.2 Rezolvare Problemei

Component Pattern permite crearea de componente reutilizabile (OptionCardGrid, AppNavigation) care encapsuleaza logica si prezentarea, reducand duplicarea codului si facilitand mentenanta prin modificari centralizate.

- 6.3.3 Implementare

Principalele componente sunt:

1. **OptionCardGrid** (components/OptionCardGrid.tsx) – componenta generica pentru afisarea unei grile de optiuni selectable, cu props pentru personalizare (dimensiuni imagini, inaltime carduri).

2. **AppNavigation** (components/AppNavigation.tsx) – componenta reutilizabila pentru navigare intre pasi.

Utilizare: Componentele sunt folosite consistent in toate paginile de configurare, cu date diferite transmise prin props.

CONCLUZIE

Acest proiect m-a interesat foarte mult, mai ales pentru ca am avut ocazia sa il programez de trei ori: prima data pentru acest devoir, apoi pentru un client, si apoi pentru un alt client. Aceasta evolutie iterativa mi-a permis sa observ si sa invat din greselile realizate initial.

Prima problema majora a fost gestionarea paginilor multiple, care a fost rezolvata prin simplificarea logicii intr-o singura pagina. Apoi a urmat problema afisarii mobile, care necesita adaptarea interfetei pentru diferite dimensiuni de ecran. Odata rezolvata aceasta, am abordat aspectele de design, adaugand feature-uri precum o bara de progres si animatii pentru a imbunatati experienta utilizatorului.

In final, problema cea mai complexa a fost depoluarea aplicatiei. In mediul local, rularea aplicatiei este mai simpla deoarece *npm run dev* este mai permisiv, in timp ce **Vercel**, care gazduieste gratuit aplicatiile mele, are cerinte mai stricte de validare si optimizare. Aceasta diferența mi-a permis sa invat importanta testarii si optimizarii codului pentru productie, nu doar pentru mediul de dezvoltare local.

