3D Sneaker Store - Full Briefing

Flux.be heeft ons gecontacteerd met de vraag om voor een Sneakermerk uit London onderstaande opdracht uit te voeren.

Bekijk alvast even volgende korte briefing van Stijn, CEO van Flux:

<https://vimeo.com/868216274/9b168203eb?share=copy>

**Het project bestaat uit drie componenten, die uiteindelijk op elkaar moeten aansluiten.**

1. Threejs sneaker configurator **(THREEJS)**
2. Nodejs API die bestellingen kan ontvangen, eventuele data kan aanleveren … **(NODEJS)**
3. VueJS app die als backend dient om bestellingen te bekijken en te beheren **(VUEJS)**

Afbeelding met schoen, Wandelschoenen, sneakers, schoeisel

Door AI gegenereerde inhoud is mogelijk onjuist.Briefing FLUX.be

Briefing: 3D Configurator voor Sneakers op Maat

**Achtergrond**

-       Wij zijn een bedrijf dat sneakers op maat aanbiedt.

-       We hanteren een 'on demand' aanpak zonder voorraad om overproductie te vermijden en onze ecologische voetafdruk te minimaliseren.

-       Onze sneakers zijn van hoge kwaliteit en we willen klanten de mogelijkheid bieden om hun persoonlijkheid te uiten via hun schoenkeuze.

**Doel**

-       Ontwikkel een 3D configurator die klanten in staat stelt om hun sneakers te personaliseren.

**Eisen:**

-       **Gebruiksvriendelijkheid**: De configurator moet intuïtief en eenvoudig te gebruiken zijn.

-       **Doelgroep**: Gericht op een jong publiek. De interface moet modern en aantrekkelijk zijn, zonder overbodige complexiteit.

-       **Personalisatie**: Klanten moeten de kleur van verschillende onderdelen van de sneaker kunnen aanpassen, zoals: zool, veters,…

**Optionele Features**:

-       Voeg extra elementen toe aan de configurator om het aantrekkelijker te maken, zoals bv een skateboard.

-       Bied extra aanpassingsmogelijkheden zoals:

o   Andere materialen

o   Charms (extra kleine details die je aan je schoen kan hangen)

o   Patronen

o   Toevoegen van tekst

-       Een 3D model van een schoen dat jullie zelf van het internet downloaden en voorbereiden.

**Technische eisen:**

-       Op basis van Three.js

o   We leveren een 3D model aan van een schoen.

-       Mogelijkheid om snapshots van de gepersonaliseerde schoen te nemen.

-       Koppeling Node.js API / Web App

-    Prototype staat live en kan door de klant getest worden

**Voorbeeld**:<https://www.nike.com/be/u/custom-nike-dunk-low-by-you-su24-10001873/8905670869#Builder>

**Samenvatting**:

We zoeken een 3D configurator die onze klanten in staat stelt om sneakers te personaliseren op een eenvoudige en intuïtieve manier. De tool moet modern en aantrekkelijk zijn, met de mogelijkheid om verschillende aspecten van de sneaker aan te passen, snapshots te nemen en direct een bestelling te plaatsen. Optionele functies zoals extra aanpassingsmogelijkheden en leuke elementen zoals skateboards kunnen worden overwogen om de gebruikerservaring te verbeteren. De bestellingen komen binnen in een VueJS web app met als backend een nodejs API met MongoDB databank.

**Evaluatiecriteria:**

* **Technische werking:** hoe goed werkt jullie oplossing om de schoenen te configureren én te bestellen
* **Conceptueel:** hoe goed sluit jullie uitwerking aan bij het doelpubliek
* **Detail:** met welk oog voor detail hebben jullie de uitwerking gedaan (vb: sonic branding? Environment maps? Error messages? Empty states?)
* **Performance:** hebben jullie stilgestaan bij performance en welke bewuste optimalisaties hebben jullie gedaan?

1 / 3D Sneaker Configurator **THREEJS**

* We krijgen een sneakermodel aangeleverd (zie Canvas), maar je mag andere bijkomende modellen gebruiken indien gewenst
* Het is niet de bedoeling dat jullie in Blender zelf een sneaker vanaf nul gaan uitwerken, ga op zoek naar een geschikt model of gebruik het model dat we hebben aangeleverd.
* Minstens één sneakermodel moet geconfigureerd kunnen worden
  + Veters
  + Zool
  + Extra optie 1
  + Extra optie 2
* Wie wil, mag meerdere opties of modellen voorzien
* Maak het niet te saai, het merk wil graag een statement maken
* Zorg voor een complete uitwerking van de configurator, zodanig dat deze kan aansluiten op de API uit punt 2. Teamwork en goede communicatie is noodzakelijk om een briefing als deze tot een goed einde te brengen.
* Denk aan realisme, maar ook aan performance: shadows, shadowMaps, fake lights, textures, environment maps, …
* Optioneel: *Sonic Branding* - pas de inspiratie die je hebt opgedaan uit de gastles Sonic Branding toe in je winkel, hoe klinkt dit merk?
* Optioneel: *A.I. art direction* - gebruik A.I. tools zoals Midjourney, Dall-E of Stable Diffusion om visuals te maken die bij het merk passen (vb voor je environment maps of de algemene look & feel van je shop)

2 / NodeJS API **NODEJS**

API

* API/v1
  + POST /orders
    - Voegt een bestelling toe aan je systeem met de nodige configuratiegegevens (je kan die faken tot bij de finale integratie)
      * Deze gegevens kunnen gaan van de kleur van de veters, tot de maat van de schoen
    - Vergeet de contactgegevens van je klant niet te bewaren

* DELETE /orders/{id}
  + Verwijdert een schoen/bestelling
  + Enkel admins kunnen dit (authenticatie is hier nodig)
* PUT/PATCH (zoek op!) /orders/{id}
  + Update de status van een bestelling
    - In productie / verzonden / …
    - Enkel admins kunnen dit
* GET /orders/{id}
  + Haalt details op voor één specifieke schoen
  + Deze route kan je gebruiken om bijvoorbeeld details van 1 schoen te tonen (vb delen van je schoen op social media)
* GET /orders
  + Haalt alle bestellingen op
  + Hier kan je eventuele parameters meegeven
    - ?sortby=votes|date
* /users
  + Nu je de basisroutes kent voor het gedeelte dat met de bestellingen te maken heeft, kan je zelf nadenken over welke routes nodig zijn om je gebruik te laten aanloggen, uitloggen, wachtwoord updaten, …

* Je API zou enkel aanspreekbaar mogen zijn als een gebruiker een correct token heeft, vooral op de routes waarbij je gegevens van je klanten gaat ophalen (vb: /shoes)
  + Lees over JWT of bekijk de videos over dit topic
  + Gebruik de JSEND structuur voor je respones <https://github.com/omniti-labs/jsend>
  + Om bestellingen te kunnen bekijken als beheerder moet je eerst inloggen, dit kan met een fixed account (geen registration page nodig)
  + Als je met een fixed account werkt, zorg dan wel dat het wachtwoord eenvoudig gewijzigd kan worden vanuit je user interface
    - Misschien heb je hiervoor een nieuwe API-route nodig
  + <https://medium.com/dev-bits/a-guide-for-adding-jwt-token-based-authentication-to-your-single-page-nodejs-applications-c403f7cf04f4>
  + Als je geen controles voorziet kan iemand schoenen verwijderen uit je API,wachtwoorden wijzigen, …

3 / VueJS frontend **VUEJS**

* De front-end wordt gebouwd met Vue.js als framework
  + Werk je schermen eerst ruw uit in Figma (of *insert your favourite tool here*), maar spendeer daar maximaal twee uur tijd aan.
* Je werkt met **BEM of TailwindCSS** voor je CSS structuur
* De layout werkt responsive, maar dat is vanzelfsprekend
* In dit vak testen we enkel binnen de laatste versie van Chrome. Je mag gebruik maken van de laatste CSS en JS-technieken.
* Gebruik van Tailwind CSS of andere component libraries is toegestaan en wordt ook aangeraden, maar het is niet verplicht, belangrijk is dat je tot een werkende oplossing komt voor je klant.
* Werk met VueJS Components (Denk zelf uit welk componenten je nodig hebt? Vb: Orders, Order, Shoe …)
* Bekijk de documentatie over Vue Router, zodat je makkelijk van het ene scherm naar het andere kan gaan zonder een page refresh (<https://router.vuejs.org/>)
* Functionaliteit
  + Als admin moet je kunnen aanloggen
  + Na het aanloggen krijg je alle bestellingen te zien (ik zou hier navigeren via de view router)
  + Als er een nieuwe bestelling geplaatst wordt, dan zie je die  live  binnenkomen
    - Je verhoogt bovendien een  live  teller met het aantal totale bestellingen
  + Je kan de details van één bestelling bekijken (vb router: /#/orders/3)
  + Je kan een bestelling markeren als
    - Verzonden, geannuleerd, in productie, …
    - Als deze status aangepast wordt, dan zie je de wijzigingen  live  in het dashboard
  + Je kan een bestelling uit het systeem verwijderen
  + Als admin kan je je wachtwoord wijzigen (maak een aparte component aan, werkt met bcrytp)
  + Alle data komt van je API of gaat naar je API

3 / Technische vereisten

* Je werkt lokaal aan je app, maar zorgt er ook voor dat je ze eenvoudig online kan *deployen* naar een cloud service zoals Vercel of Render. Je app moet online werken en testbaar zijn. Render.com is een prima optie voor je API. Vercel is eveneens handig om je frontend moeiteloos te deployen.
* Je werkt op basis van een API die je bouwt met node.js
  + Vanuit je frontend kan je requests doen naar je API via fetch()
* De databank die we gebruiken is **MongoDB**
* Als hosting kan je gebruik maken van https://render.com/ voor je node app en van https://www.mongodb.com/cloud/atlas om je databank gratis te hosten
* De backend is geschreven in **nodejs** in combinatie met het **express framework**
* Voor het live gedeelte maken we gebruik van websockets met behulp van **primus**
* We maken vanaf dag 1 gebruik van **GIT, afwijken hiervan is niet toegestaan**
* Sommige features krijgen het label **LIVE**  — dat wil zeggen dat je gebruik moet maken van **Web Sockets** (bvb met Primus of Socket.io) om live bij iedere geconnecteerde gebruiker een actie te triggeren
* Deployment doe je mogelijk als volgt
  + Je frontend is een apart GIT project en deploy je naar Netlify.com, Render.com of Vercel
  + Je backend is een apart GIT project en deploy je naar render.com (of eender welke nodejs hosting)
* Commit regelmatig, GIT is je bewijs van opgeleverd werk. Als je niet correct met GIT werkt heb je geen bewijs van je eigen werk. Het vaak gehoorde “mijn computer is gecrashed en ik heb alles in één keer overgezet naar een nieuwe GIT repo” gaat niet op. Goede, kleine, regelmatige commits in GIT, vanaf dag 1.   
  **Geen vragen = Geen problemen**

**Tips:**

* Begin met kleine stappen en bouw je complexiteit en volgorde op:
* Bouw eerst je API uit (zonder authenticatie)
  + Test je API via postman, voor je een front-end bouwt
* Bouw dan pas je front-end uit
* Probeer al eens via de app een bestelling door te sturen naar de API
  + Wordt deze bestelling goed bewaard in de databank?
  + Kan je via console.log de bestelling binnenkomen in je controller in node.js?
* Voeg stilaan BEM of Tailwind toe voor je CSS-opmaak
* Voeg authenticatie toe (vb JWT token voor beheerders)
* *… commit commit commit*