Semester 5
17 june 2024

DESIGN DOCUMENT

Product Suggestion Tool for Low Pressure Studio B.V.

Adam Wiszniowski - Świder

TABLE OF CONTENTS

INTRODUCTION

Low Pressure Studio (LPS), with over two decades of experience, operates under the brands Bataleon, Rome, and Lobster, specializing in snowboarding equipment tailored to diverse user needs. Currently, the Bataleon shop offers a wide range of products, but beginners often find it difficult to understand options like snowboard flexibility, while advanced users seek detailed specifications. This mismatch leads to poor user experiences, negative reviews, and returns.

To resolve these issues, a question-based product suggestion tool is proposed for the Bataleon website. This tool will ask 3 to 6 questions based on the user's proficiency level. Beginners will answer basic questions about terrain, boot size, and weight, while intermediate and advanced users will be asked more detailed questions about flexibility, camber, and board shape. This approach aims to improve the clarity and relevance of product information, enhancing customer satisfaction and reducing returns by ensuring users find the snowboard best suited to their needs.

The goal of this research is to help answer the main research question which is “How can we make the best snowboard suggestion tool for snowboarders of different levels which is based on available stock and user input?” as well as directly answer research subquestions 5. How can we use user input to find appropriate products in Bataleon DataBase, within the Shopify environment? and 6. How can we implement available stock into the product suggestion tool, keeping in mind that the amount of available stock will change over time?

CONTEXT

Low Pressure Studio has over 20 years of experience in creating quality snowboarding equipment. Currently it’s selling under 3 brands: Bataleon, Rome and Lobster. Low Pressure Studio's practice of tailoring designs and products to distinct target users facilitates portfolio diversification and enhances the ability to cater to individual needs effectively. This approach ensures the provision of products that resonate more closely with their unique preferences and aesthetics. My tasks are going to be oriented around the Bataleon website. Currently Bataleon shop offers many options for Snowboard equipment for many different users. Problem however presents itself when beginner users are faced with sophisticated

options to choose from, such as snowboard flexibility. Term itself is pretty self explanatory, but the meaning behind it would be hard to grasp by someone with no snowboarding experience. On the other hand, more experienced users might want more specific parameters that would better accommodate their riding style, a need that also needs to be fulfilled. To resolve this issue a solution is needed that will help both experienced and beginner users. Common solutions to similar problems are often resolved by some kind of filter function included on a main page.

GOALS

The problem that Low Pressure Studio is trying to solve is how to make the best match between a user and a snowboard. A frequent problem that online shops such as Bataleon are often faced with, is a mismatch between a product and customers' expectations of the product. Symptoms of that can present themselves in many different ways, such as a bad review or a return. In order to reduce Negative experiences of users, a system designed to help users find their best match is needed. In its current situation users are faced with a selection of choices and specifications of the Boards on a product page. Flexibility, Float level, Shape, Camber, Terrain predispositions and sizes all have different relevance for different users. Beginning Snowboarders might have no idea how Flexibility would impact the feeling of the board, but for more advanced ones, this parameter might be crucial in their choice of board. To accommodate both users a product suggestion tool should provide sufficient help for both types of users by showing more advanced options only to users who select that they consider themselves more advanced. In the desired situation LPS, has received a useful product, which helps their users with picking a snowboard that will best suit their riding style, environment and needs. The goal is an improvement in the relevance and clarity of product information presented to customers on our websites using product data points stored in shopify, resulting in better customer satisfaction and fewer returns.

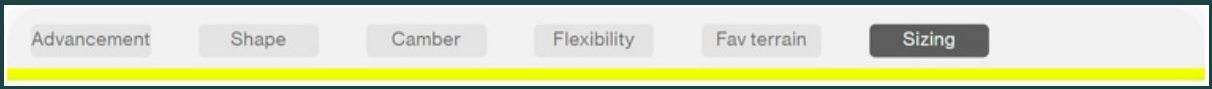
PROPOSED SOLUTION

Question Structure

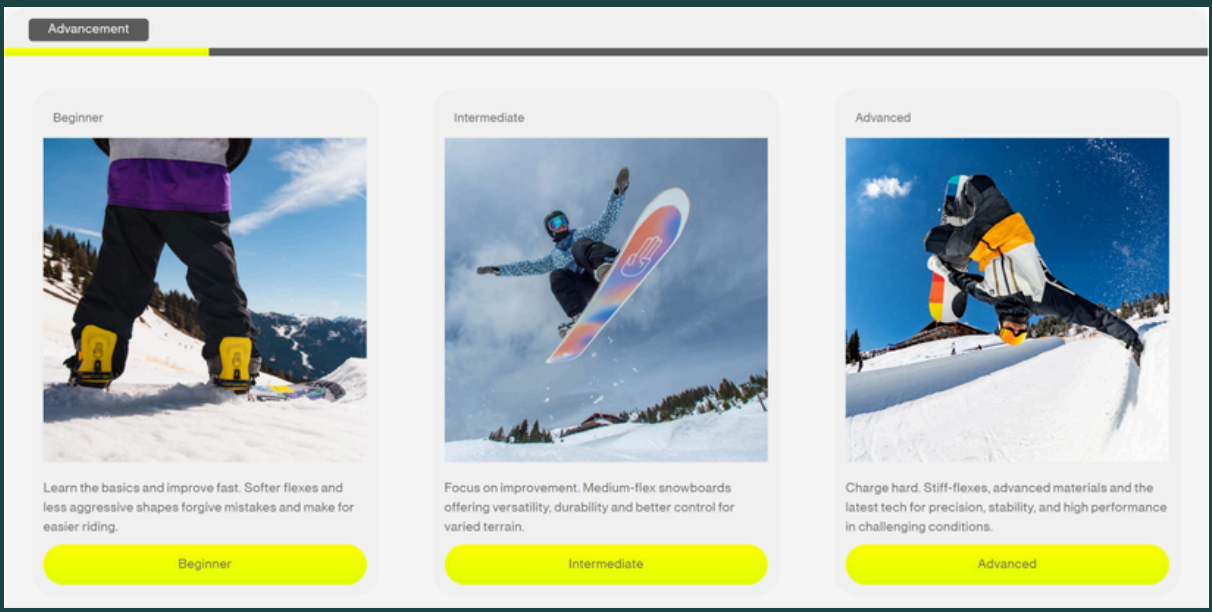
Proposed solution is a question based system which asks users from 3 to 6 questions, depending on how advanced they are. Advantages of this system are described in Competitive analysis & Best, good and bad practices report. You can find a chart with the whole question flow at Appendix A. The reasoning behind it is to provide users with more attention that is more adequate to their needs. Beginner users are going to be asked only about terrain that they intend to ride on, and about their boot size and weight. This information is sufficient enough to find a board that will fulfill users needs, and at the same time is easily understandable by virtually every person that might encounter the Bataleon website. Intermediate and Advanced users are going to be asked more sophisticated questions that will help them to pick a snowboard that will better suit their particular needs.

Intermediate users will receive 2 additional questions, one about flexibility, and one about camber. Those 2 parameters hold the most value for more advanced users, since they are primarily responsible for snowboards behavior on snow. Lastly Advanced users are going to be asked all 6 questions, which besides flexibility and camber also includes a question about the shape of the board, since this question is aimed at users who already have 1 or more boards. This solution will allow advanced users to receive a match that will best suit their needs and simultaneously will allow less advanced users to still receive a satisfactory match without being asked questions that wouldn't tell them much anyway.

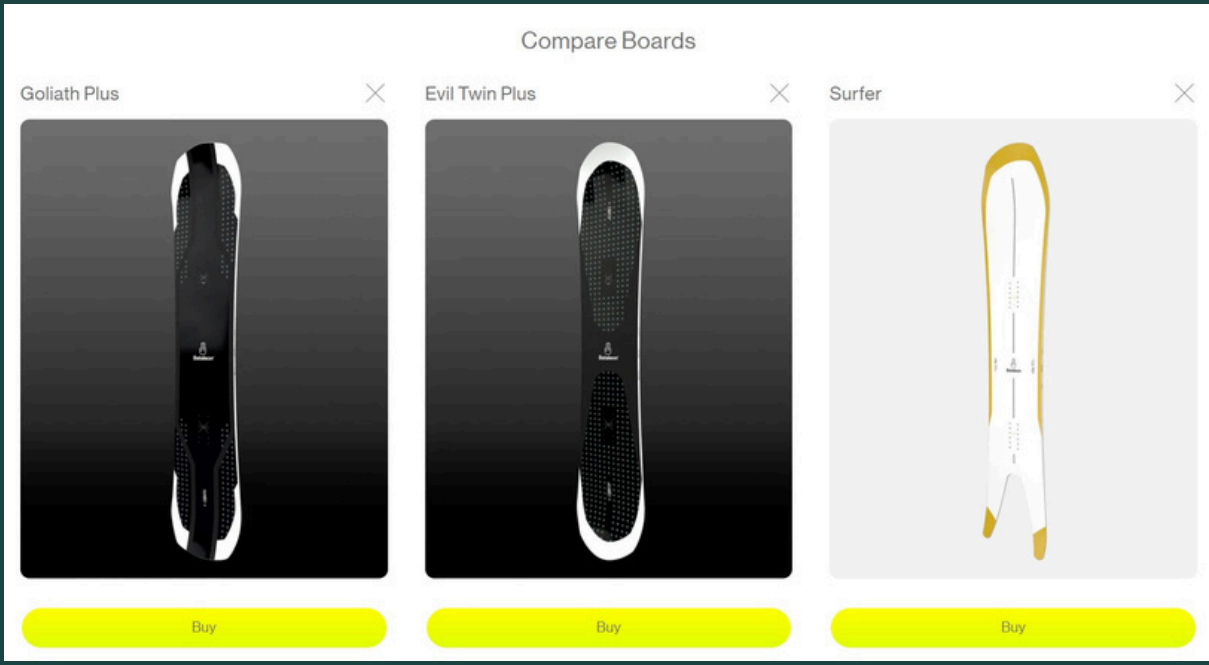
User interface



UI wise, product suggestion tool consist of a few elements. Firstly on top of the product suggestion tool, there is a header. It consists of 2 practical elements, navigational buttons, and progress bar . Progress bars' task is to inform users on how far into the questionnaire they are, and indicate how soon they'll see the results. Navigational buttons task is to allow users to seamlessly change from one question to another, should they want to make change to previously answered questions. Every part of the header from border-radius value and font to background colors and gradient color of progress bar, was styled in accordance with Bataleon style guidance, in order to make sure that product suggestion tool fits bataleon website as well as possible. Unfortunately I'm not allowed to share the style guide itself due to the company policy, but the Bataleon website is a good reference on its own. A link to it you can find here, or at the Appendix B.



Main element of the finder that is responsible for gathering users' answers are card buttons. They consist of the name of the answer at the top left, image relevant to the answer and a description in the middle which is describing what said answer is representing. Additionally at the bottom of each card button, you can see an additional green button. Despite a whole card being a button on its own, I've added the additional green "pill" button in order to highlight that it's clickable, as well as to better fit into the Bataleon website. I wanted to make the buttons and layout to feel more similar to other elements on the website, you can see an example of a "comparison" window on Bataleon website to see what I mean. You can also visit the prototype report to see how the prototype was iterated.



Code

Codewise this solution consists of several elements in a few different coding languages, which sum up to over 1000 lines of code. The languages used are JavaScript, which is used for overall functionality of a finder, such as switching between questions, displaying appropriate navigational buttons and providing users with a snowboard sufficient for their needs, HTML and CSS which are used for making website itself and styling it, and .liquid which is a coding language used by shopify which is used to calling snowboards from the website. Firstly the visual site is operated on HTML and CSS. Going from the top, navigational buttons are assigned JavaScript functions that are responsible for displaying questions that each of the buttons is representing. At the beginning however, only the first button that is responsible for "Advancement level" is displayed, since it's the first question that is being asked. This practice is in place in order to display only the navigational buttons that are meant for a particular target group, because navigational buttons represent questions that are being asked, and displaying questions that are not asked to the user would be unnecessarily confusing. Additionally the button that is showing the question that is currently displayed is always marked by having a much darker shade of gray added as a background color in order to indicate which question is currently being asked. For answering questions the user needs to click on card buttons which are responsible for

storing answers and changing which question is displayed. The values that correspond to each of the answers are stored in data-value. The first question however has an additional function to it. Because we need to incorporate different questions for different users, an additional value is needed. After Selecting the advancement level, one of 3 functions is activated. Those functions have a “chosen_path” value in them, and depending on the given answer, this value will be either 0 (beginner), 1(intermediate) or 2 (advanced). Based on this value, other functions will change which navigational buttons are displayed. Question about shape is only asked to advanced users, therefore it doesn’t need any if statements and because if it it looks like this

```
function QuestionShape(){
document.getElementById("TemplateLvl").style.display="none";
document.getElementById("TemplateShape").style.display="block";
    document.getElementById("TemplateCamber").style.display="none";
    document.getElementById("TemplateFlex").style.display="none";
document.getElementById("TemplateTerrain").style.display="none";
document.getElementById("TemplateSizing").style.display="none";
console.log("Chosen_path should be at " + Chosen_path);
console.log("QuestionShape");}
```

In comparison, the last 2 questions about terrain and about sizing, are displayed to all users, but beginner or intermediate users shouldn’t see the options for shape of the board, therefore it needs to look like that:

```
function QuestionSizing(){
   if(Chosen_path===0){
      document.getElementById("RemoveIntermediate4").style.display="none";
      document.getElementById("RemoveBeginner3").style.display="none";
      document.getElementById("RemoveBeginner4").style.display="none";
      document.getElementById("TemplateLvl").style.display="none";
      document.getElementById("TemplateShape").style.display="none";
      document.getElementById("TemplateCamber").style.display="none";
      document.getElementById("TemplateFlex").style.display="none";
      document.getElementById("TemplateTerrain").style.display="none";
      document.getElementById("TemplateSizing").style.display="block";
      console.log("QuestionSizing"); } else if(Chosen_path===1){


          document.getElementById("RemoveIntermediate4").style.display="none";
          document.getElementById("RemoveBeginner3").style.display="block";
          document.getElementById("RemoveBeginner4").style.display="block";
          document.getElementById("TemplateLvl").style.display="none";
          document.getElementById("TemplateShape").style.display="none";
          document.getElementById("TemplateCamber").style.display="none";
          document.getElementById("TemplateFlex").style.display="none";
          document.getElementById("TemplateTerrain").style.display="none";
```

```
                document.getElementById("TemplateSizing").style.display="block";
                console.log("QuestionSizing"); }else{


            document.getElementById("RemoveIntermediate4").style.display="block";
                    document.getElementById("RemoveBeginner3").style.display="block";
                    document.getElementById("RemoveBeginner4").style.display="block";
                    document.getElementById("TemplateLvl").style.display="none";
                    document.getElementById("TemplateShape").style.display="none";
                    document.getElementById("TemplateCamber").style.display="none";
                    document.getElementById("TemplateFlex").style.display="none";
                    document.getElementById("TemplateTerrain").style.display="none";
                    document.getElementById("TemplateSizing").style.display="block";
                    console.log("QuestionSizing");}

                }
```

and includes 2 “if” statements, which are there to only display what is needed to be displayed. All of the questions are using the same template, just with different values for the questions, and a template looks like this :

```
<div class="finder-template" id="TemplateTerrain" style="display:none">
                    <div class="card-row">
                        <div class="Snowboard-finder-header"> <div class
                        ="header-button" style="width=10%;"
onclick="QuestionLvl();">Advancement </div>
                            <div id="RemoveIntermediate3" class ="header-button remove"
onclick="QuestionShape();">Shape</div>
                            <div id="RemoveBeginner1" class ="header-button remove"
onclick="QuestionCamber();">Camber</div>
                            <div id="RemoveBeginner2" class ="header-button remove"
onclick="QuestionFlex();">Flexibility</div>
                            <div class ="header-button" onclick="QuestionTerrain();"
style="background-color: var(--darkest-grey); color: var(--lightest-grey);">Fav terrain</div>

                            <div class
="header-button"onclick="QuestionSizing();"style="display:none">Sizing</div>
                        </div>
                    </div> <div class="card-row" style="justify-content:start; background-color:

var(--darkest-grey);">
                        <div class="progressbar5" > </div>
                    </div> <div class="card-row" style="margin-top:2%;margin-bottom:2%;">

                        <div class ="card-column">
                            <label class="checkbox"><div class="cardButton" name="style-button"
data-name="terrain" data-value ="Park" style="height:100%"onclick="QuestionSizing();"><input
type="checkbox" name="Terrain" style="opacity:0;" onclick=TerrainPark();>
```

```
                        <h4 style="margin-bottom:4%; margin-left:3%">Park</h4>
                        <img class="fluid" width="100%" height="100%"
src=https://cdn.shopify.com/s/files/1/0553/2100/2151/files/product-finder-park.jpg?v=1 716291990/>

                        <p style="margin-top: 5%;">"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. </p>
                        <div class ="card-button-green"><p style="margin-top:
3%;">Park</p></div>
                    </div></label> </div> <div class
                    ="card-column">

                      <label class="checkbox"><div class="cardButton" name="style-button"
data-name="terrain" data-value ="Groomed" style="height:100%"onclick="QuestionSizing();">
<input type="checkbox" name="Terrain" style="opacity:0;" onclick=TerrainGroomers();>

                        <h4 style="margin-bottom:4%; margin-left:3%">Groomed</h4>
                          <img class="fluid" width="100%" height="100%"
src=https://cdn.shopify.com/s/files/1/0553/2100/2151/files/product-finder-groomers.jpg ?
v=1716291990/>
                        <p style="margin-top: 5%;">"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. </p>
                        <div class ="card-button-green"><p style="margin-top:
3%;">Groomed</p></div>
                    </div></label> </div> <div class
                    ="card-column">

                      <label class="checkbox"><div class="cardButton" name="style-button"
data-name="terrain" data-value ="Powder" style="height:100%"onclick="QuestionSizing();">
<input type="checkbox" name="Terrain" style="opacity:0;" onclick=TerrainPowder();>

                        <h4 style="margin-bottom:4%; margin-left:3%">Powder</h4>
                          <img class="fluid" width="100%" height="100%"
src=https://cdn.shopify.com/s/files/1/0553/2100/2151/files/product-finder-pow.jpg?v=17 16291989/>

                        <p style="margin-top: 5%;">"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. </p>
                        <div class ="card-button-green"><p style="margin-top:
3%;">Powder</p></div>
                    </div></label>
                    </div>
                  </div>
              </div>
```

This particular one is from a question about terrain.

How it works

After gathering values from html, the Javascript part of the code does a few things (besides what I have already mentioned). Firstly it calls snowboards from shopify database using liquid language, and makes an array with all the necessary values. Thanks to this solution, product suggestion tool will remain relevant even when the snowboards in stock change.

```
const snowboards = [

        {% paginate collections['all-snowboards'].products by 59 %}
          {% for product in collections['all-snowboards'].products %}
            {% for variant in product.variants %}
                {
                    name: "{{product.title | append: "-" | append:
variant.option1}}",
                    level: {{product.metafields.global.level}}, shape: "
                    {{product.metafields.global.shape}}", camber: "
                    {{product.metafields.global.camber}}", flexibility:
                    {{product.metafields.global.flex-1}}, ratingPark:
                    {{product.metafields.global.rating-1}}, ratingGroomers:
                    {{product.metafields.global.rating-5}}, ratingPowder:
                    {{product.metafields.global.rating-4}}, bootSizeRange:

[Number({{variant.metafields.global.recommend-boot}})-3,Number({{variant.met
afields.global.recommend-boot}})+3],
                    weightRange:
[Number({{variant.metafields.global.recommend-weight}})-3,
Number({{variant.metafields.global.recommend-weight}})+3]
                },
            {% endfor %}
          {% endfor %}
        {% endpaginate %}
        ];
```

It assigns values using {{product.metafield.global.X}} to corresponding values of the snowboards array. Later function findProduct() is executed, which is probably the most important function in the program.

```
function findProduct() {
        const form = document.getElementById('product-finder-form');
        const level =
document.querySelector('.cardButton[data-name="level"].selected')?.getAttrib
ute('data-value');
        const shape =
document.querySelector('.cardButton[data-name="shape"].selected')?.getAttrib
ute('data-value');
```

```
        const camber =
document.querySelector('.cardButton[data-name="camber"].selected')?.getAttri
bute('data-value');
        const flexibility =
document.querySelector('.cardButton[data-name="flexibility"].selected')?.get
Attribute('data-value');
        const terrain =
document.querySelector('.cardButton[data-name="terrain"].selected')?.getAttr
ibute('data-value');
        const bootSize = document.getElementById('boot-size-value"]'); const
        weight = document.getElementById('weight-value"]');

        const filteredSnowboards = snowboards.filter(board =>
            board.level == level && board.shape == shape &&
            board.camber == camber && board.flexibility ==
            flexibility && board.terrain == terrain && bootSize >=
            board.bootSizeRange[0] && bootSize <=


board.bootSizeRange[1] &&
            weight >= board.weightRange[0] && weight <= board.weightRange[1]
        ); console.log(terrain + 'is saved terrain now HERE!!!');


        const bestMatches = filteredSnowboards.slice(0, 3);

        const resultList = document.getElementById('result-list');
        resultList.innerHTML = '';

        if (bestMatches.length === 0) {
            // Find the closest match let
            closestMatch = null; let
            highestScore = 0;

            snowboards.forEach(board => {
                let score = 0; if (board.level == level) score++; if (board.shape ==
                shape) score++; if (board.camber == camber) score++; if
                (board.flexibility == flexibility) score++; if (board.terrain ==
                terrain) score++; if (bootSize >= board.bootSizeRange[0] &&
                bootSize <=


board.bootSizeRange[1]) score++;
                if (weight >= board.weightRange[0] && weight <=
board.weightRange[1]) score++;

                if (score > highestScore) {
                    highestScore = score;
```

```
                    closestMatch = board;
                } });


                resultList.innerHTML = `<p>No exact matches found. Showing the
closest match:</p>`;

                if (closestMatch) {
                    const boardDiv = document.createElement('div');
                    boardDiv.classList.add('card'); boardDiv.innerHTML = `

                        <h3>${closestMatch.name}</h3> <p id ="target">Level:
                        ${closestMatch.level}</p> <p>Shape: ${closestMatch.shape}
                        </p> <p>Camber: ${closestMatch.camber}</p> <p>Flexibility:
                        ${closestMatch.flexibility}</p> <p>Terrain:
                        ${closestMatch.terrain}</p> <p>Boot Size Range:
                        ${closestMatch.bootSizeRange[0]} -

${closestMatch.bootSizeRange[1]}</p>
                        <p>Weight Range: ${closestMatch.weightRange[0]} -
${closestMatch.weightRange[1]}</p>
                    `; resultList.appendChild(boardDiv); console.log('There were no direct
                    matches but it should give match

regardless')
                }
            } else {
                bestMatches.forEach(board => {
                    const boardDiv = document.createElement('div');
                    boardDiv.classList.add('card'); boardDiv.innerHTML = `

                        <h3>${board.name}</h3>  <p>Level:  ${board.level}</p>
                        <p>Shape:  ${board.shape}</p>  <p>Camber:
                        ${board.camber}</p> <p>Flexibility: ${board.flexibility}
                        </p>  <p>Terrain:  ${board.terrain}</p>  <p>Boot  Size
                        Range: ${board.bootSizeRange[0]} -


${board.bootSizeRange[1]}</p>
                        <p>Weight Range: ${board.weightRange[0]} -
${board.weightRange[1]}</p>
                    `; resultList.appendChild(boardDiv); console.log('It
                    should be giving matches now')

                });
            }

            //document.getElementById('result').classList.remove('hidden');
```

```
}
```

Firstly it collects data-values from every picked button, and stores them in constant values, which is a great way of collecting user input. Later it compares those values with the values of each of the boards, and limits the number of matches to best 3. Limiting the number of answers is further described in Competitive Analysis and Best good and bad practices report. In a nutshell this process is there to ensure the user receives meaningful help, that will allow them to pick a product that best suits their needs. If there were no direct matches it will analyze which values were fulfilled by the available boards, and will give an increased score value for each of the parameters that board is fulfilling. Lastly it will take the boards that were best fitting and create a div for them at the bottom of the page.

DISCUSSION

The proposed Bataleon product suggestion tool can greatly increase consumer satisfaction and user experience. It is a question-based approach that is providing a simplified experience for the beginner users and more detailed options for those who are advanced. It can improve the present situation where information about products is primarily aimed at advanced users, who know how each of the board's parameters is altering its performance.

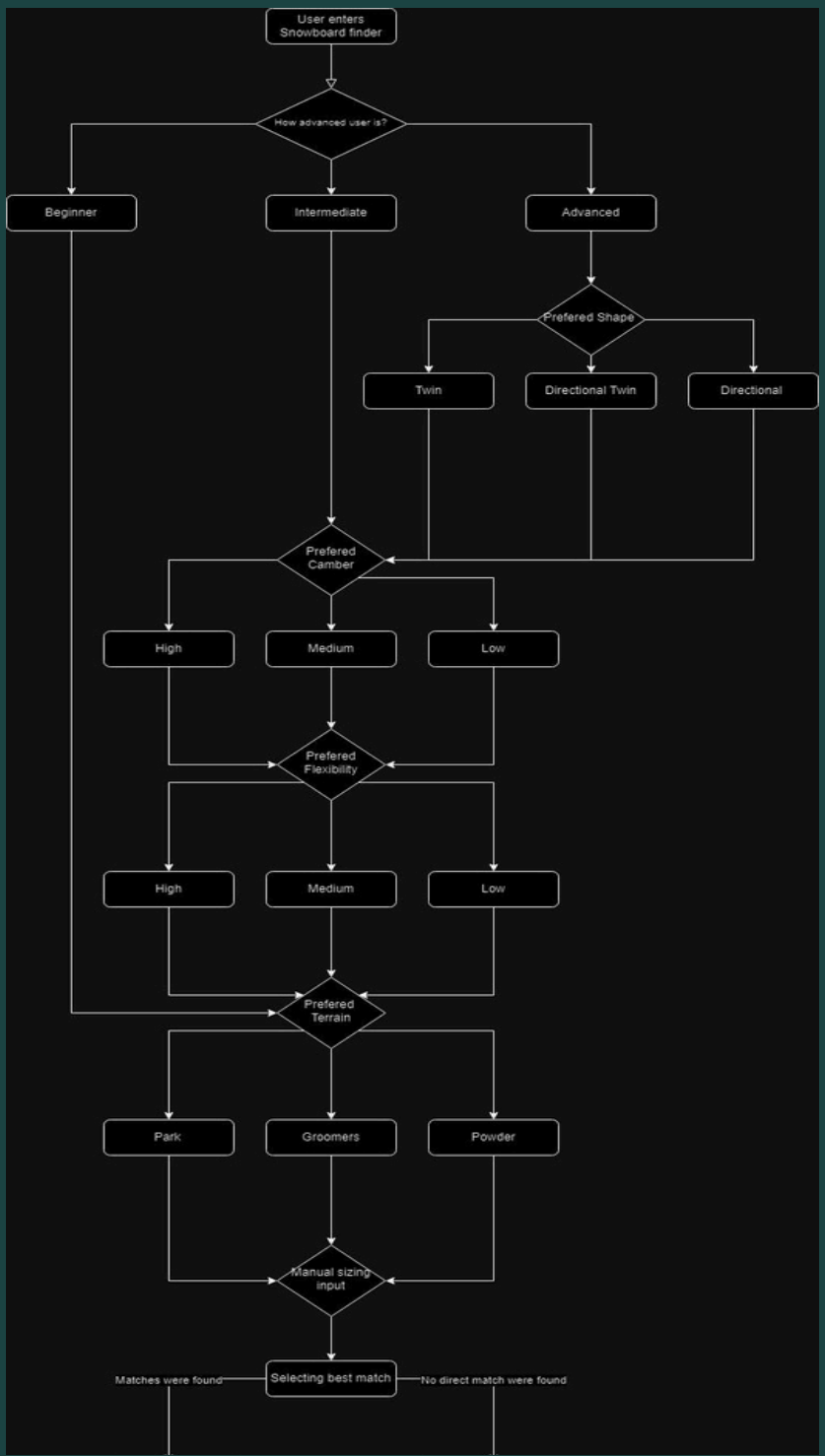
Through personalized interactions, both beginners and experienced users will benefit from this solution because it guarantees understandability and eliminates data overload. The system also assists with accurate matching between snowboard specifications and user demands by making use of the stored Shopify product data. This means that any time the user wants to select a snowboard, they will be satisfied with one that fits their taste as well as riding style based on this database's contents and user input. Lastly, since snowboards can be called from Shopify's database as needed at any moment throughout this exercise even when stock numbers fluctuate; it remains up-to-date with inventory movement.

A few things need improvement though. Mobile version does not exist yet for now but is to be implemented soon so that one can access everything through their mobile devices.

In terms of answering research questions, I believe that this document sufficiently describes how user information is extracted by using data-value features in HTML language(research question 5), and describes how stock is called by .liquid code from the shopify database (research question 6) using "{{product.metafields.global}}" methods.

Appendix
es

Appendix A



Appendix B

https://eu.bataleon.com/