**CSS**

Definindo tamanho da fonte de todo o corpo do documento

A browser's root font size is 16 px by default. When setting font-size to 62.5% in the CSS stylesheet the root font size changes to 10 px, because 16 \* 62.5 / 100 = 10. 1 REM is now equal to 10 pixels which makes converting values from Figma way easier. So 45 px is now 4.5 REM as opposed to 45 / 16 = 2.8125 REM.

html{

    font-size: 62.5%;

}

Tirando a margem default do navegador

body{

    margin: 0;

    font-family: 'Poppins';

}

Centralizando o conteúdo automaticamente nas laterais porem o topo ficará com margin-top 0

.wrapper{

    max-width: 1200px;

    margin: 0 auto;

}

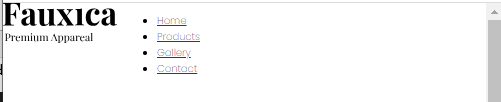
**Nav Bar**

Os itens do header (logo e conteúdo da nav) ficarão um ao lado do outro

header{

    display: flex;

}



Criando espaço entre os elementos

header{

    display: flex;

    justify-content: space-between;

}



Centralizando os itens do header (a, nav) verticalmente, criando uma margem no topo

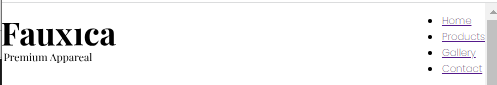
header{

    display: flex;

    justify-content: space-between;

    align-items: center;

}



Definindo margem da nav

nav{

    margin: 4em;

}

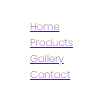


Tirando decoração dos itens da lista

nav ul{

    list-style-type: none;

}



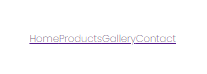
Em vez de 4 linhas, queremos 4 colunas

nav ul{

    list-style-type: none;

    display: flex;

}



Adicionando espaço entre os elementos (no figma o espaçamento é de 41)

nav ul{

    list-style-type: none;

    display: flex;

    gap: 4.1em;

}



Retirando espaço branco em volta dos itens da lista default do navegador (margem)

nav ul{

    list-style-type: none;

    display: flex;

    gap: 4.1em;

    margin: 0;

}

Tirando decoração default dos links

nav a{

    text-decoration: none;

}



Mudando a cor, transformando tudo em maiúsculo e tamanho da fonte (16 no figma). Para tamanho de fonte usar ‘rem’ e para outras coisas usar ‘em’ assim a medida de um não afeta a de outro.

nav a{

    text-decoration: none;

    color: black;

    text-transform: uppercase;

    font-size: 1.6rem;

}



Criando decoração dos links da nav quando ativos. A posição absoluta da decoração se baseia na posição relativa do elemento pai

nav a{

    text-decoration: none;

    color: black;

    text-transform: uppercase;

    font-size: 1.6rem;

    position: relative;

}

nav a.active:before{

    position: absolute;

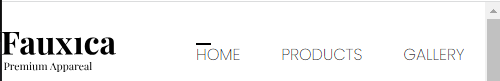
    content: "";

    width: 35%;

    height: 1px;

    border-bottom: 2px solid black;

}



Definindo a posição

bottom: -6px;

}



**Hero Section**

Dividindo o conteúdo principal em duas colunas

main{

    display: flex;

}



Adicionando espaçamento entre os dois elementos

main{

    display: flex;

    gap: 8em;

}



Espaço na parte de cima

    margin-top: 11em;

}

Como estes elementos são filhos do main que tem propriedade display:flex eles só ocupam o tamanho de seu conteúdo, é necessário fazer com que ocupem mais espaço

.left-col{

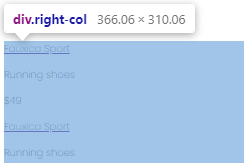
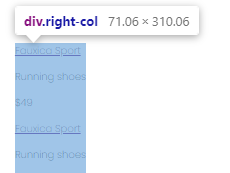
    width: 45%;

}

.right-col{

    flex-grow: 1;

}



Fonte com peso normal e outras configurações do h1

h1{

    font-size: 4.2rem;

    font-family: 'Playfair display';

    font-weight: normal;

    margin: 0;

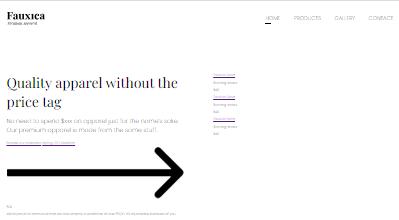
}

Configurações do parágrafo

.subhead{

    font-size: 1.8rem;

}



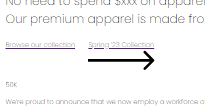
Div com os botões de chamada pra ação

.cta-btns{

    display: flex;

    gap: 2em;

}



Configurações do primeiro botão

.primary-cta{

    background-color: rgba(241, 226, 209, 0.82);

    font-size: 1.8rem;

    font-weight: bold;

    color: black;

    text-decoration: none;

    border-radius: 1.9em;

    padding: 1em 2em;

}



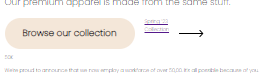
O segundo botão tem dois elementos, então precisará das mesmas configurações da div com as chamadas para ação

.cta-btns, .secondary-cta{

    display: flex;

    gap: 2em;

}



Configurações da segunda chamada para ação, padding 1em para top e bottom e 0 para right e left

.secondary-cta{

    font-size: 1.8rem;

    text-decoration: none;

    color: black;

    display: block;

    padding: 1em 0;

}



Consertando o botão ao mexer no elemento svg. ‘px’ para aumentar baseado no navegador ou preferencias de zoom do usuário

.secondary-cta svg {

    width: 20px;

}



Mais margem vertical

.cta-btns{

    margin: 5em 0;

}



Position relative por conta do confete que será adicionado

.news{

    display: flex;

    gap: 3em;

    padding: 5em;

    border: 1px solid black;

    border-radius: .8em;

    position: relative;

}



Adicionando o confete. Quando vc usa position-absolute pode usar z-index para garantir que as coisas estarão uma em cima da outra

.news:before{

    position: absolute;

    content: "";

    background: url('../images/confetti.svg');

    width: 100px;

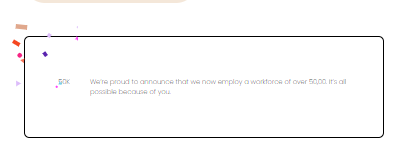
    height: 100px;

    top: -20px;

    left: -20px;

    z-index: 2;

}



Parágrafo 50K

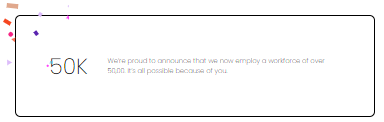
p.employees{

    font-size: 3.3em;

    margin: 0;

    font-weight: 200;

}



Tamanho e margem do outro parágrafo

p.details{

    font-size: 1.4rem;

    margin: 0;

}



Os dois elementos não estão alinhados no topo, então:

p.employees{

    font-size: 3.3em;

    margin: 0;

    font-weight: 200;

    line-height: 100%;

}



**Right Collumn Gallery**

Display: grid é usado para definir um contêiner como uma grade. Ao aplicar **display: grid;** a um elemento, você está criando um contexto de layout de grade. Isso permite que você organize os filhos desse elemento em linhas e colunas, facilitando o posicionamento e a organização do conteúdo de maneira mais flexível.



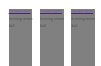
Se mudar para flex se torna colunas

.right-col{

    display: flex;

    gap: 2.3em;

}

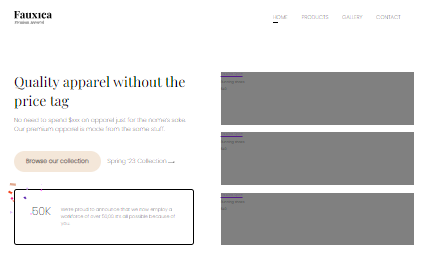
****

.right-col{

    display: grid;

    gap: 2.3em;

}

****

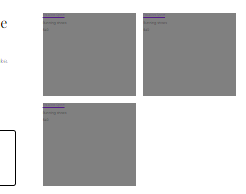
.right-col{

    display: grid;

    gap: 2.3em;

    grid-template-columns: repeat(2, auto);

}

****

Três palavras diferentes para três contêineres diferentes, visualmente nada muda ainda

.right-col{

    display: grid;

    gap: 2.3em;

    grid-template-columns: repeat(2, auto);

    grid-template-areas:

        "left right"

        "left bottom";

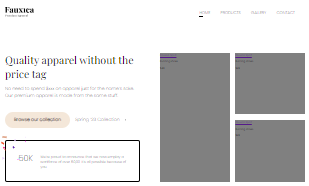
}

Agora vamos nomear cada container

.card1{

    grid-area: left;

}



Ok, não precisamos nomear os da direita pois já estão na posição correta

.right-col{

    display: grid;

    gap: 2.3em;

    grid-template-columns: repeat(2, auto);

    grid-template-areas:

        "left ."

        "left .";

}

Os elementos precisam ficar na parte de baixo, então vamos transformar o elemento pai em flex e alinhar tudo no final

.card{

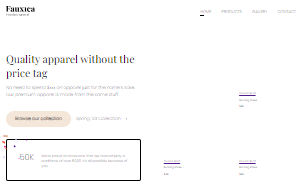
    border-radius: .8em;

    padding: 1.1em;

    display: flex;

    align-items: end;

}



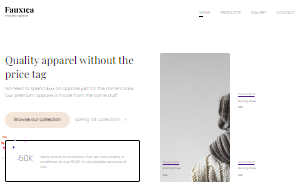
Adicionando a imagem do card1

.card1{

    grid-area: left;

    background-image: url('../images/clothing1.jpg');

}



.card{

    border-radius: .8em;

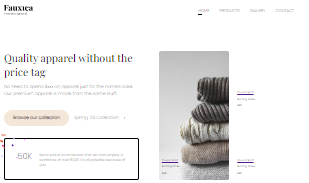
    padding: 1.1em;

    display: flex;

    align-items: end;

    background-size: 150%;

}



.card1{

    grid-area: left;

    background-image: url('../images/clothing1.jpg');

}

.card2{

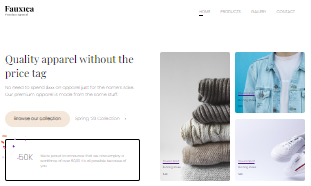
    background-image: url('../images/clothing2.jpg');

}

.card3{

    background-image: url('../images/clothing3.jpg');

}



Por algum motivo o justify-between não funcionou

.card-details{

    background: white;

    border-radius: .6em;

    padding: 2em;

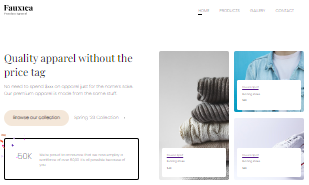
    width: 100%;

    display: flex;

    align-items: center;

    justify-content: space-between;

}



.card-details a{

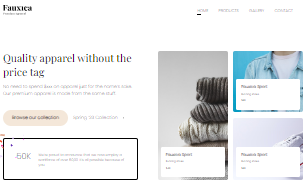
    color: black;

    text-decoration: none;

    font-size: 1.6rem;

    font-weight: bold;

}



Adicionando item esquecido. **pointer-events: none;** para um elemento, está indicando que o mouse não deve interagir com esse elemento, passando por ele como se o elemento não estivesse lá. Será importante para não afetar o link do container, caso contrário, o link fica unclicable.

.card1{

    grid-area: left;

    background-image: url('../images/clothing1.jpg');

    position: relative;

}

.card1:before{

    position: absolute;

    content: "";

    background: url('../images/tag.svg') no-repeat;

    width: 100%;

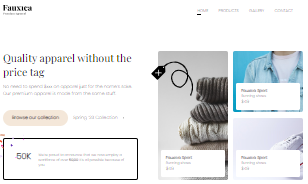
    height: 100%;

    top: 20px;

    left: -30px;

    pointer-events: none;

}



**Microinteractions**

Algumas alterações na nav com o objetivo de hovering

nav a{

    text-decoration: none;

    color: black;

    text-transform: uppercase;

    font-size: 1.6rem;

    position: relative;

}

nav a.active:before{

    position: absolute;

    content: "";

    width: 35%;

    height: 1px;

    border-bottom: 2px solid black;

    bottom: -6px;

}

nav ul li a:before{

    position: absolute;

    content: "";

    width: 0%;

    height: 1px;

    border-bottom: 2px solid rgb(119, 119, 119);

    bottom: -6px;

    transition: .3s;

}

nav ul li a:hover:before{

    width: 35%;

}



Hovering nos botões

.primary-cta:hover{

    background: #E2D1BD;

}

.secondary-cta:hover svg{

    transform: translateX(10px);

}

.secondary-cta svg {

    width: 20px;

    transition: transform .3s;

}



Hovering nos cards

.card{

    border-radius: .8em;

    padding: 1.1em;

    display: flex;

    align-items: end;

    background-size: 150%;

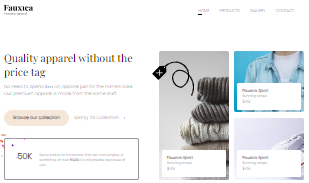
    transition: background-size 800ms;

}

.card:hover{

    background-size: 160%;

}



Enquanto o card estiver com hovering, os detalhes do cartão se moverão para cima

.card-details{

    background: white;

    border-radius: .6em;

    padding: 2em;

    width: 100%;

    display: flex;

    align-items: center;

    justify-content: space-between;

    transition: transform 800ms;

}

.card:hover .card-details{

    transform: translateY(-20px);

}



**Intro Animation**

Adicionando uma animação na página inteira

Adicionar uma div sem nada

<body>

    <div class="overlay"></div>

Reveal será o nome da animação

.overlay{

    position: absolute;

    z-index: 999;

    background: white;

    width: 100%;

    height: 100%;

    animation: reveal 1s ease-in forwards;

    transform-origin: bottom;

}

Especificar o keyframe para reveal

@keyframes reveal {

    from{

        transform: scaleY(1);

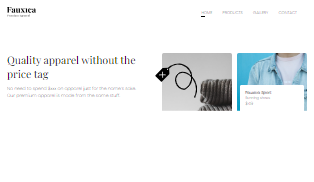
    }

    to{

        transform: scaleY(0);

    }

}



Adicionando animação no main, as coisas irão crescer

Deixar o main pequeno

main{

    animation: growIn 2s cubic-bezier(0.075, 0.82, 0.165, 1) forwards;

    transform: scale(.4);

}



Agora ele irá crescer

@keyframes growIn{

    to {

        transform: scale(1);

    }

}