INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Construtor de interfaces web

MÁRIO JOÃO GONÇALVES FERREIRO
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Construtor de Interfaces Web

Mário João Gonçalves Ferreiro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas de Informação e Conhecimentos

Orientador: Ricardo Almeida

Porto, outubro 2020

Dedicatória

Dedico este trabalho à minha mãe, pelos sacrifícios feitos de forma a poder dar-me um curso superior.

Resumo

O processo de desenvolvimento de interfaces é moroso e produz resultados intermédios desnecessários. No processo o *designer* produz imagens das interfaces, através de um programa de computador, que um programador terá de replicar em código. Um sistema que capaz de converter os ficheiros de *design* produzidos pelo *designer* para código pode reduzir o tempo de produção de interfaces. O objetivo desta tese é a construção de um sistema que faça essa conversão entre ficheiros de *design* e código aplicacional de forma automática. Tendo por base os padrões de *design* que surgiram na indústria e respetivas bibliotecas que os implementam, tanto a nível de *design* como de código aplicacional. O resultado do trabalho devolvido foi um sistema capaz de extrair um subconjunto reduzido de elementos da biblioteca de *design* e inseri-lo no código aplicacional. A organização de ficheiros no *design* também é replicada no código. Este trabalho é a primeira fase de um sistema conversor de *design* para código aplicacional. Durante a realização do mesmo foram encontrados diversos obstáculos que impediram a conclusão de outras fases da conversão.

Palavras-chave: Construtor de interfaces, *Design*, Geração de código, Orientações de *design*, Interface com o Utilizador

Abstract

The interface development process is slow and produces unnecessary intermediate results. In the process the designer produces images of the interfaces, by a computer program that a programmer will turn into code. A system capable of convert design files into code can reduce the time spent producing interfaces. The objective of this thesis is to build a system that can convert design files into code automatically. Based on design patterns that emerged in the industry and respective implementation libraries, both in design and code. The result of the developed work was a system capable of extract a small subset of elements from the design library and insert it in code files. The design file organization is replicated on code. This work is the first step of a converser system from design to code. During the development some problems appeared that made impossible the conclusion of other steps.
Keywords: Interface builder, Design, Code generation, Design guidelines, User Interfaces

Agradecimentos

Agradeço a todos os colegas que se mostraram disponíveis para a realizam das tarefas de inquérito. Em especial à Maria Meireles e Margarida Carvalho pela transmissão de conhecimento de domínio e fornecimento de material. E ao orientador Ricardo Almeida pela motivação, acompanhamento e sugestões.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

GUI	*Graphical User Interface*
IDE	*Integrated Develop Environment*
RA	*Realidade Aumentada*
UI	*User Interface*
UX	*User Experience*

1 Introdução

O presente capítulo destina-se a introduzir o problema tratado na tese, fornecendo o contexto e a descrição do problema em estudo. Também é apresentado os objetivos a cumprir e os motivos que originaram este trabalho. Por fim é indicada a estrutura do documento e como este deve ser lido.

1.1 Contexto

O desenvolvimento de interfaces é o ato de produzir um modo de interação do ser humano com a máquina, usualmente uma representação gráfica com a qual o utilizador pode interagir. O desenvolvimento de interfaces segue um processo onde pelo menos três entidades estão presentes: o cliente, o *designer* e o programador. O processo pode ser visto em maior detalhe na secção 2.1. Sendo que o processo envolve várias entidades, é necessário haver comunicação e troca de documentos entre as mesmas. Um tipo destes documentos são as maquetes desenvolvidas pelos *designers*. Estas maquetes são utilizadas pelos programadores para produzir a interface gráfica da aplicação. As maquetes, assim como a aplicação final, são produzidas num computador. Assim faria sentido que o *designer* pudesse ser capaz de criar a interface do produto final sem a necessidade de um programador, eliminando consequentemente a produção de maquetes.

O termo maquete pode ter significados diferentes em diferentes contextos. No contexto de arquitetura, mais vulgarmente usado, define-se como representação física à escala do edifício projetado(Wang *et al.*, 2019). No contexto de desenvolvimento de interfaces, a maquete pode ser vista como um conjunto de imagens que correspondem a uma página no sítio/aplicação web, e nas quais estão identificadas ações que quando realizadas nessa página a alteram total ou parcialmente (Bajammal, Mazinanian and Mesbah, 2018). No presente documento aplica-se a última definição.

1.2 Problema

O processo de desenvolvimento de interfaces passa por várias fazes podendo ficar num ciclo antes de passar para a próxima fase tornando-o moroso. Antes de chegar à fase de produção pode passar por várias fazes de idealização e refinamento. Em qualquer uma das fases pode acontecer alterações que adicionam complexidade tornando a conclusão do processo mais demorada.

Ao longo dos tempos têm sido desenvolvidas aplicações que facilitam o processo de desenvolvimento de maquetes por parte do *designer* (*Sketch*, no date; *Adobe XD Features*, no date), em algumas delas são adicionados módulos capazes de fazer a exportação de maquetes para código. Contudo o código gerado por estes é geralmente ineficiente, não estruturado e confuso. Estes problemas tornam praticamente impossível a sua utilização por um programador como base do projeto. Estas aplicações são pensadas de forma a ajudar o *designer* e não a simplificar o processo. A maioria dos designers não sabe produzir código, estes têm maior conhecimento estético não lógico.

Por outro lado, o código de geração de interface com o utilizador (UI) necessita de bastantes linhas de código, por isso foram desenvolvidas ferramentas que facilitam a produção de código de UI. Estas ferramentas podem ser bibliotecas que abstraem as primitivas de desenho (Sopin and Hamza-Lup, 2010), ou aplicações mais próximas do ambiente do *designer* (*OutSystems UI Framework*, no date; *SwiftUI*, no date). Estas ferramentas simplificam a etapa de implementação do design, mas não eliminam o *designer* do processo. Os programadores não têm o conhecimento estético para produzir interfaces apelativas.

Em suma, o processo de desenvolvimento nunca é totalmente modificado, apenas são simplificadas algumas etapas. O *designer* continua a produzir maquetes por ter mais conhecimento estético e artístico, e o programador continua a implementá-las por saber como ligar a UI a dados, e manipular estes últimos.

1.3 Objetivos

A solução proposta nesta tese passa por desenvolver um sistema que dê a capacidade aos *designers* de produzir interfaces sem necessidade de um programador. O principal objetivo é produzir interfaces em menor tempo, para isso pretende-se eliminar a necessidade de produção de maquetes. Passando logo para a fase de desenvolvimento de protótipo. Assim será desenvolvido um sistema que cumpra com os seguintes requisitos:

- Interpretar os *designs*
- Entender que elementos da interface podem ser traduzidos facilmente em código
- Identificar conceitos de *design*

- Fazer o mapeamento de conceitos *design* para código
- Criar ficheiros de código modelo para os conceitos *design*

1.4 Contributos e motivação

O tema desta tese surge pela experiência do autor como programador *front-end* na produção de UI. Grande parte dessa tarefa é monótona consistindo na replicação de imagens em código, contudo é possível ser automatizada devido a padrões que foram surgindo na indústria. Um programador preocupa-se maioritariamente com problemas lógicos, pelo que problemas estéticos são lhe mais difíceis de resolver. Um programador pode criar sistemas complexos e eficientes, mas não serem esteticamente apelativos. Por outro lado, um *designer* pode criar interfaces apelativas, mas não ser capaz de lhes atribuir funcionalidade. Assim surgiu a motivação para o desenvolvimento de sistema ajudasse a resolver os pontos fracos de cada um dos intervenientes no processo. Atribuindo uma especialização a cada interveniente, mas possibilitando uma interface autónoma de comunicação, o sistema a desenvolver.

1.5 Estrutura do documento

O documento está estruturado de forma a que possa ser lido seguido, em que o capítulo seguinte acrescenta informação sobre o anterior. Para certos conceitos ou documentos que se encontram em explicados em capítulos posteriores, existem referências para o local no documento em que estes se encontram. Possui sete capítulos seguido de referências e anexos.

O primeiro capítulo destina-se a apresentar uma introdução à área de negócio, ao problema e aos objetivos da solução.

O segundo capítulo destina-se a apresentar uma revisão de estado de arte de soluções existentes semelhantes ao objetivo, assim como descrever soluções que simplificaram etapas do processo de desenvolvimento UI.

O terceiro capítulo destina-se a apresentar a análise de valor e geração de ideias subjacentes ao problema a resolver.

O quarto capítulo destina-se a apresentar o *design* da proposta de solução, da arquitetura da solução e respetivos casos de uso, assim como a especificação dos testes a realizar.

O quinto capítulo destina-se de explicar o desenvolvimento da proposta e problemas existentes no mesmo.

O sexto capítulo destina-se a apresentar o modo de avaliação da proposta, a respetiva avaliação e os resultados obtidos.

O sétimo e último capítulo destina-se a apresentar as conclusões chegadas e apresentar trabalho futuro.

De seguida seguem-se as referências do presente documento e por fim os anexos, nestes últimos estão divididos entre material fornecido ao *designer* e ao programador.

2 Estado de Arte

No presente capítulo é descrito o processo de desenvolvimento de interfaces, e são apresentadas ferramentas que atualmente auxiliam os intervenientes do mesmo. Também são apresentadas uniformizações de criação de UI, os padrões de *design*. Essencialmente existem ferramentas que usam o resultado do *designer* para produzir código (aplicações de *design*), ferramentas que usam código para produzir *design* (programação orientada ao *design*), ferramentas dedicadas ao desenvolvimento de interfaces (construtores de interface), e ferramentas que pretendem integrar os envolvidos no processo na mesma plataforma (plataformas *low-code*). Estas ferramentas podem usar padrões de *design* para simplificar o desenvolvimento, quer das maquetes como do código UI.

2.1 Processo de desenvolvimento de interfaces

O processo de desenvolvimento de interfaces começa no pedido de um cliente para a construção de um sítio/aplicação web. Neste pedido são levantadas as funcionalidades e restrições em termos de apresentação que o cliente pretende. Caso seja necessário, é desenvolvida uma análise de mercado de forma a perceber que sítios/aplicações web existem atualmente, o que é que os seus concorrentes usam, de forma a obter ideias. Com a informação de negócio e as restrições do cliente o *designer* começa a criar uma maquete do sítio/aplicação web, geralmente do mais geral para o mais específico, por exemplo, primeiro o esquema de cores, segundo a idealização de algumas áreas com funcionalidades específicas, terceiro a ordenação destas últimas na página, entre outros. Ao longo do processo de desenvolvimento de interfaces são desenvolvidas várias maquetes até o cliente aceitar uma solução (Rodriguez-Martinez *et al.*, 2015).

Após a finalização da maquete esta é fornecida aos programadores informáticos. A função dos programadores é traduzir as imagens da maquete em código que o computador entenda, de forma a gerar uma cópia exata da maquete (Chen *et al.*, 2018). Este tipo de programador é conhecido como programador *front-end*, por serem responsáveis por desenvolver a parte que interage com o utilizador. Contudo nem sempre a maquete desenvolvida pelo *designer* pode ser facilmente transcrita em código, pelo que algumas alterações poderão ocorrer posteriormente (Leiva *et al.*, 2019). A **Erro! A origem da referência não foi encontrada.** descreve visualmente o processo e entidades envolvidas.

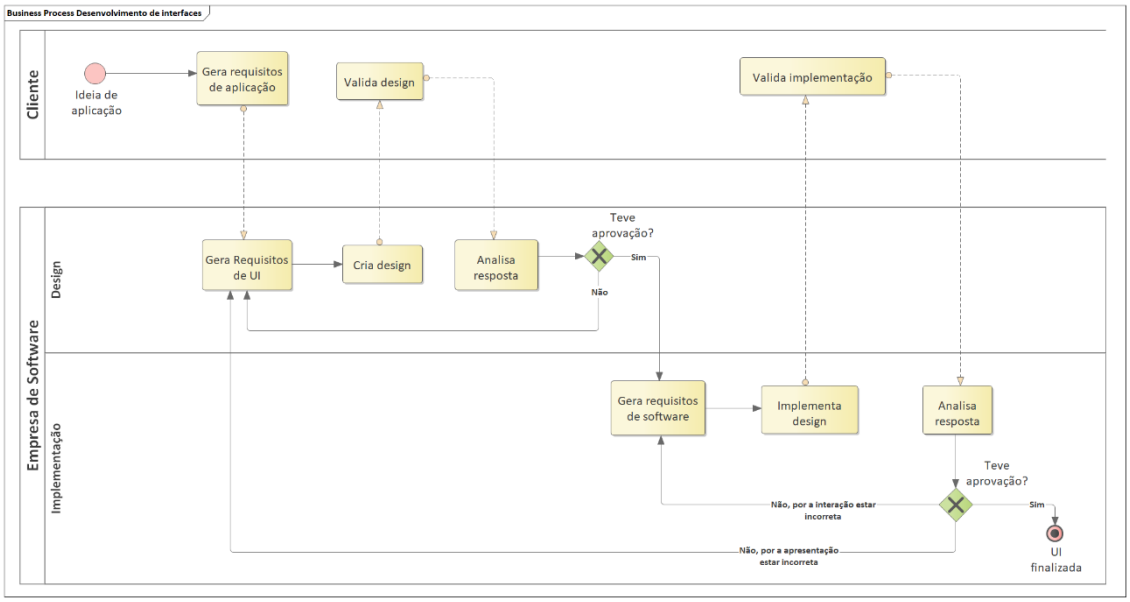


Figura 1 Processo de desenvolvimento de interfaces

Isto ocorre porque por serem especializações diferentes, a forma de pensar também é consideravelmente diferente. O *designer* tem um pensamento mais artístico, com base nas restrições do cliente ele vê uma folha a duas dimensões e desenha a interface. O programador tem um pensamento mais lógico, ao ver as imagens pensa em componentes semelhantes que pode reutilizar, numa estrutura hierárquica de componentes e de que forma é que o computador vai entender esse modelo. Em suma, o *designer* preocupa-se em tornar as ideias visíveis, enquanto o programador preocupa-se com tornar as ideias percetíveis para o computador. Esta componente mais artística dos designers pode fazer com que a mesma funcionalidade produza maquetes consideravelmente diferentes quando realizada por pessoas diferentes, tornando a experiência de utilização por parte do utilizador mais confusa.

2.2 Padrões de *design*

Inicialmente os *designers* só tinham à sua disposição um pequeno conjunto de componentes, campos de texto, pequenos ícones, botões, menus e pouco mais para construir as suas UI. Mesmo seguindo recomendações de estilo dos construtores de computadores os utilizadores tinham dificuldade em saber o significado dos elementos na interface. A existência de novas plataformas de interação também veio trazer novas formas de interação e por consequência interfaces diferentes (Tidwell, 2006). Enquanto num computador fixo o meio de interação é um rato e teclado, num dispositivo móvel o meio interação é o próprio ecrã, com o qual o utilizador interage através da via tátil. Num esforço para unificar o desenvolvimento de interfaces as grandes empresas de desenvolvimento de *software*, Apple, Google e Microsoft, criaram padrões de *design*. A Apple criou orientações de interface humano-máquina (*Human Interface*

Guidelines, no date) para cada um das suas plataformas, a Google o "Material *Design*" (*Material Design*, no date) e a Microsoft o "Fabric UI" (*UI Fabric*, no date), estes últimos padrões multiplataformas.

Os padrões de *design* não são mais que diretrizes que indicam ao *designer* como a interface deve ser desenhada. Indicam que componentes devem ser utilizados e seu respetivo formato, qual o seu posicionamento na página, como funciona a navegação da utilização da aplicação, as paletes de cores, as animações que devem realizadas na interação, entre outros. Servem para produzir maquetes mais facilmente, não para produzir código, contudo fornecem aspetos a ter em atenção na produção de código de interface com utilizador.

Um desses aspetos é mencionado no trabalho de Clemens Zeidler (Zeidler *et al.*, 2013), onde é abordado o problema de posicionamento de componentes num ecrã. Devido aos diferentes tamanhos de ecrãs dos dispositivos atuais, torna-se complicado desenvolver uma interface que se adapte às alterações de tamanho, que seja dinâmica. Para isso foram criados gestores de posicionamento, que através de regras próprias definem o tamanho e posição dos componentes em relação a outros elementos no ecrã.

2.3 Aplicações de *design*

De forma a poder construir uma maquete os designers têm atualmente uma vasta gama aplicações de *design* que os pode auxiliar. Aplicações como Sketch (*Sketch*, no date) e Adobe XD(*Adobe XD*, no date), das mais usadas na indústria, providenciam ao *designer* a capacidade de criação de maquetes com a possibilidade de exportar a solução para código, quer seja por suporte nativo da aplicação ou por plugins de terceiros.

Grande parte do trabalho nestas aplicações acontece em torno de uma área central onde são inseridos os componentes de *design*, e um conjunto de áreas de suporte que disponibilizam inserção de componentes, edição de propriedades ou ferramentas, como pode ser visto na **Erro! A origem da referência não foi encontrada.** Interface da aplicação Sketch.

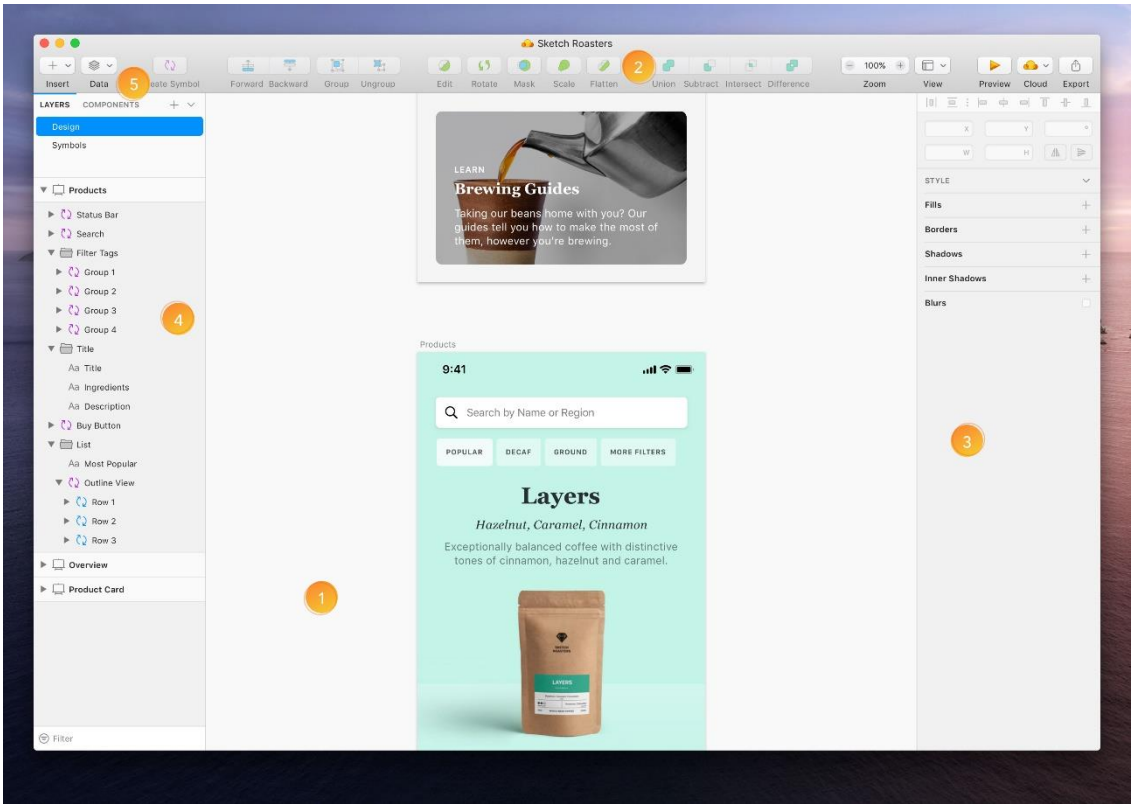


Figura 2 Interface da aplicação Sketch (*Sketch Interface*, no date)

Os números assinalados na **Erro! A origem da referência não foi encontrada.** indicam o seguinte:

1. Tela de desenho, onde são criados os diversos ecrãs de trabalho, sendo os componentes arrastados para estes ecrãs. É onde o resultado é produzido.

2. Barra de ferramentas, onde se encontram ações frequentes de edição de componentes, como escalar, redimensionar, alterar zoom, modos de seleção, criação de componentes, remover última alteração, entre outros.

3. Painel de propriedades, onde se encontram as propriedades do componente selecionado como alinhamento, tamanho, estilos, opções de redimensionamento, entre outros.

4. Painel de gestão, onde são apresentadas todas as entidades do projeto, como páginas e ecrãs de trabalho, e podem ser criadas camadas de forma a melhorar a organização dos documentos de trabalho.

5. Painel de componentes, onde se encontram os componentes de trabalho a ser usado pelo *designer*, quer seja já disponível pela própria plataforma ou criados pelo *designer*.

Grande parte da utilização nestas aplicações é realizada arrastando componentes das áreas de suporte para a tela de desenho ou os ecrãs de trabalho, e da edição de valores nas propriedades dos componentes. Também existem ferramentas que permitem desenhar as formas que compõem o desenho, mas são menos usadas pois as formas geométricas e componentes costumam ser suficientes.

As interfaces do Sketch e AdobeXD são bastantes semelhantes e possuem as mesmas funcionalidades. Nestes programas também é possível criar transições entre ecrãs, na medida em que ações num determinado componente de um ecrã de trabalho, alteram-no ou provocam uma transição para outro ecrã. Esta funcionalidade serve para dar maior interatividade ao protótipo, sendo mais fácil para o cliente perceber o resultado, pois não tem de analisar um conjunto de imagem. O cliente pode interagir diretamente com a maquete e dar com maior detalhe o seu parecer. Por outro lado o *designer* não tem de gerir uma enorme quantidade de ecrãs, um para cada situação possível (*Adobe XD Features*, no date). Estas aplicações também possuem kits de desenvolvimento que adicionam componentes e regras dos padrões de *design*. O Material Design possui kits de desenvolvimento para ambas (*Material Design design kits*, no date; *Design kits for AdobeXD*, no date).

O resultado proveniente deste tipo de aplicações não passa de um protótipo e não pode ser usado diretamente como um produto, sendo somente visível na aplicação de *design* ou como imagens individuais. É possível ver os ecrãs e componentes que definem a interface, as interações possíveis com os mesmos (no caso de ser visto na própria aplicação em que é desenvolvido), mas o produto final tem de ser desenvolvido por um programador de forma a tornar o protótipo numa aplicação funcional. É ainda necessário copiar toda a apresentação, adicionar a lógica de apresentação, quando cada componente deve estar visível e que transições entre ecrãs devem acontecer, assim como usar informação dinâmica proveniente de bases de dados.

Algumas ferramentas deste tipo conseguem extrair as maquetes para código funcional, o Sketch é uma delas. Essa funcionalidade é possível através do uso de um exportador para código HTML. Além do exportador por defeito do Sketch, existem entidades terceiras que desenvolveram o seu exportador. O Sketch2React (*Sketch2React*, no date) é um exemplo que usa etiquetas nos nomes dos elementos e um programa que usa essas etiquetas para facilitar a conversão em código. Ao contrário do Sketch, o AdobeXD delega a resolução deste problema para *plugins* desenvolvidos por terceiros. Em qualquer uma das situações o código não é eficiente, é demasiado complexo, extenso, não tem uma separação lógica de ficheiros, entre outros (*Sketch Code Export*, no date). Apesar de funcional não é legível para um ser humano, pelo que quando pequenas alterações são necessárias os programadores têm dificuldade em adicionar-lhe conteúdo pois não conseguem perceber o que o código executa.

2.4 Programação orientada a *design*

Os programadores são quem irá produzir a ferramenta final, por essa razão algumas empresas desenvolveram soluções centradas no trabalho desenvolvido por estes. A ferramenta SwiftUI (*SwiftUI*, no date) auxilia o programador a produzir interfaces. Como algumas partes do trabalho do programador são semelhantes e passíveis de automação, foram criadas abstrações parametrizáveis mais simples de usar que a escrita total do código de interface. Estas abstrações podem ser integradas numa ferramenta que permita visualizar a interface em tempo real. A visualização em tempo real, também permite interação, na medida em que é possível adicionar e editar os componentes visíveis nela. A interface desta ferramenta tem semelhanças com a interface das ferramentas de *design*, na medida em que também possui uma barra de ferramentas de componentes que o programador pode arrastar de forma a criar a UI. Assim o programador tem a possibilidade de interagir com a interface em tempo real adicionando-lhe componentes diretamente, ou através da escrita de código. Os modos de interação encontram-se sincronizados, na medida que uma alteração no primeiro fará uma alteração correspondente no segundo. Ou seja, se o programador adicionar o código para criar um botão este será visualizado na interface em tempo real, assim como se adicionar um componente à interface em tempo real o código para esse componente será inserido no código do ecrã correspondente (*WWDC 2019*, 2019).

A programação orientada ao *design* tem a vantagem de, por se tratar de programação, poder usar dados reais provenientes de bases de dados ou outros serviços e estes serem facilmente manipulados e integrados com a UI, ao contrário das aplicações de *design* onde toda a informação presente na maquete é fixa (Denuzière, Rodriguez and Granicz, 2013; *WWDC 2019*, 2019). Também tem a vantagem de se for necessário criar um componente específico para a interface, este pode ser facilmente desenvolvido e testado pelo programador, contudo não tem a mesma flexibilidade nem facilidade na criação de formas e desenhos que uma aplicação de *design*. Pois, enquanto a programação orientada ao *design* manipula funções que irão desenhar pixéis, as aplicações de *design* manipulam os pixéis diretamente.

Outra forma da programação orientada ao *design* são as bibliotecas GUI que tornam a criação de elementos de UI mais simples. Com o uso destas bibliotecas a produção da UI continua a ser responsabilidade do programador, contudo o esforço deste é consideravelmente reduzido. É o caso do trabalho desenvolvido por Ivan Soplin e Felix G. Hamza-Lup (Sopin and Hamza-Lup, 2010) no qual é desenvolvida uma biblioteca para a produção de GUI de elementos 3D. Nesta biblioteca deixam de ser utilizadas as primitivas para desenho Web3D e passam a ser usados componentes de GUI e de posicionamento, como campos de texto, painéis, botões, barras de tarefas, entre outros.

2.5 Construtores de interfaces

Os construtores de interfaces, são aplicações específicas para o desenvolvimento de interfaces. Tentam utilizar o melhor das aplicações de *design* e da programação orientada ao *design* para facilitar o processo de desenvolvimento de interfaces (Zarras *et al.*, 2018). O objetivo destas aplicações é permitir a uma só pessoa a capacidade de desenvolver a interface, esta pessoa

pode não ter nenhum conhecimento de *design* ou programação. Este tipo de aplicações não se cinge somente à área web, havendo aplicações na área de dispositivos móveis e realidade aumentada (RA) (Seichter, Looser and Billinghurst, 2008).

A ideia de dar a capacidade de construir interfaces a quem não tem conhecimento de como o fazer é parte integral dos trabalhos de Hartmut Seichter (Seichter, Looser and Billinghurst, 2008) e Kerry Shih-Ping Chang (Chang and Myers, 2014). No primeiro é dada a capacidade a quem não tem conhecimento de desenvolvimento de aplicações de RA de produzir as suas próprias experiências. Para isto são fornecidos um conjunto de modelos pré-definidos, aos quais podem ser adicionadas interações do estilo sensor-gatilho-ação. O sensor é um dispositivo com o qual o utilizador interage, por exemplo, um rato, um teclado. O gatilho são condições pelas quais serão desencadeadas ações, por exemplo um clique numa determinada área. Uma ação é algo que acontece à visualização, como por exemplo ser apresentado um marcador. Por exemplo ao passar com o rato numa determinada área de um modelo 3D, aparecer um marcador. Isto é possível porque existe um conjunto de gatilhos e ações pré-definidos através de scripts, contudo para alguém com conhecimento de programação poderiam ser criados novos scripts ou alterados os existentes. Por outro lado, no segundo é dada a capacidade a quem usualmente trabalha com folhas de cálculo de poder criar as suas aplicações. A aplicação funciona em três partes:

- Na primeira o utilizador especifica a fonte de dados em JSON, ficheiro ou API.
- Na segunda parte usa uma folha de cálculo e funções da mesma para extrair a informação da fonte de dados e fazer alguma alteração caso necessário.
- Na terceira usa um construtor de interfaces, com componentes pré-definidos, que através do arrasta para uma área de trabalho vai criando a interface.

Os componentes na interface podem por fim ter associados fórmulas que obtém os valores da folha de cálculo, também podem desencadear ações caso sejam botões, como por exemplo ordenação de listas. O resultado pode ser exportado gerando um URL onde a aplicação fica acessível.

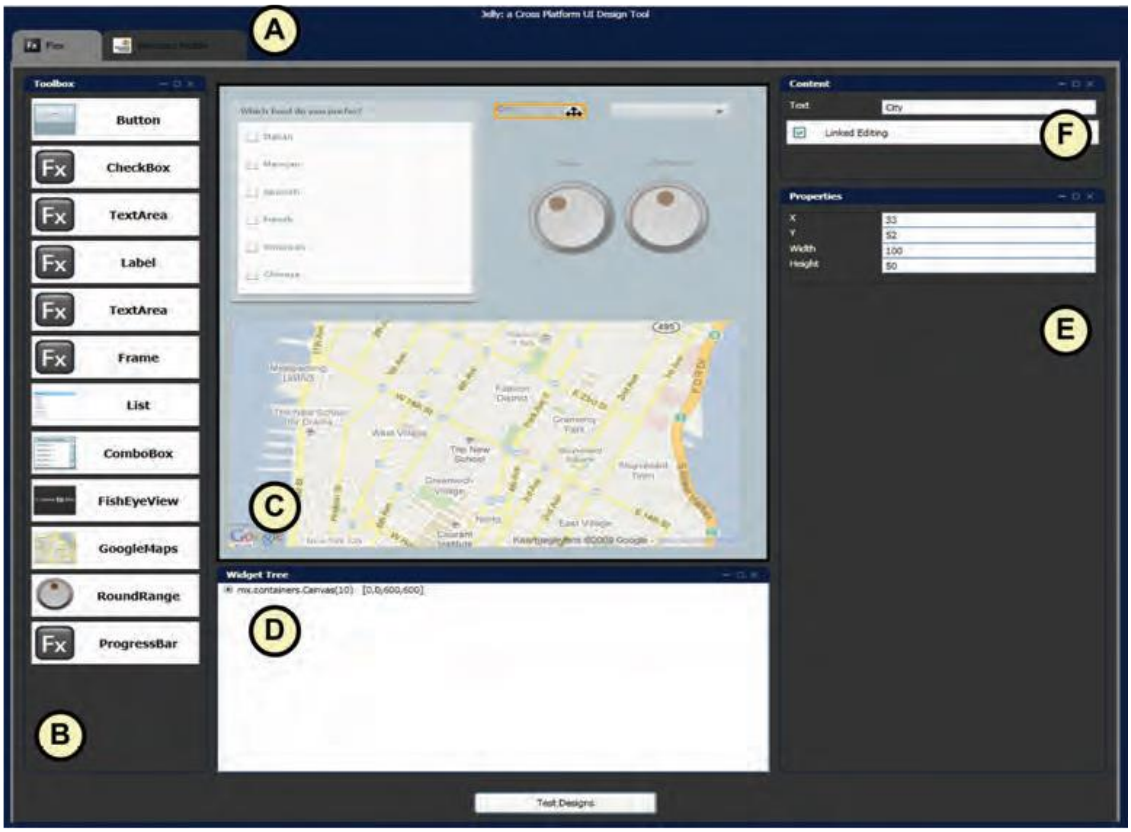


Figura 3 UI da aplicação Jelly

A utilidade dos construtores de interfaces também passa pela abstração do desenvolvimento para diferentes tamanhos de ecrãs e dispositivos, fixos ou móveis. Uma interface desenhada para ser usada num computador fixo, pode não ser exatamente igual a uma interface para ser usada num telemóvel, pois o tamanho do primeiro é consideravelmente maior em relação ao segundo, assim como no primeiro a utilização faz-se por meio de um rato, enquanto que no segundo com toques no próprio ecrã. A forma de desenvolvimento para diferentes dispositivos também pode ser consideravelmente diferente. Podem ter de ser usadas linguagens, *toolkits* ou bibliotecas diferentes. Enquanto que para desenvolvimento Android as linguagens usuais são Java ou Kotlin, para iOS são Objective-C ou Swift e para desktop podem ser Java, Python, C++, Javascript. Cada plataforma onde a interface funcionará terá as suas especificidades pelo que não pode ser usada uma linguagem para todas as plataformas. Uma solução passa pelo desenvolvimento para uma plataforma comum, os browsers, mas aqui o desempenho por parte dos dispositivos móveis é mais reduzido quando comparado com aplicações em linguagem nativa. O trabalho desenvolvido por Jan Meskens (Meskens, Luyten and Coninx, 2009, 2010) ajuda a desenvolver aplicações multiplataforma. O modo de funcionamento é semelhante a outros construtores de interface na medida em o ecrã principal também possui componentes pré-definidos (B), uma área de desenho (C), uma vista de árvore de componentes (D), um painel de propriedade (E). Contudo possui elementos próprios como uma área de conteúdo do componente (F) e abas para cada um dos dispositivos onde a interface será utilizada (A). Mais detalhes podem ser vistos na Figura 3.

Nesta aplicação não existe só um desenho de uma interface do qual é gerado o código para cada um dos dipositivos. Por haver especificidades na interação entre computadores e dispositivos móveis faz mais sentido haver uma interface para cada dispositivo onde é prevista a utilização. Assim fica mais simples tratar das especificidades de cada dispositivo. Contudo nada impede que partes da interface possam ser copiadas entre ambientes de desenvolvimento. Nesta aplicação não existe uma conversão direta entre o *design* e o código, o *design* é decomposto numa estrutura de dados mais simples e depois enviado para o motor de renderização de forma a gerar a UI. Isto pressupõe que haja comunicação entre o construtor de interface e o motor de renderização, no caso desta aplicação a comunicação é bidirecional. Isto serve para o construtor saber qual foi o elemento criado pelo motor de renderização de forma a poder criar uma ligação entre o elemento do GUI da interface no dispositivo. Assim qualquer alteração da parte do construtor de interface pode ser refletida no motor de renderização.

2.5.1 Limitações dos construtores de interfaces

Os construtores de interface aceleram bastante o desenvolvimento de aplicações, contudo não estão isentos de limitações. Por serem gerados de forma automática deixam de ter contexto. Enquanto que um programador pode dar um nome a um componente de código, os componentes usados nas GUI quando convertidos para código podem ser atribuídos números de forma a identificar componentes diferentes. Esta técnica vai tornar o código menos legível pois é muito mais fácil perceber o propósito de um componente por um nome descritivo que por um número. No mesmo sentido a inexistência e comentários no código torna a perceção do código mais complicada. Ignorar a criação de comentários no código não fornece aos programadores justificações para determinadas implementações. Outro problema são as propriedades atribuídas aos componentes. A forma mais simples de desenvolvimento é atribuir um fragmento de código a um componente, contudo pode haver componentes com responsabilidades semelhantes. Se componentes com responsabilidades semelhantes forem integrados num outro componente mais genérico poderá existir duplicação de código, aumentando a complexidade do mesmo. Outro erro comum diz respeito à interação. Os construtores podem ter a capacidade de realizar ações como abrir elementos na página, redimensionar elementos, entre outros. A maneira mais simples de dar esta capacidade seria ter o código de UI e de interação no mesmo componente, contudo esta solução dificulta a alteração do componente. Não havendo uma separação entre os dois tipos de código torna-se complicado para o construtor saber o que alterar no ficheiro de código (Zarras *et al.*, 2018).

2.6 Plataformas *low-code*

Outsystems pretende ser uma ferramenta capaz de cobrir todas as etapas de desenvolvimento de uma aplicação, não somente a construção da UI. Nela é possível gerir dados provenientes de bases de dados, adicionar lógica de negócio à base de ações, definir processos de negócio e criar interfaces. Estas funcionalidades têm ligações entre si de forma a poder-se desenvolver

uma aplicação funcional. Por exemplo associar ações a botões e utilizar manipulação de dados na resolução da ação. Para a construção da UI a Outsystems usa uma pilha tecnológica de quatro níveis, como mostra a Figura 4**Erro! A origem da referência não foi encontrada.**.

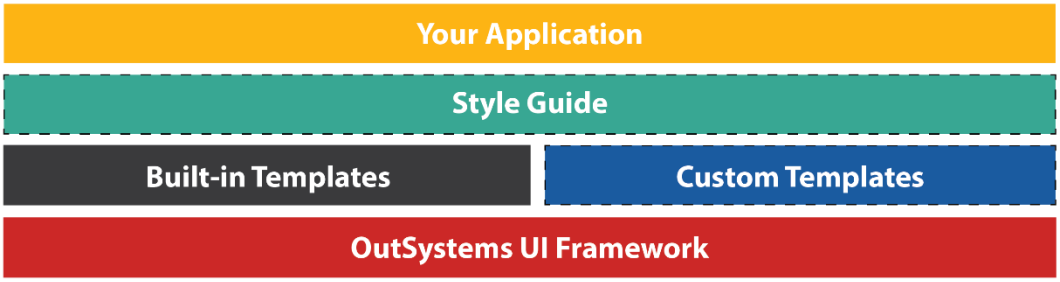


Figura 4 Pilha tecnológica de OutsystemsUI

Na base da pilha, "OutSystems UI Framework", encontra-se a base do desenvolvimento de UI, constituída por padrões de UI de disposição de conteúdo, apresentação do mesmo, interação, navegação e *widgets*. Na camada seguinte da pilha, "Built-in e Custom Templates", encontram-se modelos de ecrãs que apresentam os padrões da camada anterior numa forma estruturada. "Built-in" quando são providenciados pela ferramenta, "Custom" quando são criados pelo utilizador. Na camada seguinte, "Style Guide", é especificado o conjunto de cores e tamanhos dos *widgets* de forma a obter uma aparência consistente. E por fim na última camada, "Your Application" (A tua aplicação), existe acesso ao código CSS gerado nas camadas anteriores, caso seja necessária uma alteração muito particular.

O desenvolvimento da UI nesta ferramenta acontece de forma semelhante aos outros construtores de interface e aplicações de design, na medida em que o utilizador tem uma paleta de elementos que arrasta para um ecrã principal. A Figura 5**Erro! A origem da referência não foi encontrada.** apresenta a UI da OutSytems no desenvolvimento de interfaces.

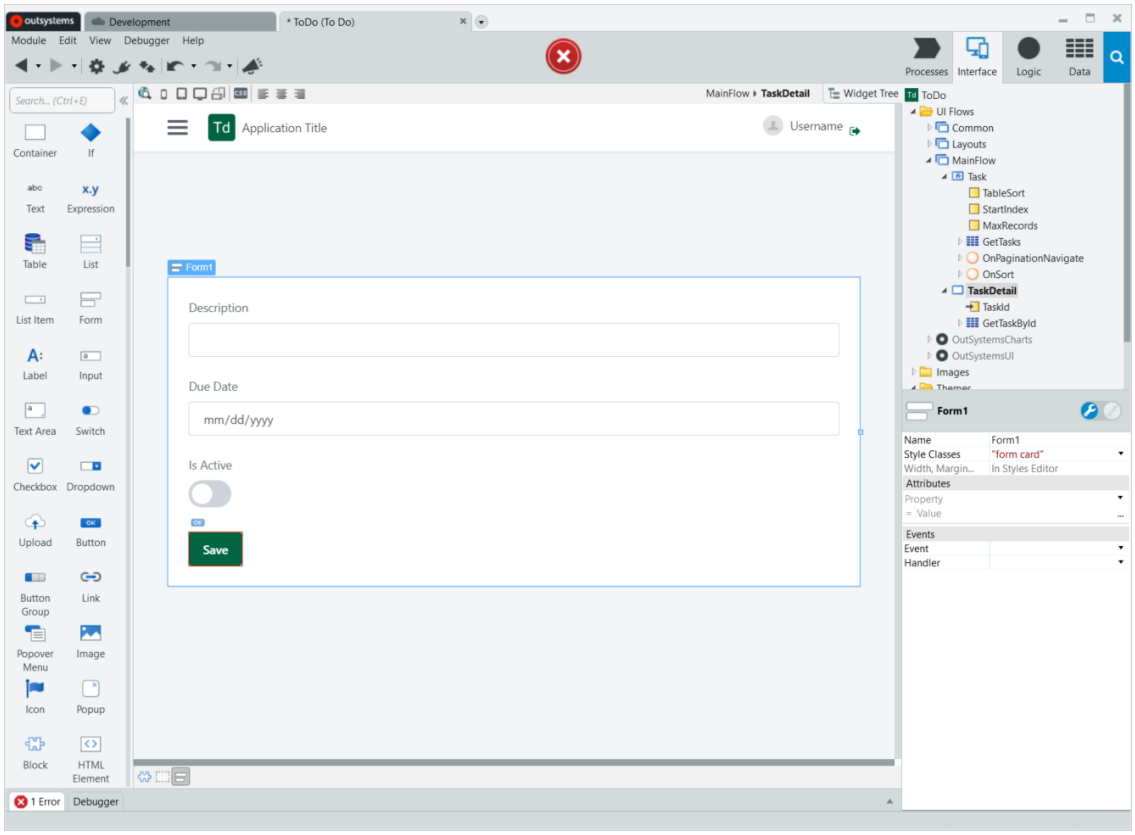


Figura 5 UI do Outsystems

3 Análise de valor

No presente capítulo encontra-se a análise de valor da proposta de solução, obtida através de conversas com *designers* de forma a adquirir conhecimento de domínio seguindo a metodologia *New Concept Development*. No mesmo capítulo encontra-se apresentação da proposta de valor seguindo do modelo Canvas. E a uma análise de relações entre conceitos do projeto seguindo a modelo FAST.

3.1 Identificação de oportunidade

O processo de desenvolvimento de interfaces, desde a conceção da ideia até à solução, é bastante moroso passando por várias fases de refinamento e por diferentes pessoas. Uma ferramenta que simplificasse esse processo traria maior produtividade aos envolvidos no processo. De acordo com o capítulo 2 já foram realizadas algumas tentativas de simplificar o processo, passando a responsabilidade para uma das entidades envolvidas. Contudo nenhuma delas conseguiu resolver a totalidade do problema, acabando por gerar outros conflitos. Como dependência de tecnologias especializadas e código gerado não utilizável.

Dessa forma faria sentido existir uma ferramenta capaz de simplificar o processo. Caso essa ferramenta existisse, os seus detentores estariam em vantagem face à concorrência. Os detentores da ferramenta poderiam criar maquetes funcionais em menor tempo e com maior facilidade, e os custos de desenvolvimento seriam reduzidos, pela eliminação de passos intermédios. Da mesma forma cada entidade do processo ficaria focada no seu trabalho, eliminando a necessidade de compreensão de outro domínio, e qualquer alteração poderia ser rapidamente integrada no resultado.

3.2 Análise de oportunidade

No capítulo 2 foram identificadas algumas formas de resolver o problema. As que se focam no trabalho do *designer* e tentam converter a sua criação para código utilizável. As que se focam no programador e tentam facilitar o processo de criação de interfaces por estas. E as que tentam ser mais abrangentes e tentam juntar o processo na mesma plataforma. Em qualquer uma das soluções o resultado não pode ser eficientemente utilizado por ambas as partes, quer por não poder ser mantido facilmente histórico de versões fora de ferramentas proprietárias, ou pelo código gerado não poder ser utilizado pelos programadores como o início do projeto, o esqueleto. Contudo as abordagens que se focam no trabalho do *designer* são competidores diretos pois só necessitavam de melhorar a conversão de *design* para código. Teriam maior facilidade em adaptar a sua solução para replicar as funcionalidades da solução.

Uma vez que os *designers* são responsáveis por traduzir requisitos de interface para uma maquete, faz sentido serem estes os principais utilizadores da solução. Assim esta última situa-se no setor dos designers. A mesma terá como diferença para os atuais competidores a capacidade de poder manter um histórico de versões e geração de código estruturado, pronto a ser usado pelos programadores como se de um projeto esqueleto se se tratasse. O surgimento de novas tecnologias e linguagens de programação e *frameworks* também traz maior facilidade no desenvolvimento da solução.

3.3 Geração da ideia e enriquecimento

Tratando-se de uma ideia nova, sem nada construído como base, o método de geração de ideia e de enriquecimento da mesma é composto por um grupo de no máximo cinco pessoas. Neste grupo uma delas é responsável pela ideia e as restantes são pessoas com especialidade no desenvolvimento de interfaces, de forma a poder dar a sua perspetiva na resolução do problema por parte da solução. As ideias geradas por este grupo foram as seguintes:

- Separar a geração da interface da geração do código. Na medida em que era fornecida uma ferramenta ao *designer* semelhante às atuais aplicações de design, que geram imagens. Contudo, neste caso, em vez de o resultado ser um conjunto de imagens seria um ficheiro com um formato descritivo dos elementos de UI utilizados. Este ficheiro seria posteriormente utilizado para gerar o código representativo da interface. Desta forma teria de ser desenvolvido um conversor da ferramenta de *design* para uma estrutura comum, e dessa estrutura para código. Assim como o inverso, de código para estrutura, e de estrutura para *design*. Esta estrutura poderia tirar proveito dos padrões de *design* para uma descrição mais simples da UI.

- Aproveitar o ambiente de desenvolvimento do programador (IDE) e criar uma abstração aos ficheiros de código, na medida em que através de um plug-in estes ficheiros são vistos como elementos de *design* e não como linhas de texto. Além da visualização também haveria ferramentas de apoio de forma a poder adicionar e manipular os elementos existentes. Neste caso tanto o programador como o *designer* trabalhavam sobre o mesmo ficheiro, mas em vistas diferentes.

- Usar uma linguagem funcional como base da aplicação, separando o estado da aplicação. Nas linguagens funcionais tudo é uma função pertencente a uma função principal, desta forma um componente pode ser uma função que recebe um conjunto de parâmetros e devolve a visualização do mesmo. Um componente também pode ser constituído por vários componentes, ou seja, uma função pode utilizar outras funções sendo que estas podem ser passadas por parâmetro. De forma à aplicação ser dinâmica esta precisa de possuir um estado mutável, de forma a alterar a sua apresentação conforme as alterações que sejam realizadas.

Manter o estado dentro dos próprios componentes torna mais complexo o desenvolvimento podendo misturar estado de negócio e estado de apresentação no mesmo lugar. Assim o estado de apresentação ficaria separado do estado de negócio, os designers alteravam uma parte do estado que não teria impacto no trabalho dos programadores. Sendo uma linguagem funcional os programadores recebiam a função base e todas as funções interiores como sendo um projeto esqueleto, não tendo de se preocupar com programar o *design* da aplicação.

3.4 Seleção de ideia

De forma a selecionar a ideia foi realizada uma entrevista com Pedro Freitas, engenheiro informático com três anos de experiência como programador e um ano como *designer*. Por ter conhecimento da área de *design* e da área de programação foi a pessoa que poderia produzir uma opinião mais consistente tendo em consideração as particularidades das duas áreas. As ideias indicadas na secção 3.3 foram-lhe apresentadas. A Tabela 1 apresenta as suas opiniões face às ideias.

Tabela 1 Opinião de ideias de solução

Ideia	**Opinião**	**Justificação**
Uso de programas de design auxiliados por uma EDUI e conversores	Positiva	Responsabilidades mantêm-se em separado Menor concorrência Possibilidade de versionamento
Uso de IDE com vistas diferentes	Bastante negativa	Alteração do mesmo ficheiro em paralelo Alta possibilidade de bloqueios e conflitos
Uso de linguagem funcional	Negativa	Resistência dos *designers* em usar outra ferramenta Facilmente copiado pela concorrência

3.5 Modelo Canvas

O modelo Canvas é uma forma de apresentar todo o negócio de uma empresa ou ideia de maneira bastante resumida, geralmente não ocupando mais que uma página A4. Na Figura 6 é apresentado o modelo Canvas da proposta de solução.

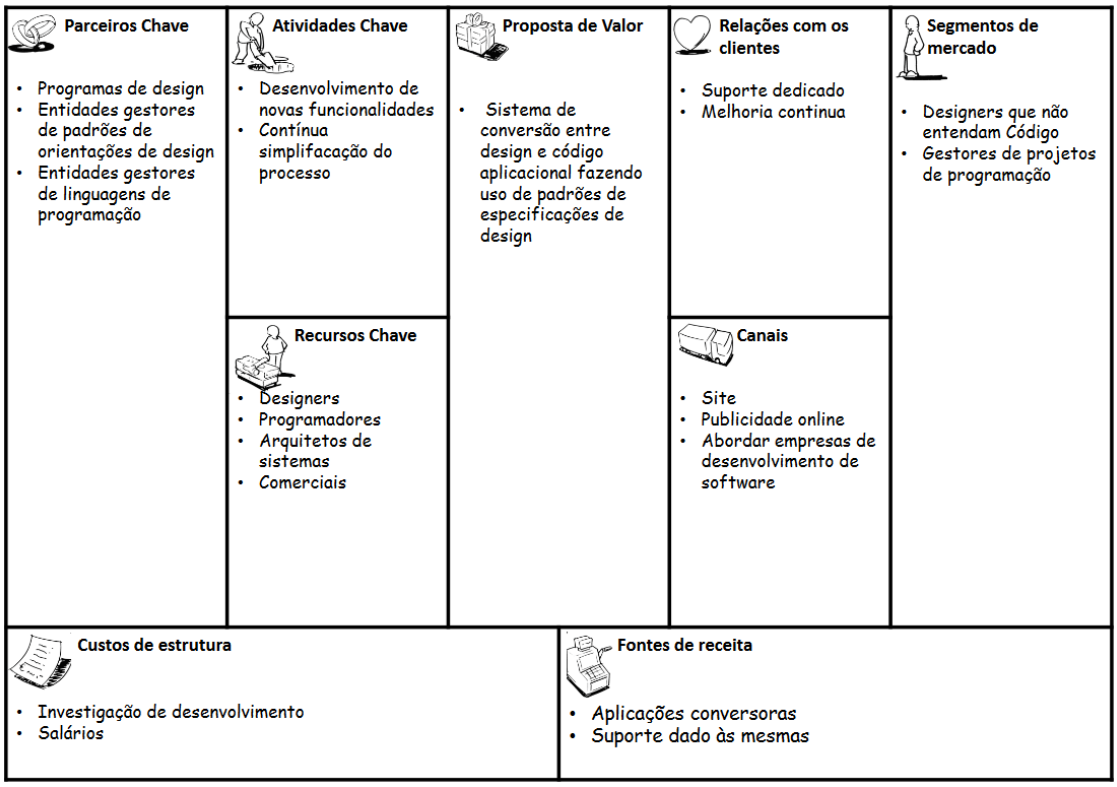


Figura 6 Modelo CANVAS

A proposta de valor do modelo é desenvolver um sistema de conversão de *design* em código, reduzindo o tempo de desenvolvimento de interfaces. Este sistema teria como principais clientes os designers independentes sem conhecimentos de código que quisessem mostrar um protótipo funcional e os gestores de projetos de programação que pretendessem reduzir o tempo de desenvolvimento dos seus programas. A forma de chegar a estes seria através de um site dedicado a explicar o objetivo do sistema e vantagens que este trazia, publicidade online, e abordar empresas de desenvolvimentos de software. Esta última requeria que um dos recursos chave fossem comerciais a vender o produto, mas *designer* e arquitetos de sistema também seriam necessários para desenvolver a solução. As tarefas a desempenhar seriam essencialmente desenvolvimento de novas funcionalidade e simplificação contínua do processo. Os parceiros chave seriam os programas de design, as entidades gestoras de padrões de design e entidades gestoras de linguagens de programação. Assim seria possível saber antecipadamente as novas funcionalidades que cada um destas implementaria e ajustar o sistema para as suportar. As fontes de receita dividem-se entre a venda do sistema na sua totalidade ou parcialmente através de aplicações conversoras e o suporte dado às mesmas,

incluindo eventuais formações de como usar a ferramenta. Mantendo uma relação com os clientes, de melhoria contínua e suporte dedicado.

3.6 Análise FAST

A análise FAST é uma modelo de análise de valor. Apresenta as funções que a solução deve suportar e como estas estão interligadas de forma a acrescentar valor. Por se focar nas funções, equipas e indivíduos e não estar sujeita a restrições físicas consegue apresentar claramente a definição do problema e o caminho para a solução (Borza, no date).

Esta análise consiste na apresentação de um diagrama, em que nele estão representadas as funções que o produto, processo ou sistema deve possuir. As funções são identificadas por caixas pretas, sendo que dentro delas encontram-se duas palavras, um verbo e um nome, que descrevem a função. As mesmas funções estão relacionadas numa forma como/porquê, sendo que a leitura da esquerda para a direita aprofunda o como, e a leitura inversa, da direita para a esquerda aprofunda o porquê.

O trajeto horizontal mais comprido denomina-se, caminho crítico, e são as funções principais da solução. Também existem áreas para tarefas que se podem aplicar em todo o caminho crítico, ou critério de design.

De acordo com a definição do modelo FAST, no contexto do problema desta tese, foram identificas as funções que se encontram representadas na Figura 7.

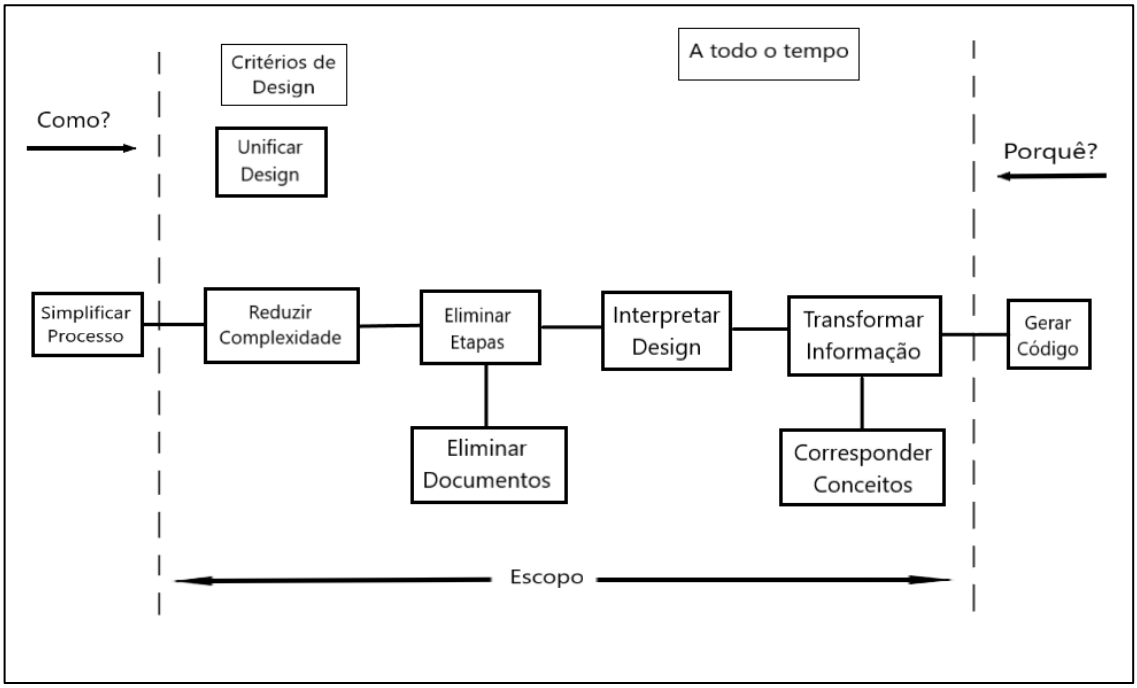


Figura 7 Diagrama FAST

A simplificação do processo é o objetivo geral desta tese e a geração de código mais específico. Ou seja, através da geração automática de código, pretende-se simplificar o processo. O valor da simplificação do processo encontra-se na redução de tempo entre a ideia e o produto final, através da redução da sua complexidade. A redução da complexidade tornará o processo mais simples de implementar. Esta é atingível pela eliminação de etapas e documentos. Estas funções no modelo fazem com que os utilizadores utilizem menos documentos para transmitir a mesma ideia. As funções de eliminação são possíveis pela interpretação do design criado pela aplicação de design. Interpretando o design conseguimos compreender os conceitos que lhe estão subjacentes. A interpretação do design é possível pela transformação da informação presente nos ficheiros do projeto da aplicação de design e pela associação entre conceitos de design e conceitos de código. Estas últimas funções em conjunto possibilitam a correspondência entre formas e padrões de design com excertos de código tornando possível o objetivo final desta tese - gerar código a partir de design. Subjacente ao presente parágrafo está a unificação do design pela utilização de padrões de design no desenvolvimento do design e do código.

4 *Design* da proposta

A realização da análise de valor trouxe a ideia de que um sistema que simplificasse o processo de criação de interfaces seria vantajoso para as entidades envolvidas no mesmo. O presente capítulo pretende demonstrar em que consiste esse sistema, quais as suas responsabilidades, a sua arquitetura e como este irá ser construído e testado.

4.1 Proposta

De acordo com o conhecimento adquirido no Estado de Arte, e tendo em conta o trabalho desenvolvido na Análise de valor, foi contruída a seguinte proposta:

De forma a solucionar o problema da extensão do processo de desenvolvimento de interfaces, e dos documentos excessivos gerados pelo mesmo pretende-se desenvolver um sistema que consiga converter o *design* em código utilizável. Neste caso, o *design* é realizado através da aplicação Sketch, a qual possui uma API (*SketchAPI*, no date) para interagir com os elementos criados pela aplicação. Além da API os elementos também estão acessíveis por manipulação direta do ficheiro Sketch. O código utilizável pretende-se que seja em React. A escolha destas tecnologias deve-se à disponibilidade de recursos humanos para desenvolver *design* e averiguar a qualidade do código gerado, contudo outras ferramentas poderiam ser utilizadas.

O sistema deverá fazer uso da uniformização trazida pelos padrões de design. Nomeadamente pelo uso de bibliotecas de componentes tanto para as aplicações de design, como para as linguagens de programação. O padrão escolhido para o desenvolvimento do sistema foi o Material Design, por ter bibliotecas disponíveis gratuitamente para o Sketch e para o React. Este padrão possui a biblioteca Sketch *material-design-stickersheet* (*Material Design design kits*, no date), desenvolvida pelos próprios criadores do padrão. E a biblioteca de componentes React Material-UI (*Material-UI*, no date), mantida por uma entidade diferente dos criadores do padrão.

O sistema terá de ser capaz de converter os conceitos de *design*, em conceitos de código, e o inverso. Tendo em conta as tecnologias e bibliotecas mencionadas no capítulo 2 e tendo em conta a análise de valor no capítulo 3, isto traduz-se em fazer corresponder componentes reutilizáveis de Sketch, conhecidos como *symbols*, em componentes reutilizáveis de código React. Além desta correspondência entre componentes qualquer opção de parametrização como tamanho, cor ou disposição também terá de se verificar.

Admite-se que sistema possa não fazer uma conversão perfeita, contudo terá de reduzir consideravelmente o trabalho desenvolvido pelo programador no desenvolvimento das interfaces. Este sistema terá de ser simples de usar, de forma a poder ser utilizado pelo próprio *designer*, que poderá não ter conhecimentos de programação. O sistema terá somente a

responsabilidade de converter o *design*, as interações com o mesmo, como mudanças de página, ou ações despoletadas por cliques, não são do intuito desta proposta e terão de ser desenvolvidas posteriormente por um programador.

O desenvolvimento do *design* Sketch tem de seguir um conjunto de regras. Cada *artboard* corresponde a uma página web, e cada página do Sketch corresponde a uma pasta na organização do código. Interações como mudanças de *artboard*, ou variações do mesmo *artboard* não deverão ser produzidas. Essas variações serão desenvolvidas pelo programador. Assim o programador e o *designer* terão a organização do seu trabalho estruturada e mais facilmente percebem os conceitos da outra área.

4.2 Objetivos

Do capítulo 4.1 foram extraídos os seguintes objetivos, que serão realizados através das seguintes tarefas:

- Simplificar o processo
    - Reduzir ficheiros utilizados
    - Separar responsabilidades
    - Simples de utilizar
    - Utilizar padrões de design
    - Desenvolvimento estruturado
- Converter *design* em código
    - Extração de conceitos dos ficheiros de *design*
    - Extração de conceitos dos ficheiros de código
    - Mapeamento de conceitos de *design* em conceitos de código
    - Mapeamento de propriedades de componentes
    - Conversão bidirecional, de *design* para código e de código para *design*

4.3 Tecnologias

Algumas tecnologias já foram mencionadas no capítulo 2, contudo uma melhor descrição das mesmas faz sentido pois algumas delas influenciam as decisões de *design*. Desta forma neste

capítulo primeiro são mencionadas as tecnologias escolhidas, seguidas de uma explicação mais detalhada das que mais influenciam as decisões de *design*. As tecnologias escolhidas para o desenvolvimento da proposta de solução foram as seguintes:

- **Sketch** como aplicação de desenvolvimento de UI, pela disponibilidade de pessoas que pudessem desenvolver *design*
- **ReactJs** por ser um *framework* JavaScript de construção de aplicações web bastante usado na indústria
- **Material *Design*** para simplificar a construção de UI, por seguir um padrão em vez de trabalhar com qualquer forma geométrica
- **Material-UI** por ser a implementação dos componentes do Material *Design* para ReactJs
- ***material-design-stickersheet*** por ser a biblioteca de Material *Design* para o Sketch desenvolvida pelos próprios criadores do padrão
- **JSON** por ser o formato dos ficheiros do arquivo Sketch e por ser facilmente importado/exportado a partir de JavaScript
- **JavaScript** para o desenvolvimento do plugin, de forma a manter a linguagem entre a API e do código final

Pelo facto de o Sketch ser uma tecnologia exclusiva para o sistema operativo macOS e pelo desenvolvimento ser feito em Windows, não foi possível utilizar a Sketch API. De qualquer forma a informação também se encontra disponível nos ficheiros do arquivo Sketch e esta está devidamente documentada (*Sketch File Format*, no date). A existência deste ficheiro e respetiva documentação não impediu que o Sketch fosse utilizado no desenvolvimento da solução. Contudo tornou impossível a sua utilização direta, sendo necessário recorrer a outras tecnologias para extraírem informação do ficheiro. Pelo ficheiro Sketch se encontrar no formato JSON foi decidido usar um ambiente de execução de JavaScript. Assim a tecnologia seguinte foi adicionada às anteriores:

- **NodeJS** para executar o sistema desenvolvido e aceder aos ficheiros Sketch e criar os ficheiros de código.

4.3.1 Sketch

O Sketch é uma aplicação usada pelos designers de forma a gerar maquetes de páginas *web*. Esta aplicação cria um projeto que tem por base um ficheiro de extensão *sketch*. Este ficheiro é um arquivo de ficheiros JSON e pastas. O designer ao utilizar o Sketch está a manipular este arquivo, adicionando-lhe novos ficheiros ou alterando os existentes.

De forma a poder criar uma maquete o designer organiza o seu trabalho em páginas e quadros de arte, conhecidos no Sketch por *artboards*. As páginas são grandes como grandes contentores que organizam os *artboards* onde são desenhadas as páginas web da maquete. As páginas web são desenhadas através da utilização de ferramentas de formas geométricas ou componentes, sendo estes últimos um conjunto estruturado das primeiras, pronto a ser reutilizado. Os componentes são conhecidos no Sketch por *symbols*. Os *symbols* não são somente agregados de formas geométricas, também podem ser compostos por outros *symbols*.

Os designers podem tirar partido da característica de reutilização dos *symbols* para criar bibliotecas Sketch. Estas bibliotecas são conjuntos de *symbols* que podem ser reutilizados em vários projetos, caso sejam usados como uma dependência. O uso das bibliotecas tem a vantagem de que qualquer alteração num *symbol* de uma biblioteca, provoque uma alteração no *symbol* correspondente em todos os projetos que a usem. As bibliotecas podem ser adicionadas localmente ao projeto, ficando inseridas no interior do mesmo, numa página de nome *symbols,* ou usadas como dependência externa ficando colocados os *symbols* utilizados na ficheiro principal do projeto, *document.json*.

Outros ficheiros estão disponíveis no ficheiro sketch corresponde ao projeto. A raiz deste ficheiro é composta por três ficheiros de nomes:

- **meta.json:** Agrega metadados sobre o projeto em si como páginas, quadros de arte, versão da aplicação, entre outros
- **document.json:** Agrega informações comuns a todo o documento, como objetos e estilos, provenientes de bibliotecas e localizações de páginas no sistema de ficheiros, entre outros
- **user.json:** Agrega informações pessoais do utilizador, como nível de zoom, tamanhos dos painéis da aplicação, entre outros

Além destes ficheiros também possui três pastas:

- **pages:** Onde se encontram localizados os ficheiros JSON correspondentes às páginas usadas no projeto
- **images:** Onde se encontram as imagens utilizadas no projeto
- **preview:** Onde se encontra uma imagem da última página editada pelo utilizador

Esta estrutura existe em qualquer ficheiro sketch incluindo as bibliotecas. A diferença destas para os projetos encontra-se nos *symbols* presentes e na ligação de dependência para outros projetos.

Para o contexto desta tese os ficheiros e pastas mais importantes são: o meta.json de forma a identificar as páginas e quadros de arte, o document,json de forma a identificar os elementos provenientes de bibliotecas, e a pasta *pages* e todos os ficheiros incluídos nesta.

Cada um destes ficheiros segue uma estrutura JSON pré-definida e cada objeto, e respetivas chaves têm um significado próprio.

O ficheiro meta.json possui uma chave do tipo objeto "pagesAndArtboards", em que cada chave desse objeto é o identificador da página. O valor dessa propriedade é por si outro objeto com uma chave “name” (nome) e outra “artboards”. O valor desta última é um objeto semelhante ao “pagesAndArtboards” na medida em que as chaves do “artboards” são identificadores dos *artboards* e os valores são objetos, mas nestes a única informação presente é o nome do *artboard*.

O ficheiro document.json possui uma chave "foreignSymbols" do tipo vetor, no qual se encontram vários objetos de symbols importados de bibliotecas que são usadas no projeto. Nestes objetos *symbols* existe uma chave "symbolMaster" cujo valor indica em detalhe o *symbol* importado da biblioteca. As chaves com maior significado do objeto “symbolMaster” são o “symbolId” por indicar o identificador daquele *symbol* na biblioteca, sendo o ID referenciado pelas instâncias que o implementem. E o “name” por identificar o tipo de componente, se é um cartão, um botão, um campo de texto, entre outros. O valor do “name” segue uma estrutura de palavras separadas pelo caracter barra “/”. Isto serve para organizar os symbols dentro da aplicação Sketch. Assim caso existam dois tipos de botão, redondo e retangular é possível ter ambos agrupados num menu “botão” e especificados em submenus, “redondo” e “retangular”. Outra chave importante do ficheiro document.json é a "pages”, do tipo contentor. Esta agrega os objetos das referências aos ficheiros das páginas presentes no projeto. Cada objeto de referência a página possui uma chave “_ref” cujo valor indica a localização onde a página se encontra relativamente à localização do ficheiro document.json.

Em cada um dos ficheiros relativo a uma página, encontra-se uma chave “layers” do tipo contentor que agrega todos os *artboards* presentes nessa página, assim como *symbols* ou formas não pertencentes a um *artboard*. Os *artboards* identificam-se por terem o valor "artboard" na chave "_class". Um *artboard* tem uma chave “name” indicando o nome do mesmo. Este valor poderá ser utilizado para dar nome ao ficheiro da página web, na organização de ficheiros no lado do código. O *artboard*, à semelhança da página, também possui uma chave “layers”, neste caso agrega os *symbols* e formas usadas especificamente naquele *artboard*. Esta semelhança faz notar uma estrutura hierárquica de contentores, i.e. uma página tem *artboards*, e estes têm o conteúdo. É sobre os *artboard* que os *designers* dedicam a maior parte do seu trabalho, por isso são de maior importância. Os *symbols* nas “layers” dos *artboards* são identificados por ter uma chave "_class" com o valor "symbolInstance". Neste objeto também existe uma chave “symbolID”, esta chave tem o mesmo valor que um “symbolMaster” presente no “foreignSymbols” do ficheiro document.json.

A relação hierárquica e respetivas cardinalidades podem ser vistas na Figura 8.

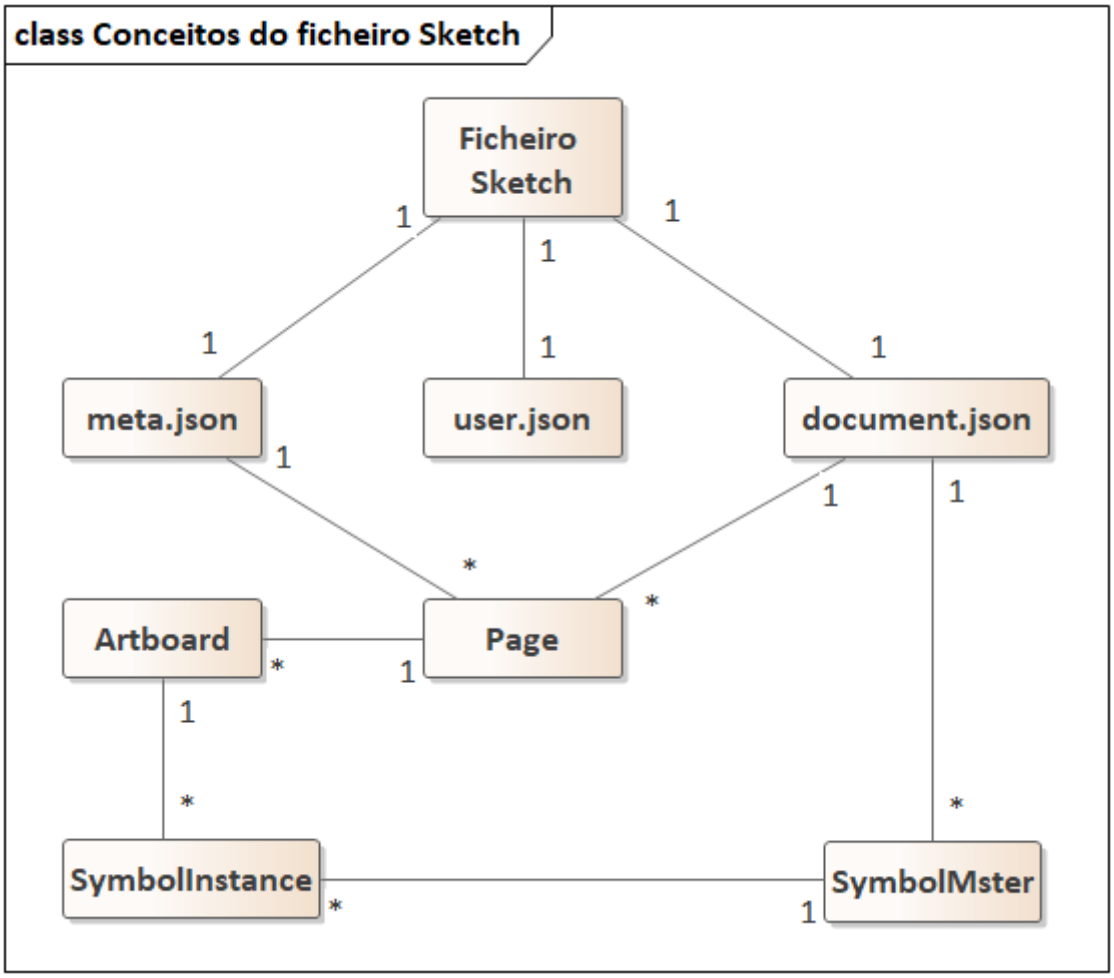


Figura 8 Arquitetura do ficheiro Sketch

4.3.2 ReactJS

O ReactJS, mais conhecido por React, de acordo com a informação disponível no respetivo sítio web, é uma biblioteca JavaScript para construir interfaces com o utilizador. Esta biblioteca está muito relacionada com o conceito de componente. Componente é uma parte da interface que pode ser reutilizado. No React os componentes seguem uma estrutura hierárquica onde um componente é constituído por outros componentes. Os componentes na base da hierarquia são somente constituídos por elementos. Um componente não precisa de ser constituído por outros componentes ou elementos, pode conter os dois. Os elementos são como etiquetas HTML, o resultado gerado pelo React. Apesar das semelhanças com HTML, os elementos escrevem-se numa sintaxe própria do React chamada JSX.

O JSX pode ser entendido como uma mistura entre HTML e JavaScript, na medida em que junta as etiquetas HTML com variáveis, condições e funções do JavaScript. Na Figura 9 é apresentado um fragmente de código JSX onde é possível ver a integração entre HTML e JavaScript numa só

declaração. O método *render* presente na mesma figura é o que permite que o resultado seja apresentado ao utilizador.

```jsx
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Figura 9 Fragmento de código JSX (*Introducing JSX – React*, no date)

Os componentes para poderem ser apresentados necessitam de um método *render* que retorne uma expressão JSX. Os componentes não são elementos estáticos que apenas apresentam o conteúdo do método *render*, podem sofrer alterações consoante o seu estado. O estado é um conjunto de variáveis internas ao componente que inicialmente tem um valor, mas que pode ser alterado através da função *setState*.

Apesar de os componentes poderem alterar o seu estado por vezes é preciso transferir informação de variáveis entre componentes. Atendendo à estrutura hierárquica, a informação é transmitida naturalmente de componente pai para componente filho. Existem formas de transferir informação no sentido inverso, mas são desaconselhadas. Pois rapidamente aumentam a complexidade da solução e fogem ao modelo unidirecional de transferência de informação. Restruturar componentes pode ser suficiente.

Ao React podem ser adicionadas bibliotecas de forma a adicionar componentes. Os componentes podem ser puramente visuais, parametrizáveis ou não, componentes com funcionalidades como ligações a serviços web, ou ambos os tipos de componente.

4.3.3 Material *Design* Stickersheet

A material *design* sticketsheet é a biblioteca Sketch do Material Design. Esta contém todos os componentes do padrão construídos como *symbols*. Por a biblioteca ser um ficheiro Sketch é possível consultar o seu conteúdo como se de um projeto se tratasse. Assim a biblioteca contém uma página guia para a explicar e páginas sobre o efeito de temas de cores, neste caso claro e escuro. Estas páginas implicam que haja outros *symbols*, específicos para as opções temáticas ou para o guia.

Os *symbols* encontram-se organizados em *artboards* genéricos. Estes *artboards* podem dizer respeito a componentes de seleção como botões, caixas de seleção, botões de radio; barras, como por exemplo, barra de aplicação, barra de navegação superior ou inferior; cartões, com várias opções, como com ou sem imagem, com texto, só título. A organização da biblioteca faz com o *symbol* de origem do componente seja mais difícil de descobrir, pois um botão escuro, não deixa de ser um botão, apenas apresenta uma propriedade côr já definida. Contudo as propriedades das instâncias dos *symbol* encontram-se definidas nas propriedades dos *symbolMaster*.

4.3.4 Material-UI

Material-UI é uma biblioteca que implementa as normas do Material Design em componentes React. Esta biblioteca não foi desenvolvida pelos criadores do Material Design, pelo que alguns componentes ou normas mais complexas podem estar em falta, especialmente quando as normas se contradizem. De qualquer forma a biblioteca fornece um conjunto vasto de componentes que simplificam o desenvolvimento de interfaces.

A biblioteca fornece um conjunto de componentes React que podem ser utilizados em conjunto com o código desenvolvido. Estes componentes pré-fabricados têm disponível uma API para os parametrizar e apresentar comportamentos ou apresentações diferentes. Esta API consiste num conjunto de *props* que o componente do Material-UI pode receber, assim como os respetivos valores possíveis. Por exemplo um botão possui *props* de cor, tamanho, variação e ativação. Estas propriedades podem, respetivamente, mudar a cor do botão, o tamanho pré-definido do mesmo, botão de tamanho pequeno, médio ou grande, ou o tipo do botão, se só apresenta texto, se tem contornos ou também tem relevo, e ainda se pode ser clicável.

Apesar de os componentes já terem um estilo pré-definido este não é somente personalizável por *props*, também pode ser alterado através de propriedades CSS. Os componentes poderem ser um agregado de outros componentes mais simples, também personalizáveis. A biblioteca Material-UI disponibiliza uma *prop* para alterar as propriedades CSS de cada um dos elementos que constituem o componente, ou alterar o componente em si consoante o seu estado. Por exemplo um campo de texto pode apresentar personalizações diferentes consoante o seu estado. Pode ter um contorno preto quando está ativo, uma tonalidade cinzenta quando não está acessível, um contorno azul quando está selecionado, um contorno vermelho quando o seu valor está incorreto. Além da personalização direta do componente a Material-UI também dispõe de personalização por temas. Os temas são conjuntos de propriedades CSS que se aplicam a todos os componentes. No contexto de uma página de uma marca, esta poderia ter no azul a sua cor de marca, dessa forma fazia sentido que a cor principal das páginas fosse o azul. Com os temas é possível definir que a cor principal e esta ser aplicada em todos os componentes, não sendo necessário personalizar individualmente os componentes utilizados. Opções como espaçamentos, cores secundárias, tamanhos de letra também estão disponíveis.

De forma a poder ser utilizado um componente Material-UI só é necessário importá-lo da biblioteca e adicioná-lo ao método *render* do componente React. Como pode ser visto na Figura 10.

```
import React from 'react';
import ReactDOM from 'react-dom';
import Button from '@material-ui/core/Button';

function App() {
  return (
    <Button variant="contained" color="primary">
      Hello World
    </Button>
  );
}

ReactDOM.render(<App />, document.querySelector('#app'));
```

Figura 10 Utilização da Material-UI (*Usage - Material-UI*, no date)

Durante o desenvolvimento desta tese os criadores do Material-UI criaram a sua biblioteca de símbolos Sketch (Tassinari, 2020), contudo a utilização desta biblioteca não é gratuita e por isso não foi utilizada.

4.4 Arquitetura

Através do conhecimento adquirido no capítulo 2 em conjunto com a informação técnica presente no capítulo 4.3 gerou-se o modelo de domínio presente na Figura 11. Nele é identificadas as entidades que constituem o ficheiro Sketch, assim como as constituem o código ReactJs. A relação entre essas entidades está identificada através de relações. Assim como a correspondência entre as entidades que constituem o ficheiro Sketch e as que constituem o código de aplicação ReactJs.

A arquitetura escolhida para o desenvolvimento da solução foi uma arquitetura em camadas. Uma camada de conversão entre código e *design* ou o inverso. E duas camadas de transformação de informação, uma para o código outra para o *design*. As camadas de transformação responsáveis pela extração de informação dos ficheiros, e pela inserção de informação convertida. A camada de conversão possui as regras pelas quais a conversão se deve efetuar. Ou seja, o mapeamento entre conceitos de código e conceitos de *design*. As camadas podem ser vista, na vista lógica presente na Figura 12.

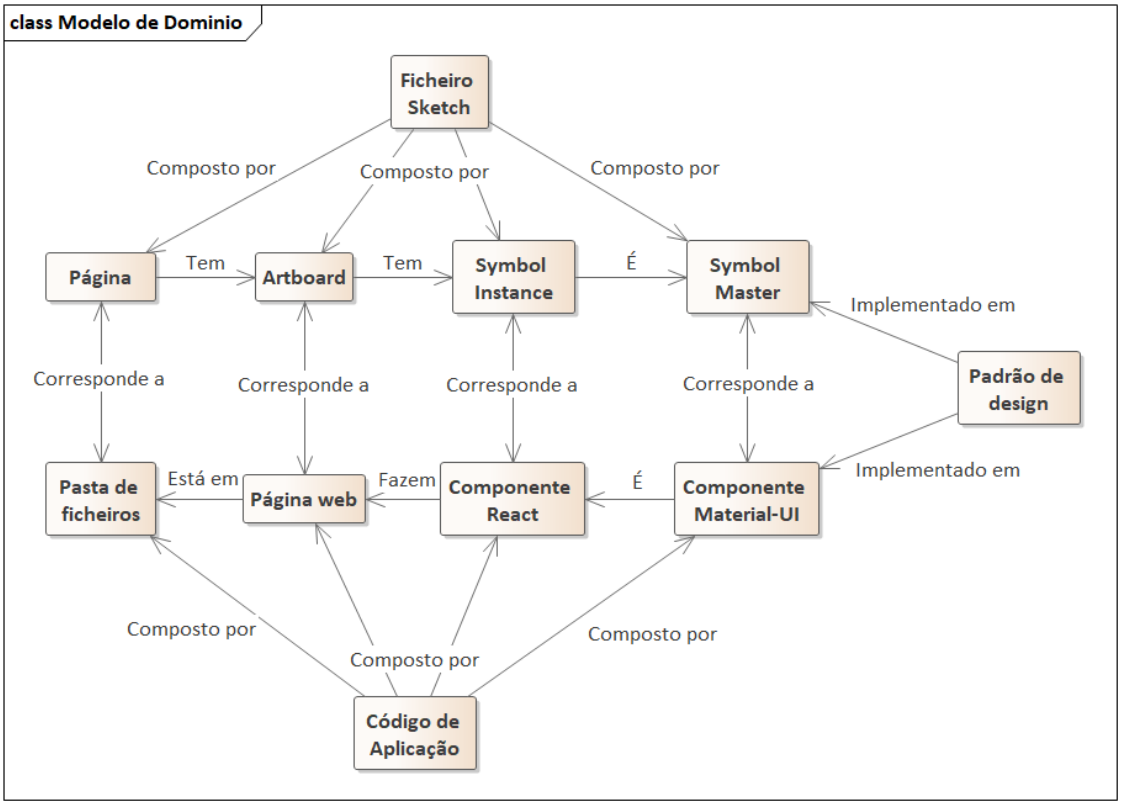


Figura 11 Modelo de domínio

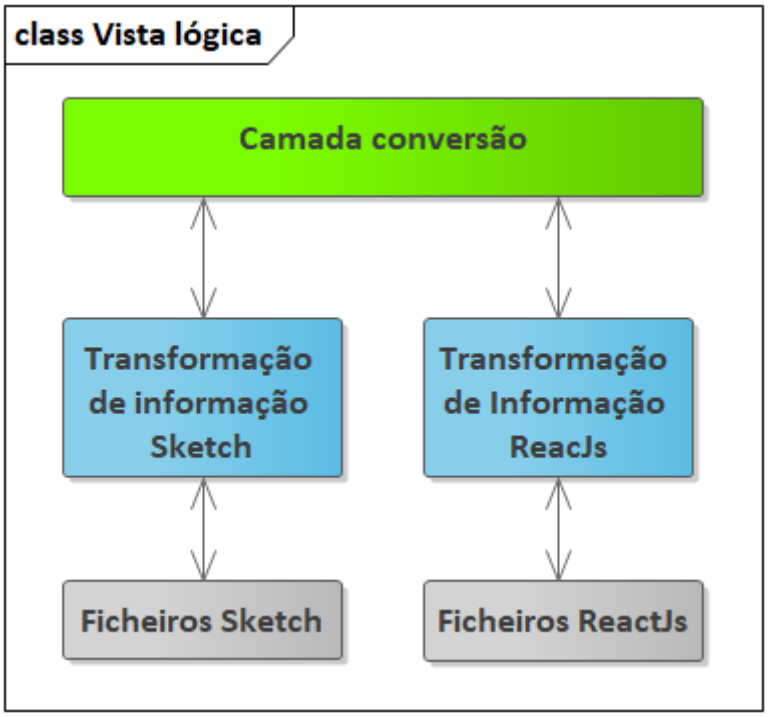


Figura 12 Vista lógica

Nas camadas referidas no parágrafo anterior há uma que se mantém maioritariamente fixa. A camada de conversão não muda com frequência, apenas pode sofrer alterações quando alguma das uma das outras camadas sofre alterações significativas, como novos componentes disponíveis ou alteração ao comportamento de componentes já existentes. Desta forma, foi decido usar duas estruturas em ficheiro JSON para mapear a conversão de conceitos. Esta decisão do uso de duas estruturas prende-se por uma melhor leitura do mesmo por parte de um ser humano. Poderia ser usada uma só estrutura que mapeava cada conceito no seu correspondente. Cada objeto teria informação em como converter de *design* para código e de código para *design*. Ora a informação de conversão em ambos os sentidos no mesmo objeto torna mais confusa a sua leitura, mesmo que os identificadores dos conceitos nas estruturas sejam claros. O uso de duas estruturas traz uma divisão física entre os fluxos de conversão, apesar do custo de manutenção associado. Ou seja, pode haver casos em que a informação tem de ser alterada em dois sítios ao invés de um.

As outras duas camadas implicam mais trabalho devido à necessidade de transformação de informação. Contudo, e à semelhança da camada anterior, cada uma delas pode ser separada em dois módulos consoante o fluxo de informação. Assim as camadas de transformação foram dividida em dois módulos, um para extrair a informação e outro para inserir informação.

Apesar de se tratar de módulos e camadas distintos não invalida que estejam presentes na mesma aplicação, desde que corretamente identificados. A construção de uma só aplicação que possa converter *design* para código ou o inverso tem a vantagem de ser uma plataforma comum para os atores envolvidos. Apesar de *designers* poderem não saber código podem ter uma perceção de como o seu *design* será apresentado na prática. E os programadores podem perceber o impacto das suas alterações no *design* e comunicar eficazmente com os *designers*. Desta forma, a Figura 13 apresenta um diagrama descritivo da arquitetura da proposta de solução.

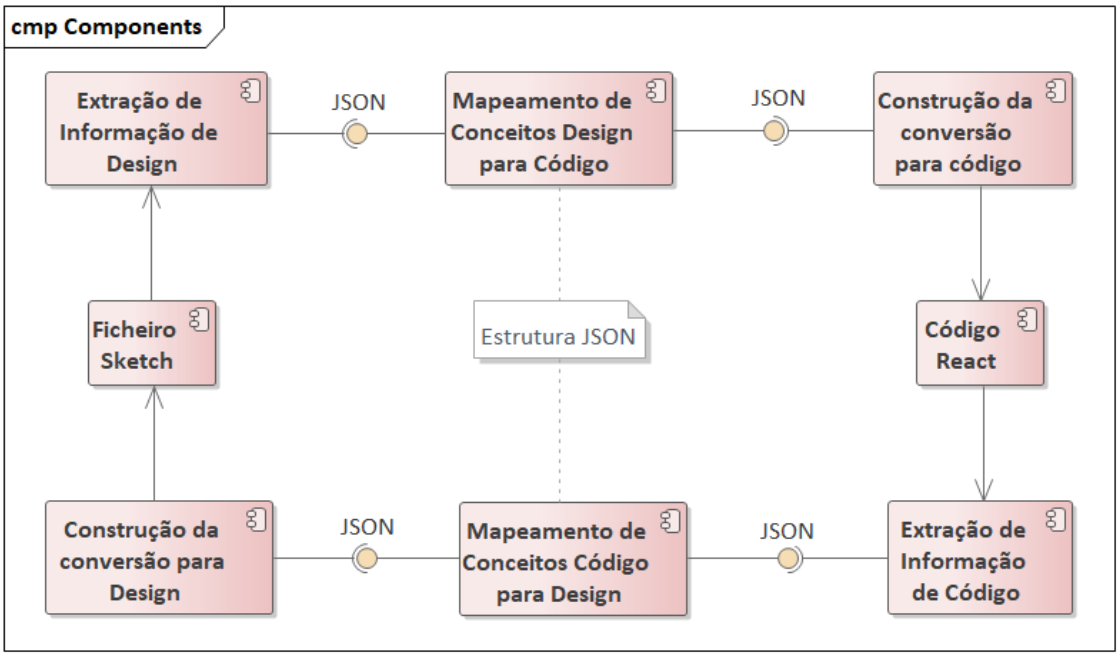


Figura 13 Diagrama de Componentes da proposta de solução

Algumas decisões arquiteturais foram influenciadas pela incapacidade de usar algumas ferramentas. Apesar de o Materia-UI ter disponível uma biblioteca Sketch, esta não pôde ser usada devido à necessidade de compra de licença da mesma. Desta forma a biblioteca material-design-stickersheet foi escolhida pois biblioteca encontra-se mais próxima das regras do padrão de *design*, mas terá sempre incompatibilidades com o Material-UI. Outro problema que influenciou as decisões arquiteturais foi o facto de o Sketch só funcionar no sistema operativo MacOS e o sistema operativo de desenvolvimento ser Windows. Isto impossibilitou que a API do Sketch pudesse ser utilizada. Com o uso da biblioteca a extração e inserção de informação seria consideravelmente mais simples, pois a API já disponibilizava métodos para o efeito. De forma a solucionar este problema foi decidido extrair a informação do ficheiro Sketch e utilizar o ambiente de execução de JavaScript NodeJS. O NodeJS dá a capacidade desenvolver uma aplicação capaz de ler e escrever ficheiros, sendo facilmente integrada com os ficheiros JSON provenientes do arquivo Sketch, assim como facilmente integrada com o código React por ambos terem por base JavaScript.

4.4.1 Extração de informação

A extração de informação consiste na obtenção dos conceitos de *design* e de código. Esses conceitos consistem nos componentes usados e respetivas propriedades, assim como a estrutura de ficheiros e pastas, e como a organização do *design*. De forma a extrair estes conceitos serão desenvolvidos dois módulos, um para a extração de conceitos de *design* dos ficheiros Sketch e outro para extração dos conceitos de código do ficheiro React.

Estes módulos extraem a navegação percorrendo os ficheiros e através de palavras-chave ou caracteres vão descobrindo os componentes, propriedades e respetivos valores. No caso da extração de *design* para código, a extração segue o processo da Figura 14.

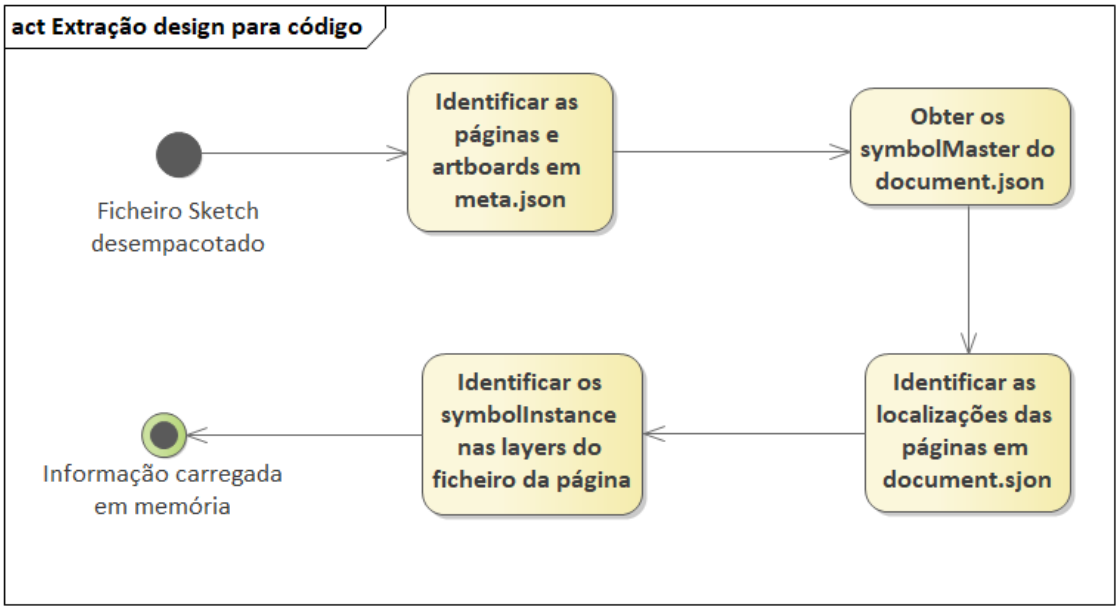


Figura 14 Fluxo da extração de *design* para código

Esta informação fica guardada em objetos JavaScript que depois podem ser utilizados para mapear os conceitos para conceitos de código.

No caso da extração de informação de código para *design*, a extração segue o fluxo presente na Figura 15.

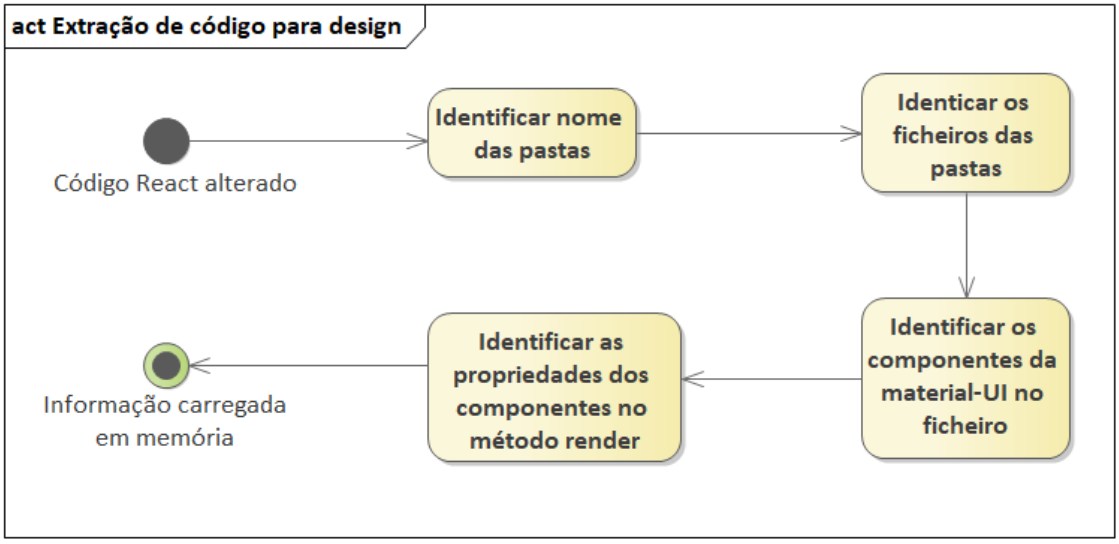


Figura 15 Fluxo de extração de código para *design*

Igualmente à extração no sentido oposto, a informação fica guardada em objetos JavaScript para ser utilizada pelo mapeamento de conceitos.

4.4.2 Mapeamento de Conceitos

O objetivo do mapeamento de conceitos consiste em fazer corresponder os *symbols* da biblioteca Sketch com os componentes da biblioteca React e o inverso. Além da correspondência entre componentes as propriedades dos mesmos também devem estar presentes.

A forma escolhida para guardar esta informação foram dois ficheiros no formato JSON, um para cada tipo de conversão. O uso de ficheiros JSON torna a integração com o desenvolvimento do sistema muito mais simplificada, pois podem ser inseridos diretamente num objeto JavaScript. Estes ficheiros encontram-se estruturados de forma a que as chaves indiquem o nome do componente e o valor seja um objeto com a *string template* do mesmo, nesse mesmo objeto encontra-se outro com os valores por defeitos usados pelas variáveis desse *template*.

O excerto de código seguinte demonstra um exemplo da conversão de *design* para código.

```
{
  "Button": {
    "importStatement": "Button",
    "renderStatement": "<Button>${text}</Button>",
    "defaults": {
      "text": "Button"
    }
  }
}
```

Código 1 Excerto do mapeamento de conceitos

4.4.3 Construção da conversão

A construção da conversão consiste na utilização do ficheiro de mapeamento de conceitos para criar ou alterar os ficheiros ou pastas do resultado final, código ou *design*. As *strings* do *template* de mapeamento de conceitos em conjunto com os valores por defeitos, ou as propriedades do conceito, serão utilizadas para gerar a *string* que será inserida no ficheiro final. Alguns conceitos também poderão ser utilizados para gerar pastas de forma ao resultado final ficar devidamente organizado. No caso da conversão de *design* para código a construção segue o fluxo da Figura 16.

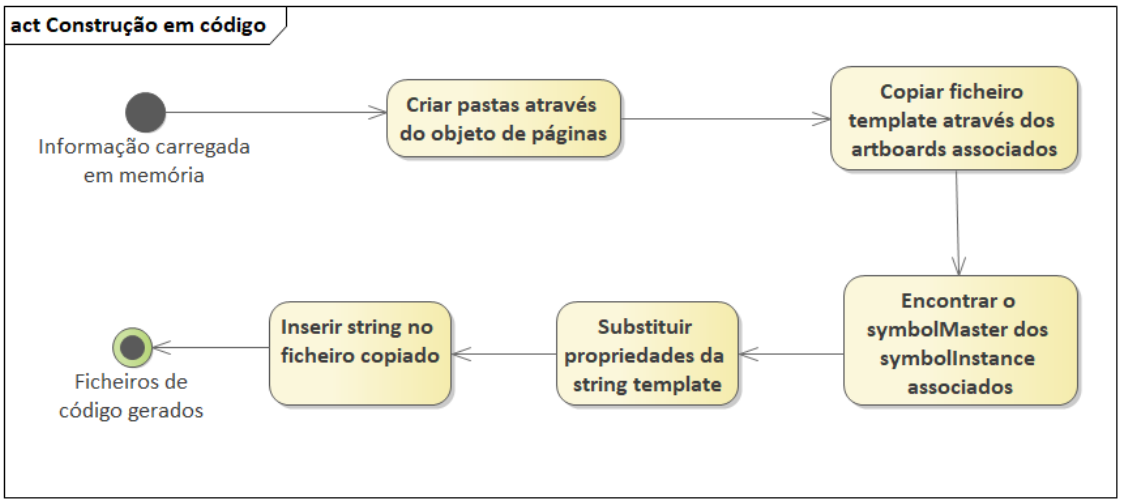


Figura 16 Fluxo da construção em código

Após a execução da construção o programador terá os ficheiros de código com a UI a replicar a existente no *design* só tendo de adicionar as ações e dados.

No caso da construção de código para *design*, a construção segue o fluxo da Figura 17.

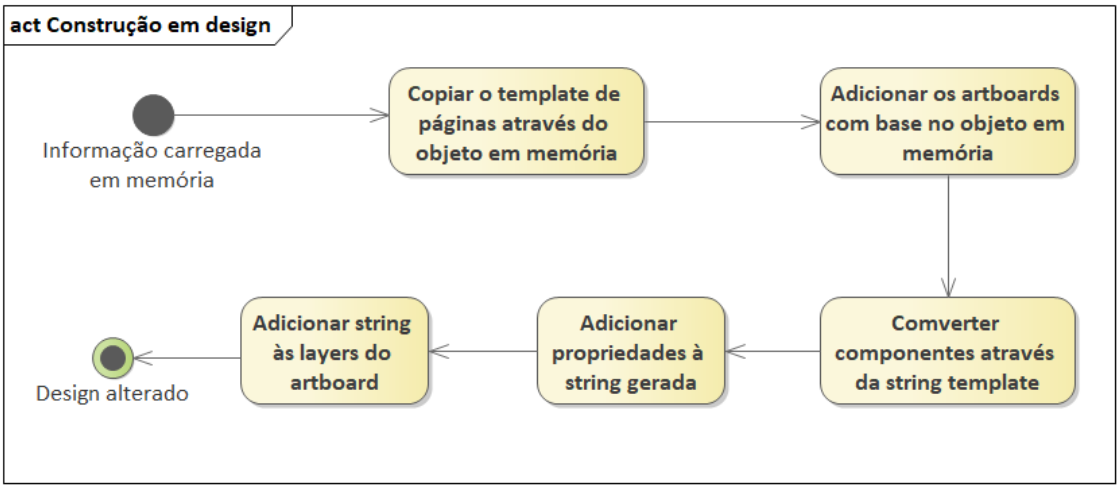


Figura 17 Fluxo da construção em design

Desta forma o *designer* terá disponível detalhes que possam ter sido codificados pelo programador.

4.5 Casos de uso

Os casos de uso representam as ações de entidades com o sistema. Na Figura 18 são apresentados os casos de uso a ser suportados pelo sistema.

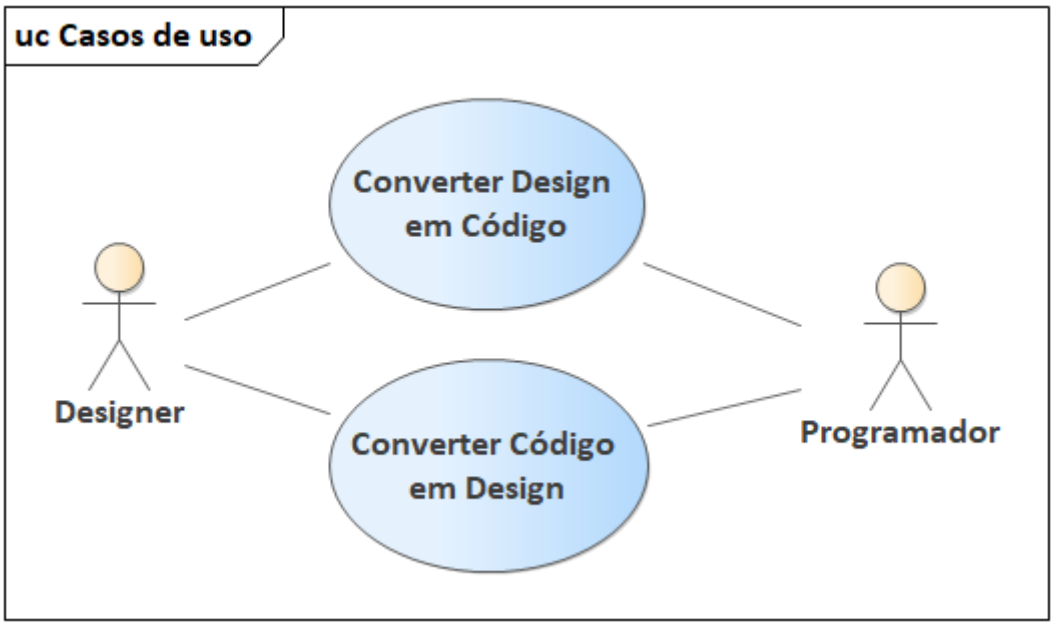


Figura 18 Diagrama de casos de uso

Apesar de ambos os utilizadores do sistema, *designer* e programador poderem executar cada um dos casos indicados na Figura 18, o *designer* executará mais vezes a conversão de design

para código e o programador o inverso, converter código para design. Do ponto de vista do *designer*, este pretende converter *design* em código quando realizou maquetes que pretende que sejam implementas. Neste caso o sistema irá ler o ficheiro do programa que o *designer* usou, irá extrair os conceitos e opções de personalização destes, fará um mapeamento entre estes e os componentes de código, e criará os ficheiros de código necessários. Do ponto de vista do programador, este pretende fornecer converter código em *design* quando fizer alterações não inicialmente previstas no *design*. Após revisão do cliente podem surgir problemas de experiência de utilização. Estes problemas de interação são mais facilmente resolvidos pelos programadores. Neste sentido quanto a experiência de utilização está confirmada é do interesse do *designer* saber quais as alterações. Assim o sistema irá remover informação lógica de programação dos ficheiros de código, e destes extrair informação de como este está construído. De seguida irá mapear os conceitos de código em *design*, e alterar o ficheiro do programa de *design* de forma a refletir as alterações no código.

4.6 Implantação

A proposta de solução consiste numa aplicação conversora. Nesse sentido a sua implantação fica em cada um dos computadores usados pelos *designers* e pelos programadores. Apesar de haver necessidade de instalação de alguns programas como o NodeJS por parte dos *designers,* e necessidade de transferência do ficheiro Sketch para os programadores, esta forma traz maior autonomia e simplicidade ao trabalhado de ambos. A Figura 19 mostra o diagrama de implantação da proposta de solução.

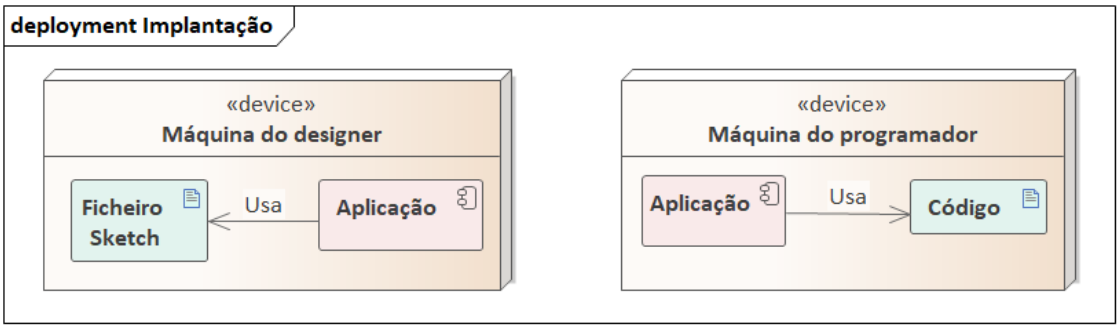


Figura 19 Diagrama de implantação

O sistema poderia ser desenvolvido de forma diferente e usar duas aplicações que submetessem a sua interface para um base de dados comum. Em vez de a estrutura comum ser um ficheiro editável, seria um modelo numa base de dados, e cada uma das aplicações saberia como converter do modelo para a sua responsabilidade. Este modelo poderia estar disponível numa Web API para os atores do processo o poderem alterar em simultâneo. Apesar de remover a necessidade de transferência de informação e instalação de ferramentas o desenvolvimento seria consideravelmente mais complexo uma vez que seria necessário manter a base de dados mantendo-a num estado coerente. Teria de se definir como manter a

informação caso o armazenamento de dados não estivesse disponível, gerir conflitos entre alterações por parte do *designer* e do programador, entre outros.

5 Desenvolvimento

O presente capítulo descreve o método de desenvolvimento e as várias iterações pelas quais este passou. Nas iterações os problemas encontrados e a sua resolução são mencionados. O capítulo possui um subcapítulo a descrever o método de desenvolvimento, seguido de cada uma das iterações. Cada uma trabalha sobre a anterior, detalhando-a ou alterando-a.

5.1 Método de desenvolvimento

O desenvolvimento do projeto seguiu uma abordagem iterativa e incremental. Esta abordagem define-se como uma sucessão de pequenas etapas de desenvolvimento onde são produzidas pequenas funcionalidades. Este método permite adaptar-se a alterações de negócio que possam existir. Estão previstas várias etapas de desenvolvimento:

1. Construção da extração de informação do Sketch, inicialmente com uma instância de cada conceito Sketch (página, *artboard*, *symbolInstance* e *symbolMaster*)
2. Construção do ficheiro de mapeamento de *design* para código
3. Construção da criação dos ficheiros de código e respetiva conversão do mapeamento
4. Adição de novas instâncias de conceitos Sketch, várias páginas, *artboards*, *symbolInstances* e *symbolMasters*.
5. Adição de propriedades estéticas ao ficheiro de mapeamento
6. Extração de informação do código React para *design*
7. Construção do ficheiro de mapeamento de código para *design*
8. Edição dos ficheiros de *design* e respetiva conversão do mapeamento
9. Adição de propriedades estéticas ao ficheiro de mapeamento

5.2 Iteração 1

Na iteração 1 foi utilizada para exploração das tecnologias, saber se estas eram compatíveis.

Nesta iteração foi descoberto que o desenvolvimento usando a API do Sketch não era possível com as ferramentas disponíveis. A ideia era desenvolver um plugin para o Sketch, utilizando a API JavaScript do Sketch. Mas os plugins são executados no próprio programa, por intermédio de scripts. Como o programa, à data, só está disponível no sistema operativo MacOS, só é possível desenvolver plugins com um computador com esse sistema operativo. Pelo autor não ter acesso direto a computador com MacOS, a opção de um plugin tornou-se impossível. Contudo como mencionado na secção 4.3.1, o ficheiro Sketch não é mais que um arquivo de ficheiros JSON. Estes ficheiros não estão dependentes do sistema operativo, sendo iguais em MacOS e Windows, este último o sistema operativo utlizado pelo autor no desenvolvimento.

Assim optou-se pelo uso do ficheiro para extração de informação. O desenvolvimento manteve-se na linguagem JavaScript pela interoperabilidade com os ficheiros JSON. Desta forma foi necessário recorrer a um ambiente de execução JavaScript de forma a correr o código, neste caso NodeJS.

Durante o esta fase também foi descoberta uma biblioteca de componentes Sketch dos criadores da biblioteca Material-UI. A possibilidade do uso desta biblioteca em substituição da material-design-stickersheet foi ponderada de forma a simplificar a integração. Contudo por esta ser paga, a ideia foi abandonada, mantendo-se a ideia inicial de usar a biblioteca material-design-stickersheet. Também foi explorada a utilização de uma biblioteca proprietária de uma empresa, mas esta era consideravelmente mais complexa, pois tinha um maior número de elementos reutilizáveis. E seria mais inconveniente produzir alterações à mesma.

5.3 Iteração 2

Na iteração 2 o objetivo era ter o esqueleto da aplicação. Esta aplicação teria de ser capaz de fazer todo o ciclo de conversão de *design* para código, mas no design só existia um elemento simples, um botão. O mesmo *design* seria assim constituído por uma página, um *artboard* e um *symbolInstance*. O mapeamento de conceito só teria assim um elemento e não seriam admitidas propriedades. Ou seja, o pretendido era transcrever somente o componente ignorando qualquer propriedade estética. No lado do código, a pasta, o ficheiro e o conteúdo teriam de ser corretamente criados.

Foi desenvolvido um modelo de objetos JavaScript para cada um dos seguintes conceitos: *artboard*, páginas, *symbolMaster* e *symbolnstance*. Nestes objetos foi introduzida a informação presente no ficheiro. Após a informação estar carregada em memória a mesma foi percorrida de forma a ser transformada em código. Este passo foi possível através de um ficheiro JSON de mapeamento de conceitos, no qual estava presente o conceito e o *template* da *String* de código correspondente e o *template* do ficheiro de componente.

A extração de informação iniciou-se na pesquisa das páginas e *artboards* no ficheiro de meta-dados, meta.json. Neste ficheiro foi percorrido o conteúdo do objeto *pagesAndArtboards*. Das chaves do mesmo obteve-se os identificadores das páginas e da propriedade *name,* do valor de

cada objeto, obteve-se o nome da página. Para cada um dos objetos que representavam as páginas, obtiveram-se os *artboards* relacionados com a página através da propriedade *artboards*. Estes foram contruídos de forma idêntica, na medida em que a chave do objeto indicava o identificador do *artboard* e a propriedade *name* do mesmo, o nome desse *artboard*. Os *symbolMaster* foram obtidos pela através da navegação pelo conteúdo da propriedade *foreignSymbols,* do objeto correspondente ao ficheiro *document.json*. Sendo que que o conteúdo dos *symbolMaster* se encontravam numa propriedade do mesmo nome. Desta propriedade foi adquirido o identificador do *symbolMaster* e o respetivo nome. Da mesma forma indicada no presente parágrafo. Contudo o nome apresenta uma particularidade em que o nome deste identifica o tipo de componente no mapeamento de conceitos. Isto foi uma opção de *design* por parte de autor e implicou alterações à forma como a biblioteca estava contruída. Nesta os nomes dos componentes estão separados pelo caracter "/" por forma a apresentar os *symbols* numa interface com menus hierárquicos em que cada "/" representa um nível inferior de hierarquia. A biblioteca *material-design-stickersheet* apresentava componentes em diferentes níveis hierárquicos e alguns nomes de componente apresentavam caracteres semelhantes a emojis. De forma a solucionar este problema foi definido que o primeiro elemento da hierarquia seria correspondente ao componente e os seguintes poderiam ser a casos específicos de reutilização desse componente. Desta forma para um componente botão com dois tamanhos, este poderia existir com os seguintes nomes: "Butão/Pequeno", "Botão/Grande". A decisão de o primeiro elemento ser correspondente ao componente teve como princípio a simplificação do código, mas mantendo a capacidade de escalabilidade da solução. Por cada página estar representada por um ficheiro numa determinada localização no sistema de ficheiros, esta localização era necessária. A localização das páginas encontra-se no ficheiro *document.json* na propriedade *pages*. Esta propriedade é um contentor de objetos de páginas e a propriedade "*_ref*" indica a localização no sistema de ficheiros. Assim o contentor da propriedade *pages* foi iterado e cada localização do ficheiro foi associada ao modelo de páginas correspondente. A única em coisa em falta eram as *symbolInstances* presentes nos ficheiros de páginas obtidos anteriormente. Estas páginas foram carregadas em memória e para cada uma delas foi iterada a propriedade *"layers"* correspondente ao conteúdo presente na página. Para esta propriedade só foram analisados os objetos cuja propriedade *class* era igual a *"artboard"*, qualquer outro elemento excluído de um *artboard* seria ignorado. No *artboard* também foi iterada a propriedade *"layers"* para aceder ao conteúdo do mesmo. Neste sentido as instâncias de *symbols* foram extraídas para o respetivo modelo, com as associações para o *artboard* e *symbolMaster* que lhes dizem respeito, guardando a ID destes últimos.

O excerto de código 2, apresentado de seguida, mostra cada uma das entidades extraídas do ficheiro no formato JSON.

```
pages {
  '5D61C615-1587-43F0-A960-7B2C2B4355F2': {
    name: 'Page 1',
    location: 'pages/5D61C615-1587-43F0-A960-7B2C2B4355F2'
  }
}
artboards {
  'E13B8420-5600-4840-8350-77745DBADD2A': {
     name: 'Artboard',
```

```
    pageId: '5D61C615-1587-43F0-A960-7B2C2B4355F2'
  }
}
symbolMasters {
  '45A3D419-9A9D-4837-90D6-CA550A1AE267': 'Button',
  '7F63DA96-9549-455B-A1F7-9A5620982503': 'ΩElements'
}
symbolInstances {
  '25B47232-BE94-4E02-88C8-C8B8B12D2A62': {
    artboardId: 'E13B8420-5600-4840-8350-77745DBADD2A',
    symbolMasterId: '45A3D419-9A9D-4837-90D6-CA550A1AE267',
    name: 'Button'
  }
}
```

Código 2 Impressão de consola do conteúdo do modelo após extração

Após o carregamento da informação do projeto Sketch no modelo em memória, este teria de ser convertido em código. Assim, o modelo foi iterado, começando pelas páginas. Estas iriam corresponder a pastas de ficheiros de código. Desta forma os *designers* podiam organizar o seu trabalho em áreas comuns e os programadores beneficiar dessa organização obtendo páginas web organizadas. Para cada página foi iterado os artboards correspondentes. Os artboards seriam as páginas web, um ficheiro descrevendo um componente React.

Visto que os componentes React seguem uma estrutura bem definida, foi criado um ficheiro modelo que representasse um componente genérico. Este ficheiro tinha etiquetas para identificar campos variáveis, aos quais posteriormente seria adicionado o conteúdo da página web. As etiquetas seriam comentários no código com um nome identificador como "Importações", precedido do caracter "$", uma vez que este caracter isolado não tem significado especial em JavaScript. O ficheiro modelo era assim copiado para a pasta correspondente à respetiva página. O excerto de código 3 apresenta o conteúdo desse ficheiro.

```
//Imports
import React, { Component } from 'react'
import {
  //$Imports
} from 'material-ui'

class ComponentName extends Component{

  render(){
    return <div>
      {/*$Childs*/}
    </div>
  }
}
```

Código 3 Ficheiro modelo de componente ReactJs

A única coisa em falta era conteúdo das páginas, os *symbolInstance*. Assim foi obtido, através dos modelos, os *symbolInstance* do respetivo *artboard* em análise. A informação deste não era necessária para saber a que componente Material-UI dizia respeito, mas seria possível obter informação sobre opções de personalização ou texto dos mesmos. O componente ao qual o

symbolInstance dizia respeito foi obtido pela referência ao *symbolMaster* existente no mesmo. Sabendo o *symbolMaster* foi possível consultar o ficheiro de mapeamento de conceitos de forma o obter as strings modelo, de importação e renderização, para substituir no ficheiro do *artboard*. O excerto de código 4 apresenta o conteúdo do ficheiro de mapeamento de conceitos.

```
{
  "Button": {
    "importStatement": "Button",
    "renderStatement": "<Button>${text}</Button>",
    "defaults": {
      "text": "Button"
    }
  }
}
```

Código 4 Estrutura de dados de mapeamento de conceitos

Às strings modelo foi substituído propriedades por defeito por propriedades presentes no *symbol*nstance. Com a informação pronta a única etapa em falta era inseri-la no ficheiro correspondente. Ficheiro esse que teria o mesmo nome do *artboard* indicado no projeto Sketch. Por fim, o ficheiro foi lido para uma variável, as etiquetas referentes à importação do componente Material-Ui, nome do componente e objetos de renderização, foram substituídas e foram escritas no mesmo ficheiro, alterando o conteúdo inicial. Após a iteração por todo o modelo o código estava pronto a ser entregue aos programadores para estes poderem trabalhar sobre o mesmo. O excerto de código 5, apresenta o ficheiro de código gerado pelo sistema. Todo este processo foi realizado de forma síncrona.

```
//Imports
import React, { Component } from 'react'
import {
  Button,

} from 'material-ui'

class Artboard extends Component{

  render(){
    return <div>
      <Button>Button</Button>

    </div>
  }
}
```

Código 5 Código gerado pelo sistema

Durante esta fase de desenvolvimento também foi explorada a possibilidade de ter uma base de dados em memória SQLite. Esta base de dados seria uma modelo do conteúdo do *design* e respetivas relações. Nela também poderiam estar definidas as regras de mapeamento de conceitos. O uso de uma base de dados poderia ser facilmente integrado num trabalho futuro caso se se trabalhasse num sistema cliente-servidor. Apesar de vantajosa pela possibilidade de persistência de dados e fácil obtenção de dados relacionados, nesta fase do desenvolvimento o seu uso não era vantajoso. A necessidade de gestão de informação e as tecnologias utilizadas

serem maioritariamente desconhecidas traziam maior complexidade dificultando a conclusão do objetivo.

5.4 Iteração 3

A iteração2 criou a base da solução pelo que nas seguintes iterações o desenvolvimento foi consideravelmente mais simples. Na iteração 3 o objetivo passou por adicionar a conversão um maior número de *symbols*. Desta forma o *designer* teria um maior leque de opções para construir os seus *designs* além de botões. Contudo durante o desenvolvimento desta iteração foi detetado que havia vários *symbols* para o mesmo componente, sendo que alguns deles não eram compatíveis com a regra escolhida para identificação dos mesmos. Como mencionado no capítulo 5.3, existência de caracteres semelhantes a emojis no nome do *symbol* tornavam complexa a sua correta identificação. A existência de versões de *symbols* que correspondiam ao mesmo componente também aumentava a complexidade do problema. Assim foi decido reduzir a quantidade de *symbols* disponíveis. Como Menus, Caixas de texto, Cartões, Caixas de seleção e Menus de seleção.

Foi criada uma biblioteca onde foram extraídos os *symbols* da biblioteca original. Esta solução tornou o sistema incompatível com a biblioteca inicialmente escolhida, mas reduziu a complexidade do desenvolvimento. Esta simplificação não impediu a possível compatibilidade com a biblioteca inicial, pois os restantes symbols poderiam ser adicionados de forma iterativa e incremental.

5.5 Iteração 4

Na iteração 4 o objetivo passou pela adição de conversão de texto ao *design*. Tando ao nível de texto livre como a elementos textuais dos *symbols* existentes. Este último semelhante à alteração de algumas propriedades estéticas do *symbol*. Também foi alterada a biblioteca de forma a ter um conjunto de *symbols* que permitisse a construção de formulários. Sendo eles: Botões, Caixas de texto, Caixas de seleção, Cartões e Caixas de notificação. O excerto de código 6 apresenta o ficheiro de mapeamento de conceitos na quarta iteração.

```
{
  "Button": {
    "importStatement": "Button",
    "renderStatement": "<Button>${text}</Button>",
    "defaults": {
      "text": "Button"
    }
  },
  "Chips": {
    "importStatement": "Chip",
    "renderStatement": "<Chip label=\"${text}\"/>",
    "defaults": {
      "text": "Chip"
```

```
    }
  },
  "Checkboxes": {
    "importStatement": "Checkbox",
    "renderStatement": "<Checkbox/>",
    "defaults": {
    }
  },
  "Text fields": {
    "importStatement": "TextField",
    "renderStatement": "<TextField/>",
    "defaults":{
    }
  },
  "Card": {
    "importStatement": "Card",
    "renderStatement": "<Card></Card>",
    "defaults": {
    }
  }
}
```

Código 6 Mapeamento de conceitos na quarta iteração

Para o texto livre foi criado um modelo que o representasse com o texto do mesmo, e o *artboard* ao qual dizia respeito. A criação do modelo foi adiciona logo após a criação do modelo das *symbol*nstaces, quando o conteúdo de um *artboard* se encontrava em análise. Na seção de geração de código, a criação do texto livre foi adicionada à *string* de renderização após a instanciação dos componentes. Esta solução não coloca o texto no lugar correto, aparecendo primeiro os componentes e posteriormente o texto. O mesmo acontece com os *symbols*, estes não aparecem no local correto na página, mas sim pela ordem em que aparecem no documento. Contudo deixa de ser necessário a criação dos mesmos, sendo somente necessário colocar os componentes na disposição correta. Um exemplo desse modelo pode ser visto no código 7 apresentado de seguida.

```
freeText {
  '20485411-1A0F-4653-AE1E-B5AC86BB8278': {
    artboardId: '1E828F14-BB14-429D-8A6C-63ED5D393816',
    text: 'Regista a tua  Actividade Desportiva '
  } ,
  '03DC3513-F6B0-436D-BF69-F04B8D6CA74D': {
    artboardId: '1E828F14-BB14-429D-8A6C-63ED5D393816',
    text: 'Os dados recolhidos serão tratados, analisados e divulgados de
acordo com as regras de investigação científica.\n'
  },
  '3A40F4B4-7185-41B5-B63F-E13E3C000D31': {
    artboardId: '61BDA983-71AB-4A55-9956-EE57EF28C7C6',
    text: 'Dados pessoais'
  },
  'C9FFEAC7-C091-456F-AADA-4D7B5D2C375D': {
    artboardId: '61BDA983-71AB-4A55-9956-EE57EF28C7C6',
    text: 'Atividade Desportiva'
  },
  'B0B96BF9-4FAD-4E6F-A9CF-11113C21B67C': {
     artboardId: '61BDA983-71AB-4A55-9956-EE57EF28C7C6',
     text: 'Nome'
  },
```

```
  '31A20604-6DF5-476E-9AB1-EBC2DDA5C7F1': {
     artboardId: '61BDA983-71AB-4A55-9956-EE57EF28C7C6',
     text: 'Idade'
  },
  'F1BDA45C-7356-4E67-A71E-D82872022DDF': {
    artboardId: '61BDA983-71AB-4A55-9956-EE57EF28C7C6',
    text: 'Total  horas semanais'
  }
}
```

Código 7 Modelo de texto livre

Para os elementos textuais dos *symbolMaster*, foram identificados aqueles que possuíam texto relevante para substituição, texto que tivesse um intuito permanente e não servisse como texto de substituição ou de apoio. Como o texto de carácter não permanente costuma depender da funcionalidade, este poderia criar confusão aos programadores ao existir logo quando recebiam a aplicação. Seria mais intuitivo para estes adicionar o texto no momento do desenvolvimento da funcionalidade que eliminar texto já existente. Dos *symbols* identificados no início do presente capítulo só o botão e a caixa de notificações é que cumpriam as restrições. No código 6 é possível ver isto, pois só estes é que têm um elemento *text*, no objeto *defaults*. Ao analisar os ficheiros de *design* percebeu-se que existia uma propriedade textual que podia ser alterada nestes componentes. Esta propriedade tinha um ID que era substituído pelas *symbolInstances* na propriedade *overrideValues*. Nesta propriedade podiam estar presentes várias opções de personalização, assim como vários textos do componente. Assim para os componentes selecionados da biblioteca foi descoberta o identificador do texto através da análise do valor por defeito. Esse identificador foi usado diretamente no código na extração do *symbolnstance*, para extrair texto a alterar. E na altura da iteração do mapeamento de conceitos, caso existe texto a ser alterado, o valor no respetivo modelo era utilizado. Um dos ficheiros gerados pode ser visto no excerto de código 8. De notar que, no mesmo excerto de código, o componente não consegue ser interpretado por o seu nome é o número um. Isto acontece devido ao nome provir do nome do *artboard* no ficheiro ser Sketch ser 1. Este foi um problema detetado após a sessão de experimentação, descrita no capítulo 6.2.

```
//Imports
import React, { Component } from 'react'
import {
  Button,
Typography,

} from 'material-ui'

class 1 extends Component{

  render(){
    return <div>
      <Button>Button</Button>

<Typography>Regista a tua  Actividade Desportiva </Typography>
<Typography>Os dados recolhidos serão tratados, analisados e divulgados de
acordo com as regras de investigação científica.
</Typography>
```

```
            </div>
        }
    }
```

Código 8 Exemplo de um ficheiro gerado na quarta iteração

Os ficheiros de código também estavam corretamente organizados consoante as páginas usadas *design* em Sketch. A Figura 20 mostra a organização de pastas gerada com base num design fornecido na sessão de experimentação do capítulo 6.2. Neste caso a pasta "*Page 1*" corresponde a uma página no documento Sketch e os ficheiros "1", "2", e "3" a *artboards*.

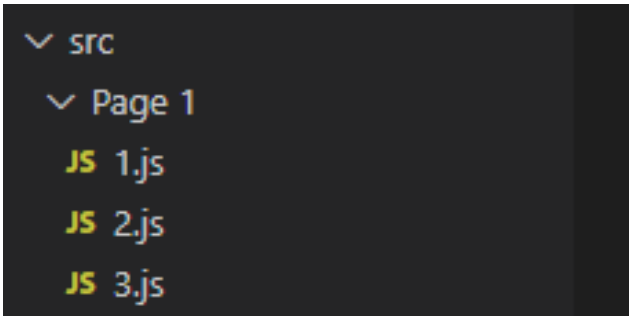


Figura 20 Organização de pastas após conversão

6 Experimentação e avaliação

No presente capítulo são abordados os métodos de experimentação e avaliação. A experimentação foi realizada através de sessões de experimentação onde foram produzidos requisitos para o sistema e corrigidos os resultados do mesmo. A avaliação foi realizada através de inquéritos de forma a obter informação sobre a utilização da proposta e da qualidade ao nível da utilização e do resultado.

6.1 Método de avaliação

O método de avaliação serão os inquéritos realizados após as sessões de experimentação descritas em maior detalhe na secção 6.2. As sessões de experimentação são atividades que os designer e programadores terão de desenvolver para fornecer recursos ao sistema e produzir aplicações funcionais. Nos inquéritos pretende-se obter informação que permita aferir a qualidade subjetiva que cada participante nas sessões atribui à solução. Os inquéritos estão divididos em três partes, uma parte de caracterização do participante, uma avaliação das funcionalidade e qualidade da solução, e uma secção de preferências de funcionalidades futuras. Os resultados dos inquéritos serão utilizados para produzir uma análise de dados que a avaliará a qualidade da solução apresentada.

6.2 Sessões de experimentação

De forma a averiguar resultados foi realizada uma sessão de experimentação com designers e programadores. Esta sessão consistiu no desenvolvimento de uma pequena interface utilizando a biblioteca Sketch desenvolvida no contexto do projeto. Os designers seguiram um guião com um conjunto de requisitos que a interface devia cumprir e os resultados esperados. Estes resultados seriam usados como entrada na solução. No processo tiveram liberdade para criar a

interface desejada. Esta sessão não foi acompanhada, contudo os *designers* tiveram de responder a um inquérito para avaliar a experiência de utilização. No final do exercício, o projeto Sketch foi enviado para o autor da tese para poder ser convertido para código. Esta decisão deveu-se à necessidade de os *designers* terem de instalar ferramentas fora das habitualmente usadas no seu trabalho.

Cada projeto Sketch produzido pelos *designers*, foi convertido em código. Este código foi posteriormente fornecido aos programadores para correções e avaliação da conversão. Contudo quando os projetos foram convertidos percebeu-se que a conversão não estava funcional. Isto deveu-se a um conceito que a solução não estava a ter em consideração. No desenvolvimento de *design* em Sketch é possível agrupar vários objetos para simplificar o trabalho do *designer* tornando-o mais rápido. Este agrupamento de objetos produz níveis de hierarquia cuja solução não tem considera. Mesmo assim houve dois projetos em que tal situação não acontecia. O código produzido por estes projetos foi o que foi enviado aos programadores.

Os programadores receberam um projeto ReactJs base, com as páginas e componentes exportados do *design*, um ficheiro PDF com as imagens maquete do *design* e um guião com os objetivos pretendidos. O resultado do código não estava igual ao *design*, e apresentava erros de compilação, pelo que o objetivo dos programadores foi corrigir os defeitos encontrados. A maioria dos erros de compilação, eram nomes de componente que não eram compatíveis com as regras de código e questões de apresentação. Após a conclusão do exercício os programadores responderam também a um inquérito para avaliar a solução e a sua utilidade. Esta sessão de avaliação também não foi acompanhada.

6.3 Análise de dados

A análise de dados tem por base os inquéritos realizados aos *designers* e programadores. No total foram inquiridos, 9 sujeitos, 5 *designers* e 4 programadores tentando-se obter variabilidade na experiência de trabalho de cada um. Nesta análise de dados, primeiro serão apresentados os resultados do inquérito referente ao uso do Sketch e posteriormente os resultados das correções em ReactJs. No final será explorada possíveis ligações entre ambos, ciente que o número de experiências é reduzido. Por ser necessário despender tempo na realização da tarefa e pela solução se encontrar numa fase preliminar, a obtenção de sujeitos de estudo foi insuficiente. Pretende-se com os resultados ficar com uma tendência de opiniões. Sendo estas consideradas mais próximas da verdade quando a variação entre respostas é praticamente inexistente.

6.3.1 Dados de experiências Sketch

Os dados da presente análise provêm do inquérito no anexo 8.1.2. Na análise de dados será ao nível de cada pergunta do inquérito. A primeira parte desta análise diz respeito à caracterização dos inquiridos.

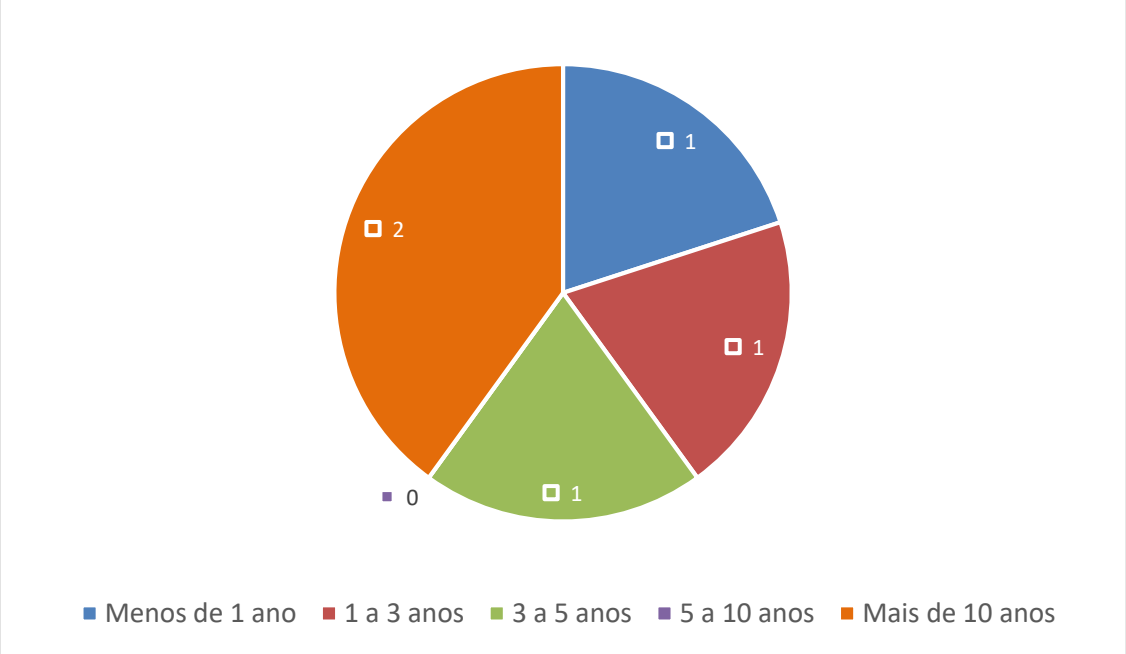


Figura 21 Qual é a sua experiência em UI/UX

Esta pergunta pretendia caracterizar a experiência dos inquiridos consoante a sua experiência na área de interfaces com o utilizador (UI) e experiência de utilização (UX). Na Figura 21 é possível ver que a amostra é distribuída, evitando casos de dados enviesados. Duas pessoas têm mais de 10 anos de experiência, nenhum dos inquiridos têm tem entre 5 a 10 anos. Os restantes três inquiridos distribuem-se igualitariamente com um elemento cada, nas categorias de menos de 1 ano de experiência, entre 1 a 3 anos e entre 5 a 10.

Tendo em conta que o Sketch foi uma tecnologia central na solução foi importante conhecer a experiência dos inquiridos nesta tecnologia.

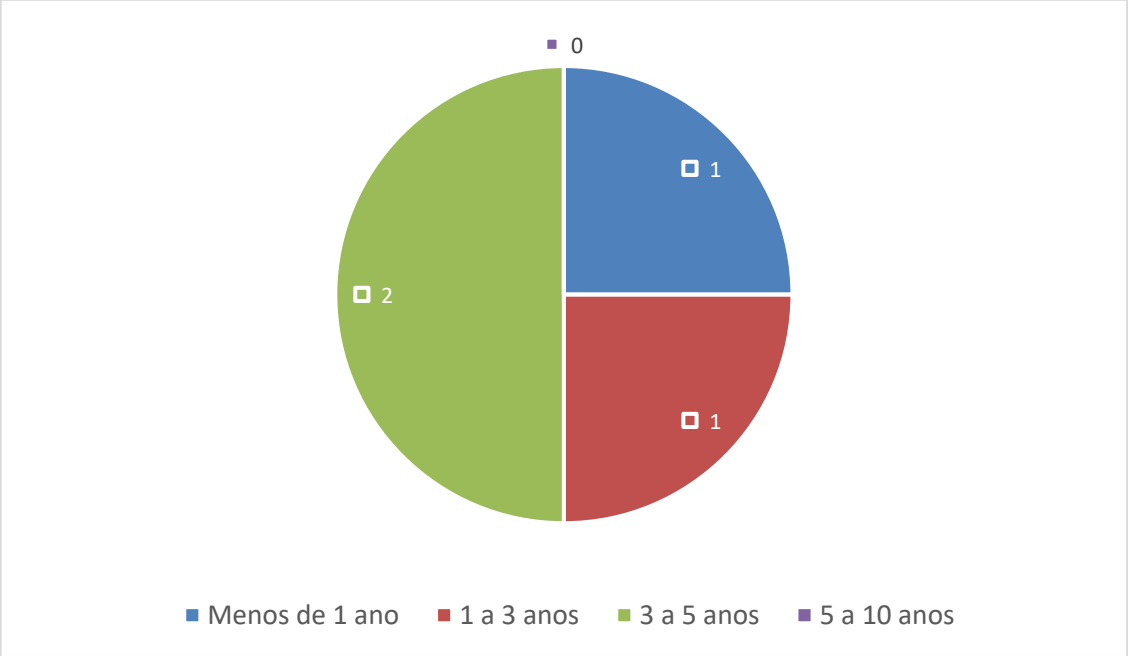


Figura 22 Qual é a sua experiência na utilização da ferramenta Sketch?

Como pode ser visto na Figura 22 a maior parte dos inquiridos possui entre 3 a 5 anos de experiência Sketch. Existem uma pessoa com menos de um 1 de experiência na tecnologia, outra com 1 a 3 anos de experiência, e não existem inquiridos com mais de 5 anos de experiência. Nesta pergunta entende-se que a experiência na tecnologia é recente, provavelmente têm capacidade em usar a tecnologia, mas podem não ser mestres no uso da mesma.

Também foi perguntado o sexo dos inquiridos na tentativa de saber se existiam diferenças entre sexos.

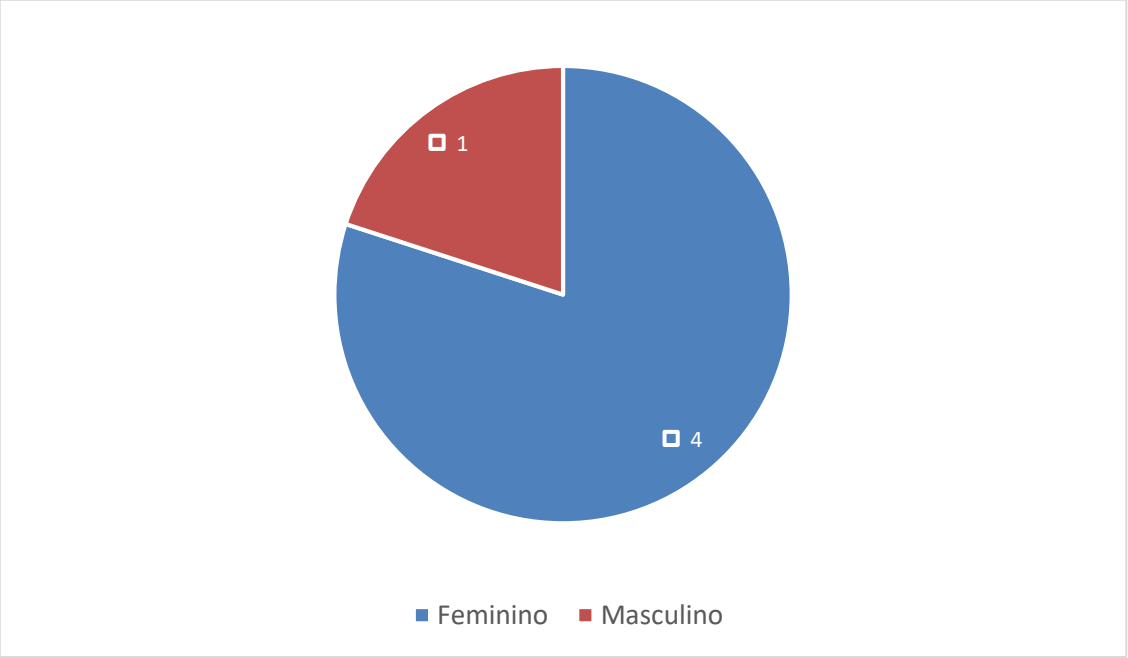


Figura 23 Qual é o seu sexo?

No inquérito para a usabilidade de Sketch denota-se que a maior parte dos inquiridos, é do sexo feminino.

Assim conclui-se que a amostra é maioritariamente do sexo feminino, cuja experiência de trabalho em Sketch é recente, mas a experiência na área de experiência com utilizador e de utilização variada.

A secção de perguntas seguintes pretendeu obter informação sobre o trabalho desenvolvido. Inicialmente foi questionado a perceção do questionário, esta pergunta tinha como objetivo perceber se eventuais erros no objetivo pudessem ter origem no entendimento do enunciado.

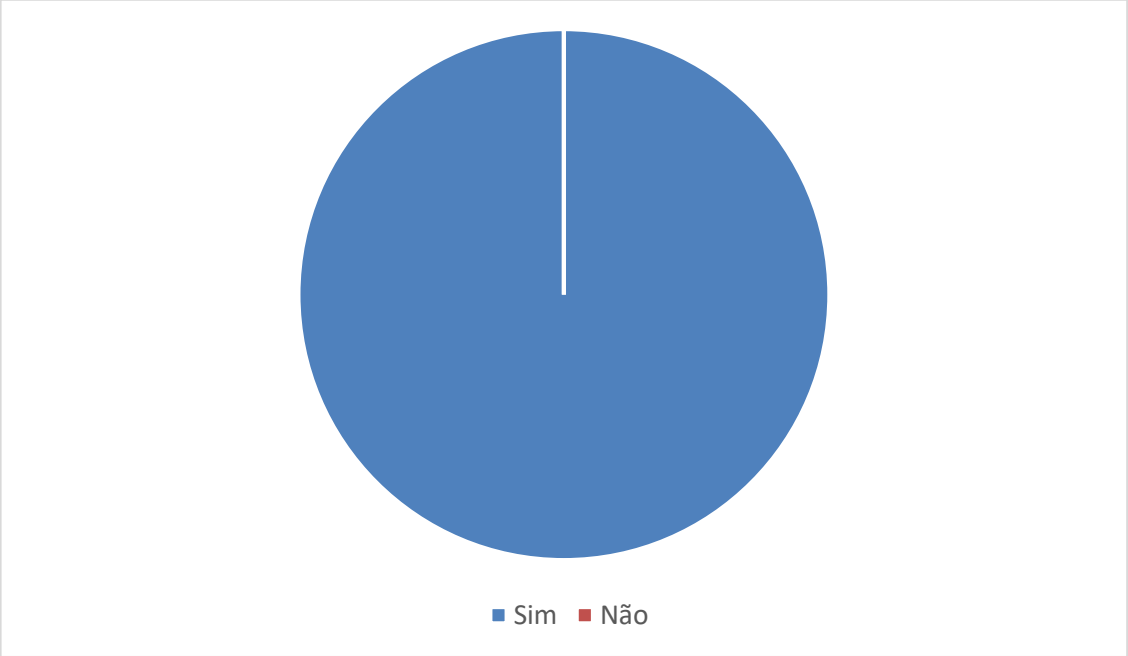


Figura 24 O enunciado foi suficientemente claro no pretendido?

Com base nos resultados da Figura 24 conclui-se que o enunciado foi corretamente entendido, sendo que os erros da tarefa não tiveram origem na perceção do enunciado.

Pela forma de organização de trabalho do inquirido poder ser diferente da imposta no enunciado, foi questionado se havia alguma diferença face à organização do inquirido. Especulava-se que uma organização diferente de páginas e *artboards* por parte do inquirido, pudesse originar problemas no resultado da tarefa.

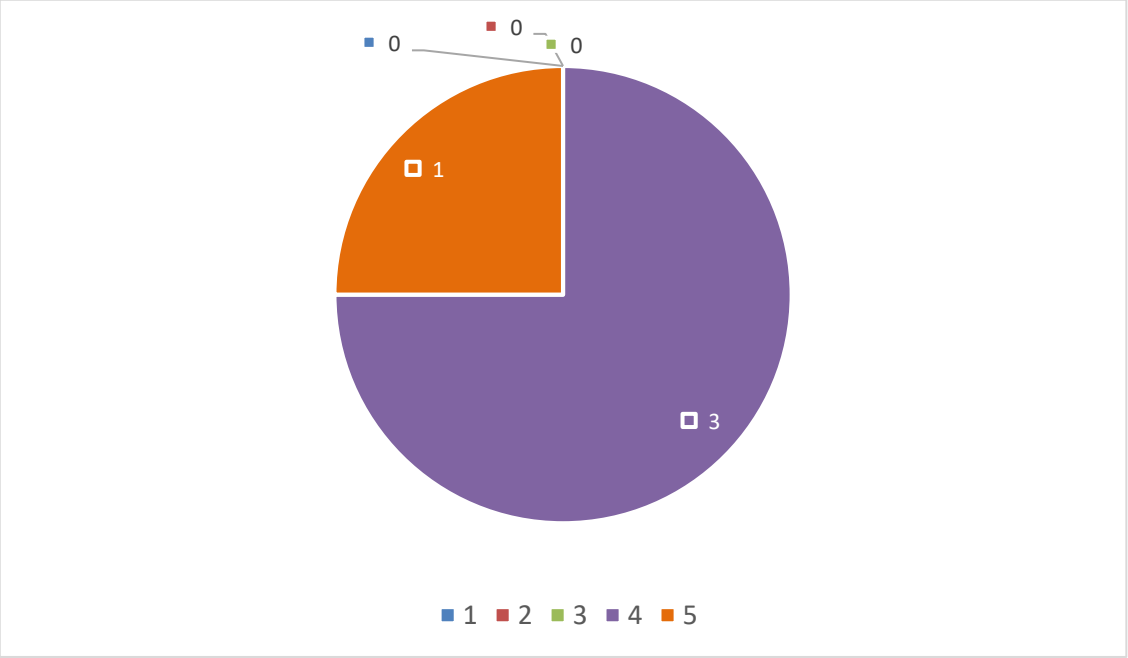


Figura 25 Classifique de 1 a 5, sendo 1 Totalmente Distinta e 5 Bastante Semelhante, se a organização de página e artboards imposta é semelhante à sua forma habitual de trabalho

Assim denota-se que que maior parte dos inquiridos têm uma organização semelhante do seu trabalho à imposta pela tarefa, sendo a média de respostas de 4,4 valores.

Tendo em conta as restrições pretendeu-se obter o impacto que estas apresentaram no desenvolvimento da tarefa.

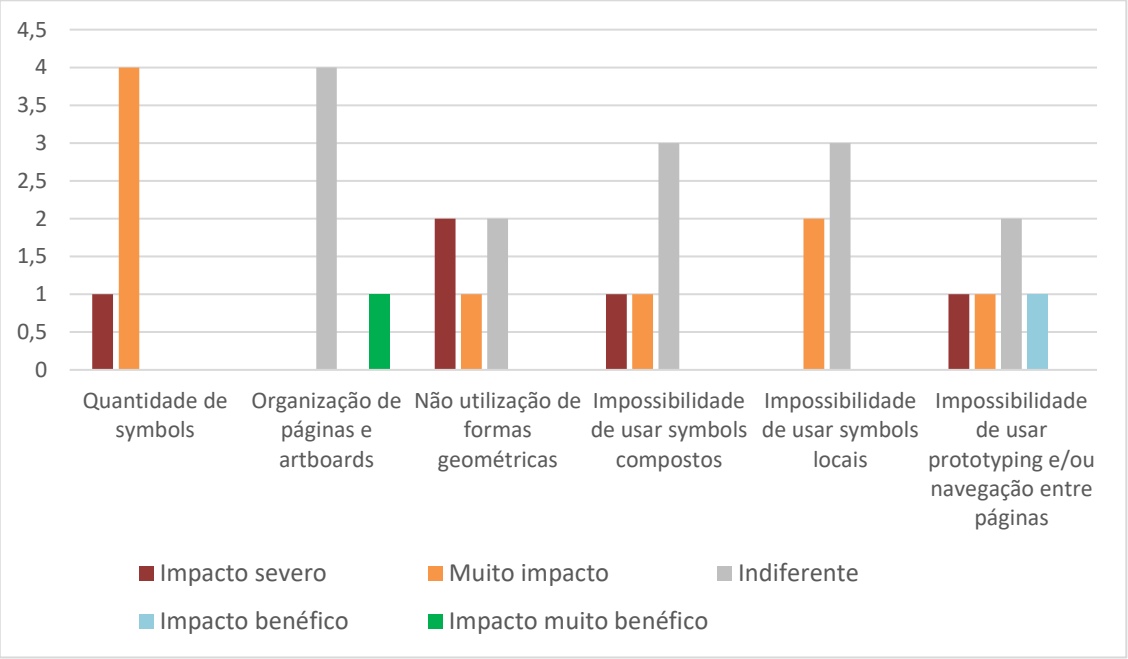


Figura 26 Indique o grau de impacto das restrições no desenvolvimento do projecto

Com base na Figura 26 a "Quantidade de *symbolsI*" apresentou bastante impacto no desenvolvimento apresentando somente opções negativas, uma com impacto severo e quatro com muito impacto. Quanto à "Organização de páginas e *artboards*" o impacto foi na sua maioria registando-se indiferença no impacto dessa restrição. De notar também um resultado muito benéfico, provavelmente proveniente do inquirido cuja experiência de trabalho é inferior a um ano. Na questão da "Não utilização de formas geométricas" as respostas tiveram um resultado negativo, apesar de disperso. Não foi consensual apresentando dois resultados de impacto severo e um resultado de muito impacto. E também dois resultados de indiferença. Este resultado é surpreendente para o autor, visto que a teoria era que o trabalho fosse realizado maioritariamente por *symbols*. Havendo uma resposta negativa nesta questão parece indicar que as formas geométricas são mais importantes do inicialmente esperado. Na questão da "Impossibilidade de usar *symbols* compostos" o resultado parece ser indiferente. Apresenta um resultado de impacto severo outro de muito impacto e três de indiferença. Aparentando não ser assim tão revelante a utilização de *symbols* compostos. É possível que pela tarefa desempenhada ser simples esta funcionalidade não fosse tão relevante como em *designs* mais complexos. Quanto à "Impossibilidade de usar *symbols* locais" também apresentou indiferença de impacto. Três resultados indiferentes e dois com muito impacto. Não é surpreendente o resultado, por estes eventualmente serem inseridos em bibliotecas, mas poderá ser algo a explorar em maior detalhe no futuro. Na questão de "Impossibilidade de usar *prototyping* e/ou navegação entre páginas apresentou resultados díspares, dois resultados indiferentes, um resultado muito severo, outro de muito impacto e outro com impacto benéfico. Assim apresenta-se como uma funcionalidade de estudo, na perspetiva de perceber dentro da mesma os pontos mais relevantes.

Na última secção do inquérito pretendia-se conhecer as funcionalidades com maior interesse por parte dos inquiridos, tendo em conta o trabalho realizado até ao momento.

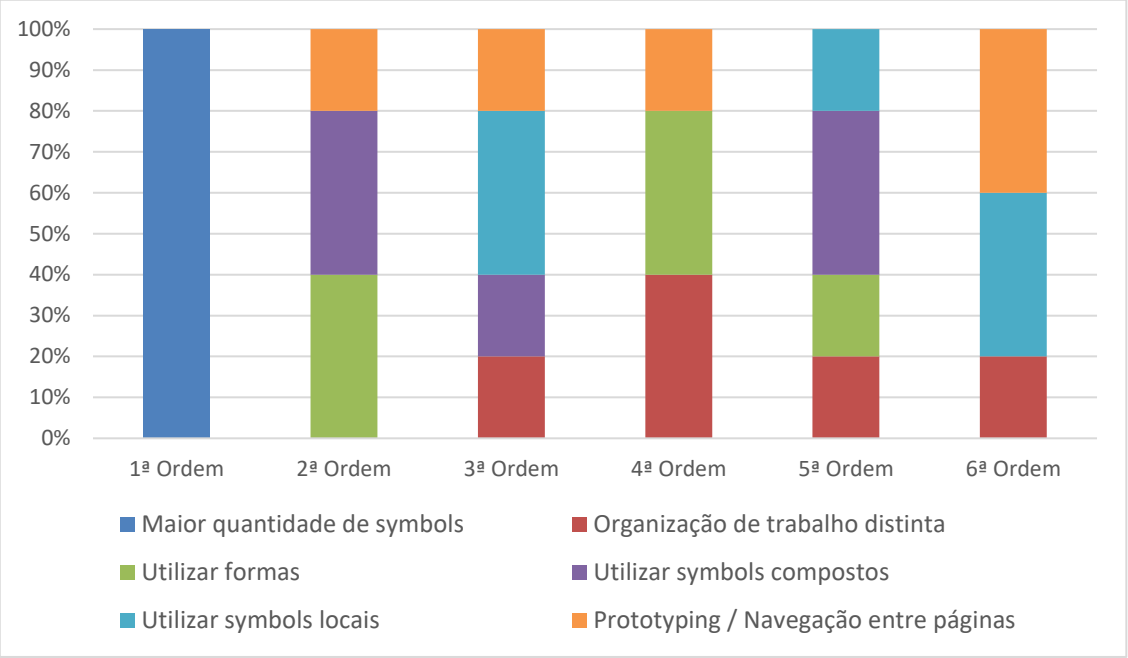


Figura 27 Indique por ordem decrescente a sua preferência de funcionalidades futuras. A primeira linha é a preferida a última é a menos relevante

Na Figura 27 é possível reparar que maior quantidade de *symbols* é a funcionalidade mais pretendida, com 100% dos inquiridos indicarem-na como a primeira a ser implementada. As restantes funcionalidades encontram-se mais dispersas. Contudo a utilização de formas aparenta ser funcionalidade seguinte a implementar. Até à 4ª ordem de preferência apresenta um total 80% na preferência dos inquiridos. Até à 5ª ordem, a utilização de *symbols* compostos destaca-se atingindo a totalidade dos inquiridos, em conjunto com a utilização de formas. As funcionalidades menos relevantes para os inquiridos neste momento aparentam ser, organização distinta de trabalho, utilização de *symbols* locais e utilização do conceito de *prototyping*. Não sendo possível identificar uma ordem de preferência nestas, pois encontram-se dispersas pelas ordens de preferência de forma idêntica. Estes resultados confirmam a tendência apresentada no impacto das restrições, na medida em que a quantidade de *symbols* é a mais preferida e a organização de trabalho distinta não tem tanto interesse.

Com o sistema contruído tentou perceber-se se este teria interesse em ser utilizado pelos *designers*.

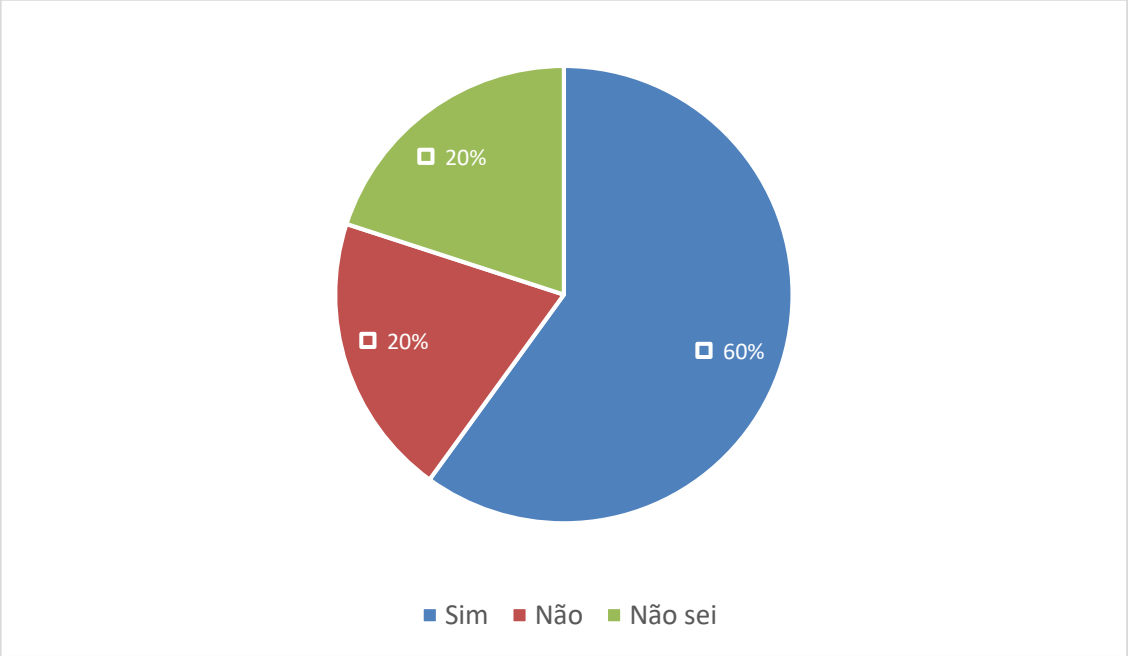


Figura 28 Utilizaria um sistema conversor de design em código no seu trabalho?

De acordo com a Figura 28 a maioria dos inquiridos utilizaria um sistema conversor de *design* em código o que indica que a continuação do projeto é viável.

Por fim tentou obter-se como este sistema seria idealmente usado pelos designers.

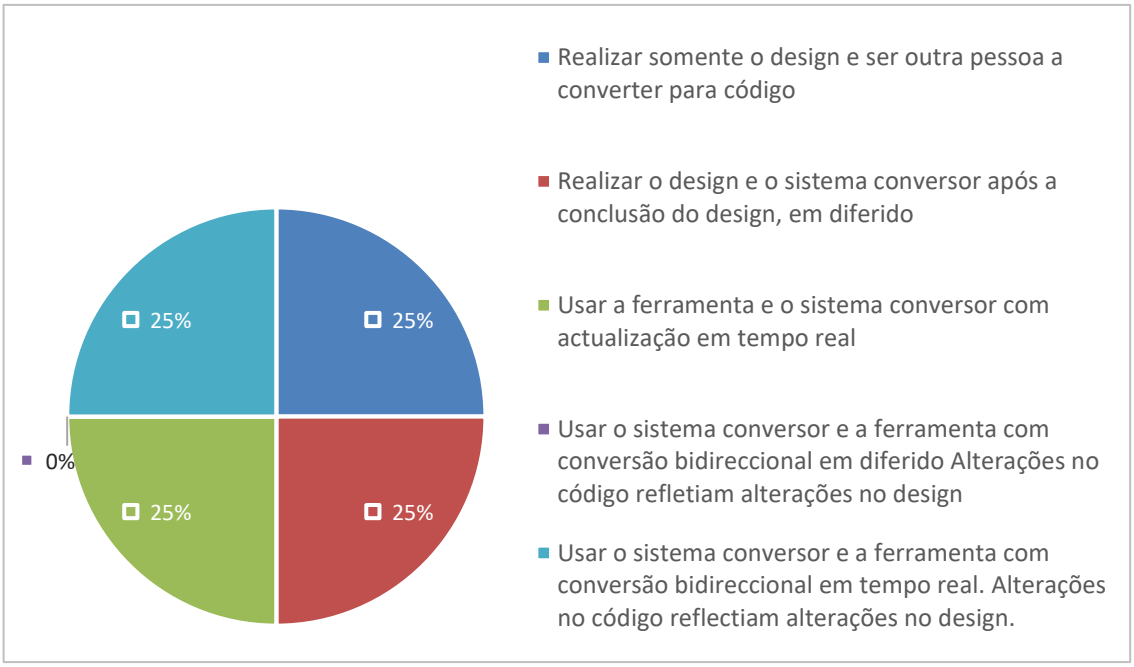


Figura 29 Caso a resposta à pergunta anterior não tenha sido não. Indique a sua preferência no uso de um sistema de conversão de design para código.

De acordo com o gráfico não é claro qual a preferência de uso do sistema, pelo que num trabalho futuro esta questão terá de ser respondida antes do desenvolvimento.

6.3.2 Dados de experiências React

O inquérito aos programadores seguiu uma lógica igual ao dos *designers*. Primeiro apresenta uma secção de caracterização dos inquiridos, uma secção de perguntas sobre a tarefa realizada e perspetivas de futuro. O inquérito e os dados estão disponíveis em anexo.

De forma a caracterizar os inquiridos pretendeu-se saber a qua experiência nas tecnologias utilizadas.

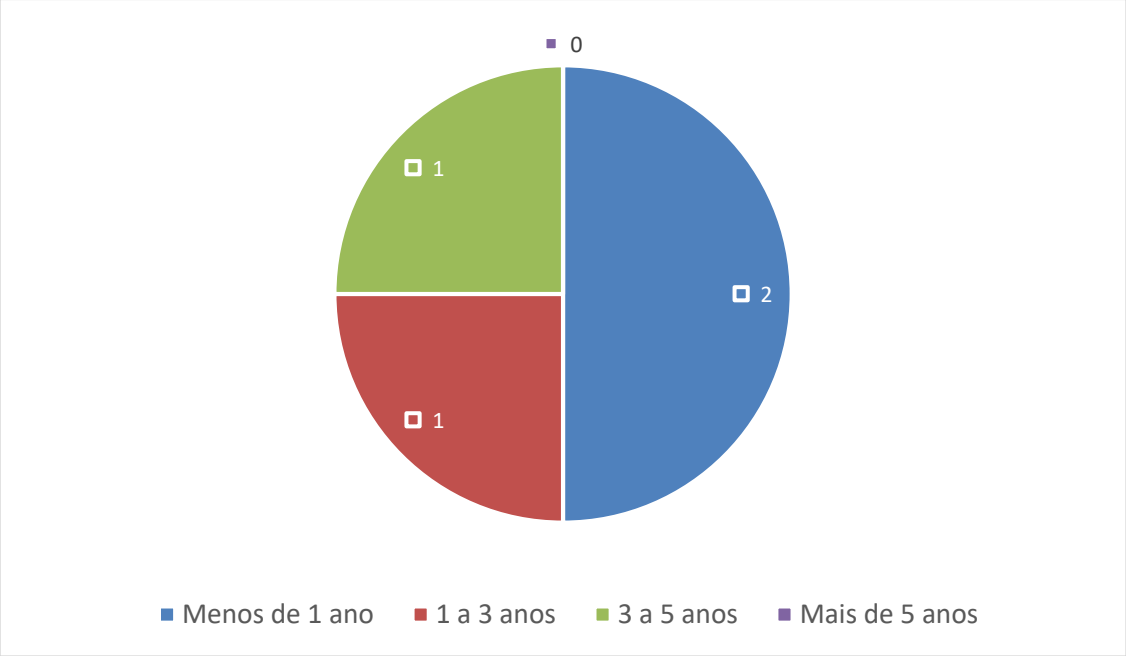


Figura 30 Qual é a sua experiência em React?

A experiência dos inquiridos em React é recente, tendo três dos inquiridos uma experiência inferior a três anos. Destaca-se um inquirido com experiência superior a três anos.

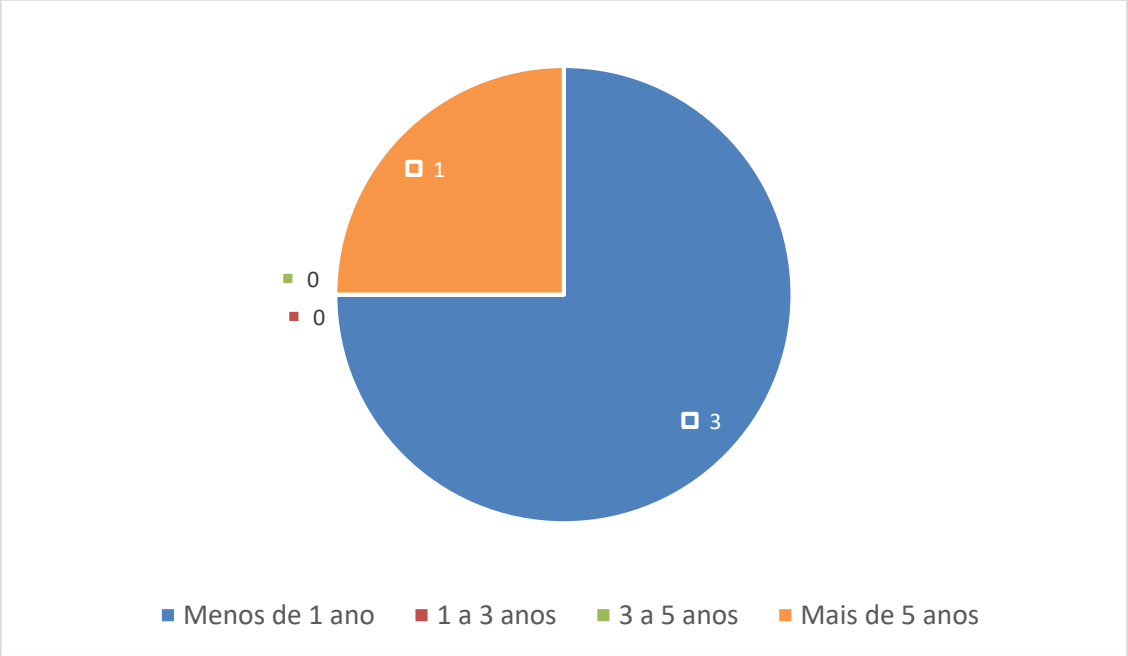


Figura 31 Qual é a sua experiência em Material-UI

Quanto à experiência em Material-UI este ainda é mais reduzida havendo três inquiridos com experiência inferior a um ano. De notar que a experiência superior a cinco anos por parte de inquirido. Este resultado deve-se provavelmente a um erro na resposta, pois sendo o Material-UI uma tecnologia dependente de ReactJs é estranho não haver inquiridos com experiência superior a cinco anos em ReactJs, mas haver em Material-UI.

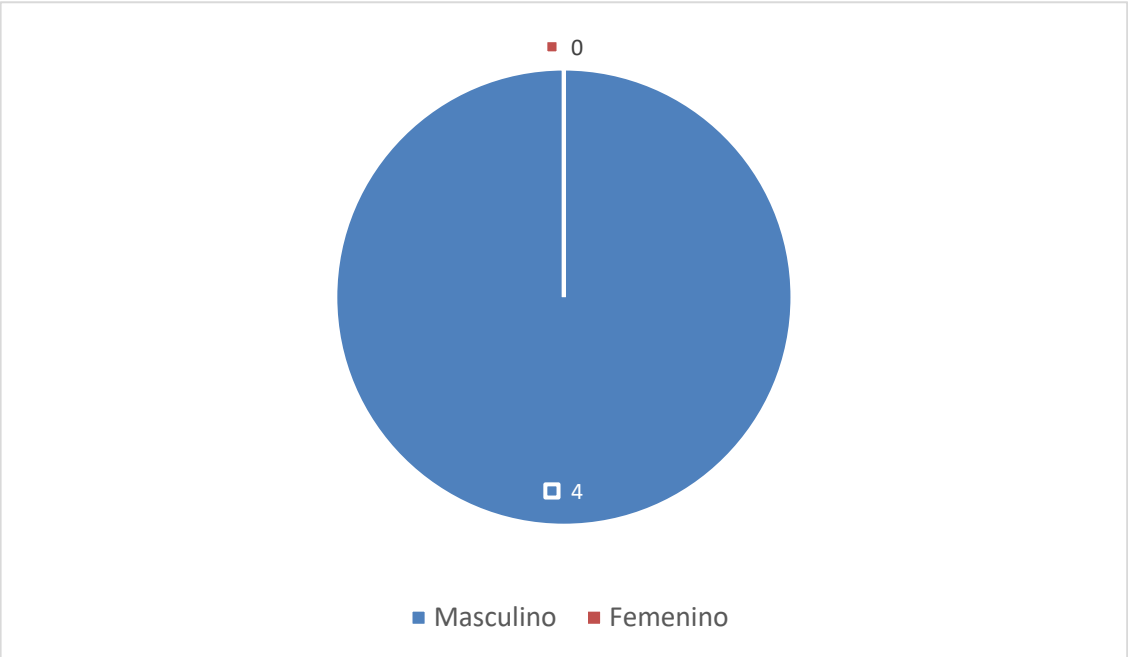


Figura 32 Qual é o seu sexo?

Quanto ao sexo dos inquiridos, todos são do sexo masculino. Assim a amostra caracteriza-se como indivíduos do sexo masculino com pouca experiência nas tecnologias utilizadas. Isto pode dever-se às tecnologias também serem recentes e ainda não tiveram de amadurecer na indústria.

De seguida tentou perceber-se o desempenho no desenvolvimento da tarefa. Inicialmente a compreensão do enunciado e posteriormente a opinião quanto às funcionalidades da solução.

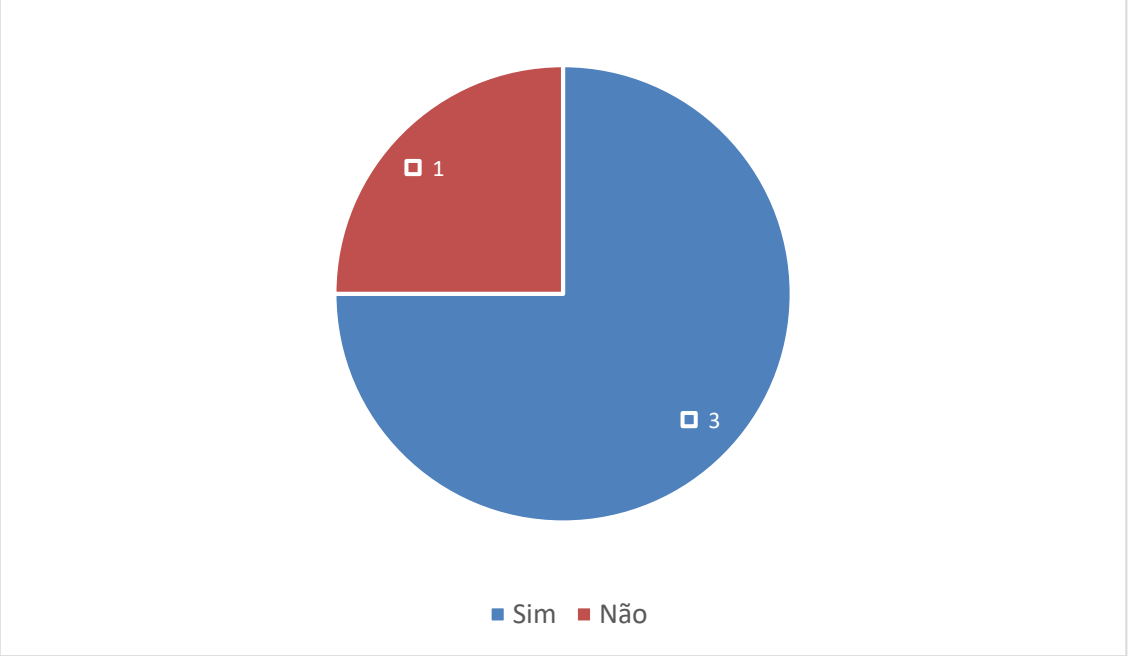


Figura 33 O enunciado foi suficientemente claro no pretendido?

Na compreensão do enunciado houve um inquirido que inicialmente não compreendeu a tarefa a realizar. Contudo durante a realização da tarefa foi devidamente explicado o pretendido tendo o inquirido ficado devidamente esclarecido.

Para o programador também foi questionada se a organização dos ficheiros e pastas era semelhante à forma deste trabalhar.

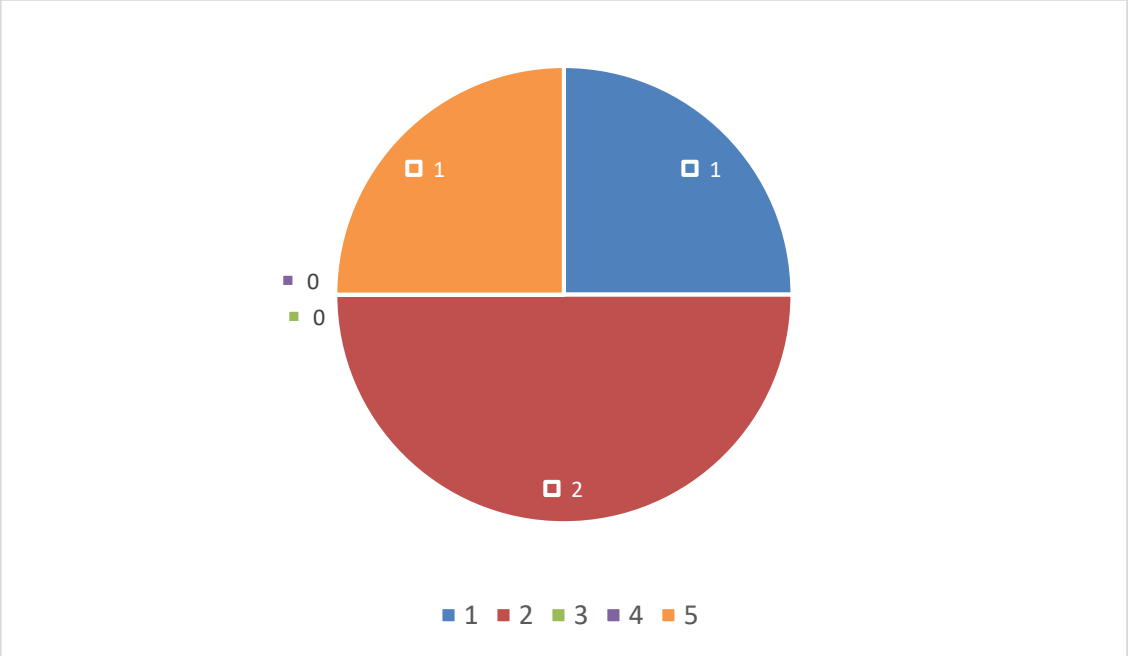


Figura 34 Classifique de 1 a 5, sendo 1 Totalmente Distinta e 5 Bastante Semelhante, se a organização de pastas e ficheiros é semelhante à sua forma habitual de trabalho.

Das respostas dos inquiridos percebe-se que a organização que os inquiridos são ao seu código é diferente do que a ferramenta suporta. Mais de metade dos inquiridos teve uma resposta negativa.

Como a ferramenta de conversão traduz *design* para código era importante saber o grau de semelhança com o *design* providenciado.

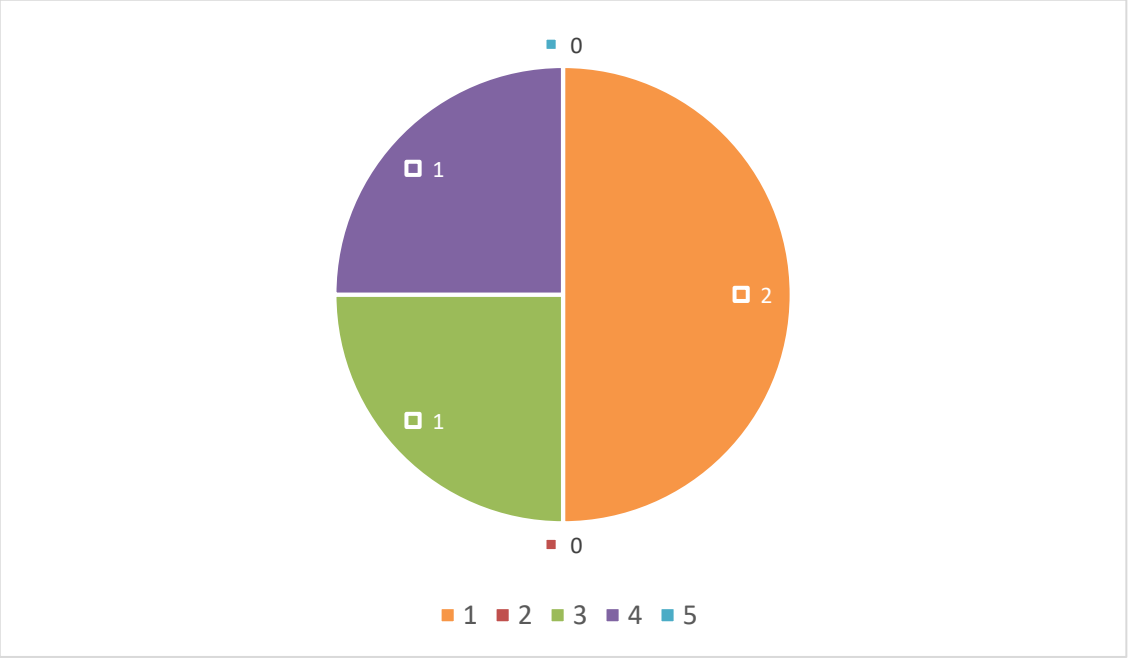


Figura 35 Classifique de 1 a 5, sendo 1 Totalmente Distinto e 5 Bastante Semelhante, a semelhança entre o código inicial e o design esperado

Neste caso o código inicialmente gerado era consideravelmente diferente do *design* a replicar. Havendo duas respostas como sendo totalmente diferente, uma indiferente e uma ligeiramente semelhante. Os programadores não receberam o mesmo design, tendo sido fornecidos dois designs diferentes igualmente distribuídos. Nesse sentido pode ter acontecido que as respostas mais negativas sejam referentes a um design e as mais positivas a outro.

A tarefa dos programadores era corrigir os defeitos do código por isso foi-lhes questionada a quantidade de ajuste que estes tiveram de realizar.

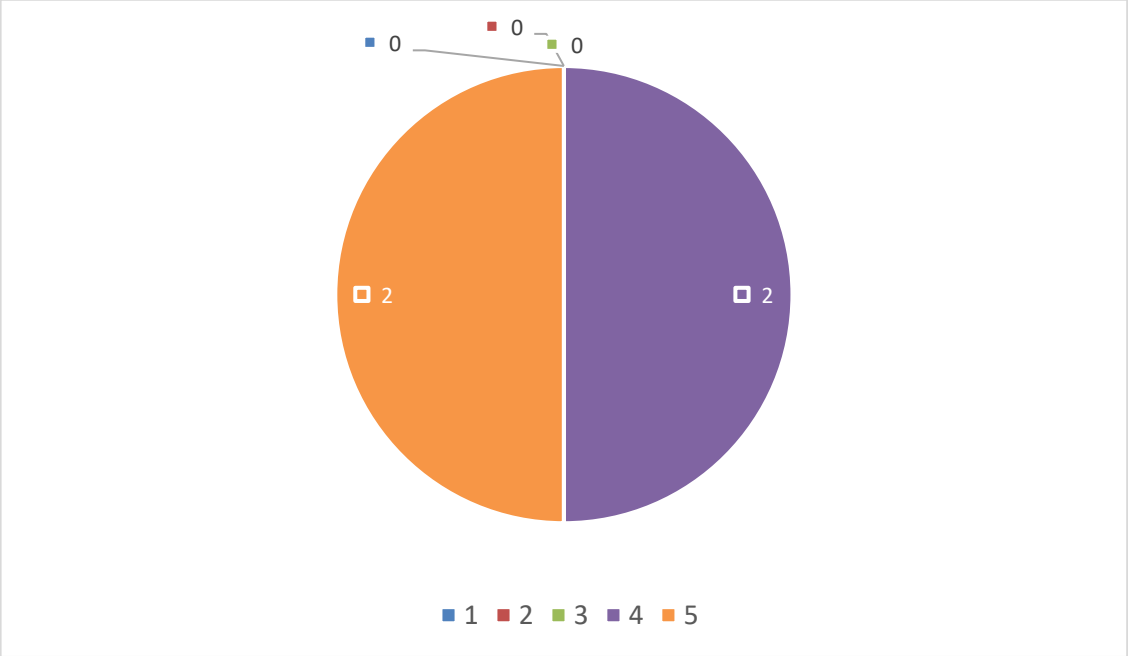


Figura 36 Classifique de 1 a 5, sendo 1 Pouco Ajuste e 5 Muito Ajuste, o ajuste necessário para fazer corresponder o código ao design

De acordo com os inquiridos a quantidade de ajuste foi elevada, não havendo respostas inferiores a quatro valores. Não pode ser considerado uma surpresa tendo em conta a pouca semelhança entre o código gerado inicialmente para o *design*.

Como objetivo final da tarefa era perceber se o sistema tinha reduzido a quantidade de trabalho ao programador.

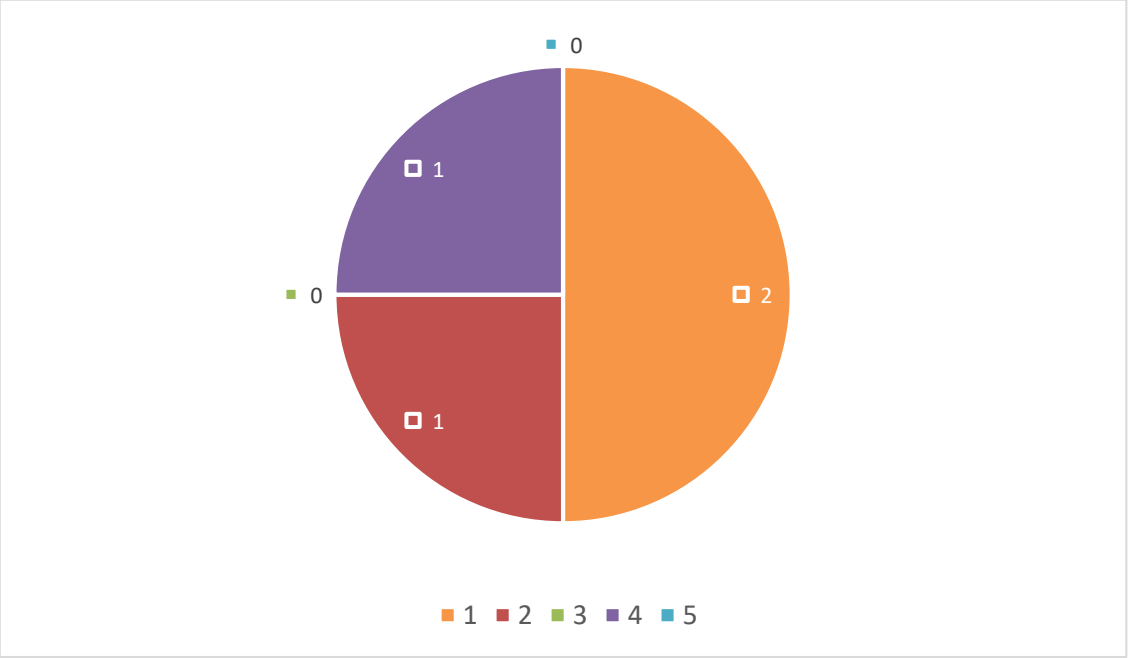


Figura 37 Classifique de 1 a 5, sendo 1 Pouca Redução e 5 Bastante Redução, a redução do trabalho através do uso da ferramenta

As respostas aparentam indicar que o a redução de trabalho não foi significativa, tendo três dos inquiridos respondido com resposta negativa. Contudo aparentemente um dos inquiridos sentiu o seu trabalho reduzido. O conjunto de respostas da secção indica a generalidade dos inquiridos não achou que a ferramenta o tivesse ajudado significativamente e que este resolvesse o problema pretendido.

A última secção foi dedicada à preferência de desenvolvimentos futuros e modo de utilização da ferramenta. Assim foi pedido para organizar a preferência de funcionalidades futuras.

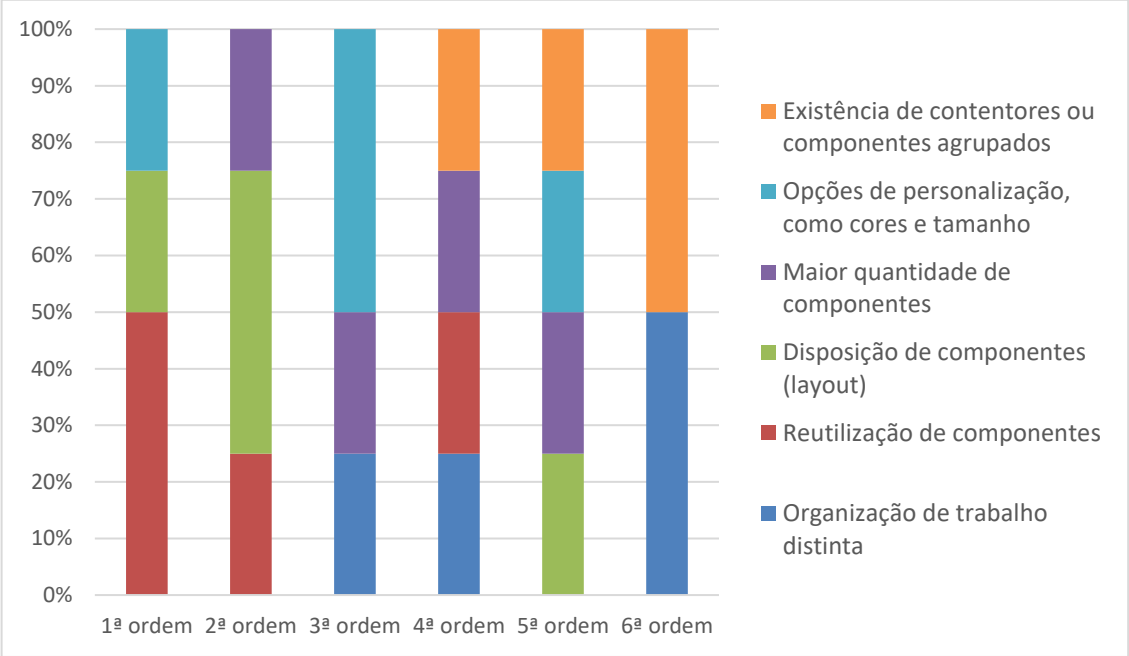


Figura 38 Indique por ordem decrescente a sua preferência de funcionalidades futuras. A primeira linha é a preferida e a última a menos relevante.

Na Figura 38 a reutilização e disposição de componentes parecem ser as funcionalidades com maior preferência, por até à segunda ordem possuírem a preferência de 80% dos inquiridos. Como o trabalho do programador recorre a reutilização de código de forma a reduzir o seu trabalho, é natural que este pretenda que os componentes possam ser reutilizados. A disposição de elementos na página também é uma parte crítica do seu trabalho, por isso não é surpreendente que esteja no topo de preferências. As opções de personalização seguem-se na ordem de preferências, totalizando também 80% até à terceira ordem. A personalização seria normalmente o passo a seguir após a disposição de componentes, no desenvolvimento de interfaces. Assim este dado vem em linha com o processo realizado pelo programador. A maior quantidade de componentes segue-se, totalizando 80% na quinta ordem. Como os componentes provêm de uma biblioteca, inseri-los no código é simples. Contudo colocá-los no local correto é mais complexo. Por isso é que a existência de componente não é tão significativa para os programadores. Por fim, a existência de uma organização de trabalho distinta e de contentores ou componentes agrupados são as funcionalidades com menor interesse para o programador, provavelmente por terem grande impacto no desenvolvimento da UI.

Mesmo com as limitações é importante saber se o sistema poderia ser utilizado no estado atual.

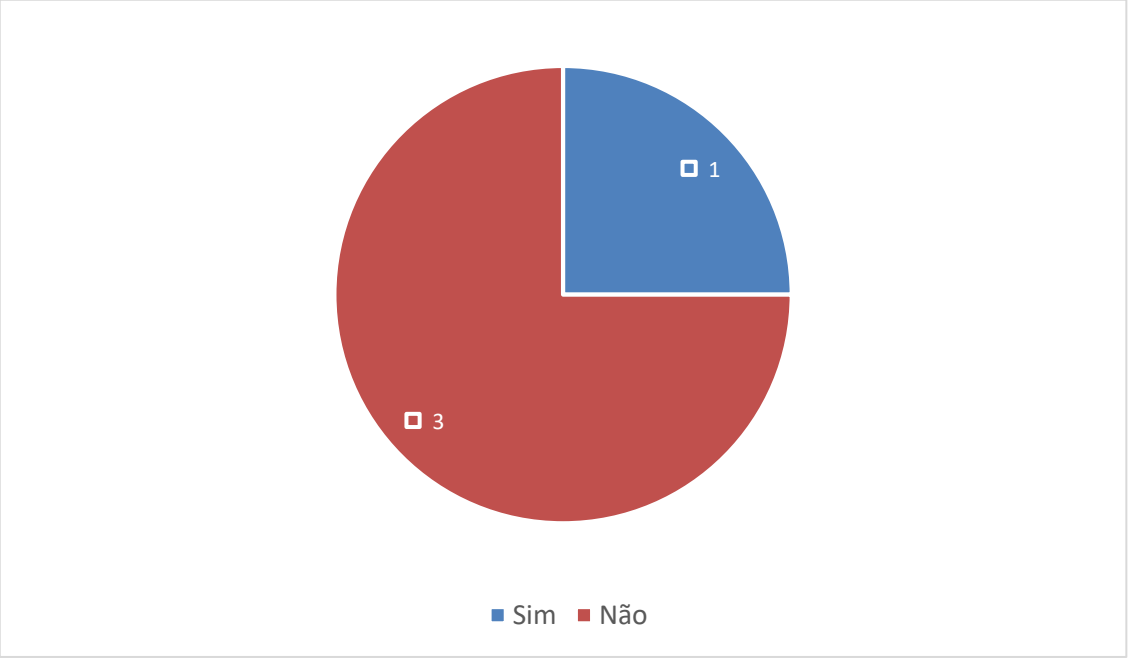


Figura 39 No estado actual utilizaria o sistema conversor?

A resposta dos programadores inquiridos foi maioritariamente negativa, havendo três inquiridos a responder que não, e um a indicar que sim. Face às limitações da tecnologia o resultado não é inesperado.

Contudo pelo sistema atual não ajudar significativamente os programadores, nada indica que estes não pretendam ter o apoio de um sistema semelhante.

Figura 40 Utilizaria um sistema conversor de design em código no seu trabalho

Os dados de uma possível utilização de um sistema conversor não são claros. Havendo um inquirido que respondeu que utilizaria e outro que não. Contudo ao haver dois resultados “não sei” aparentam indicar que com melhorias ao sistema, este poderia ser utilizado.

No caso de ser utilizado seria importante saber qual o modo de utilização.

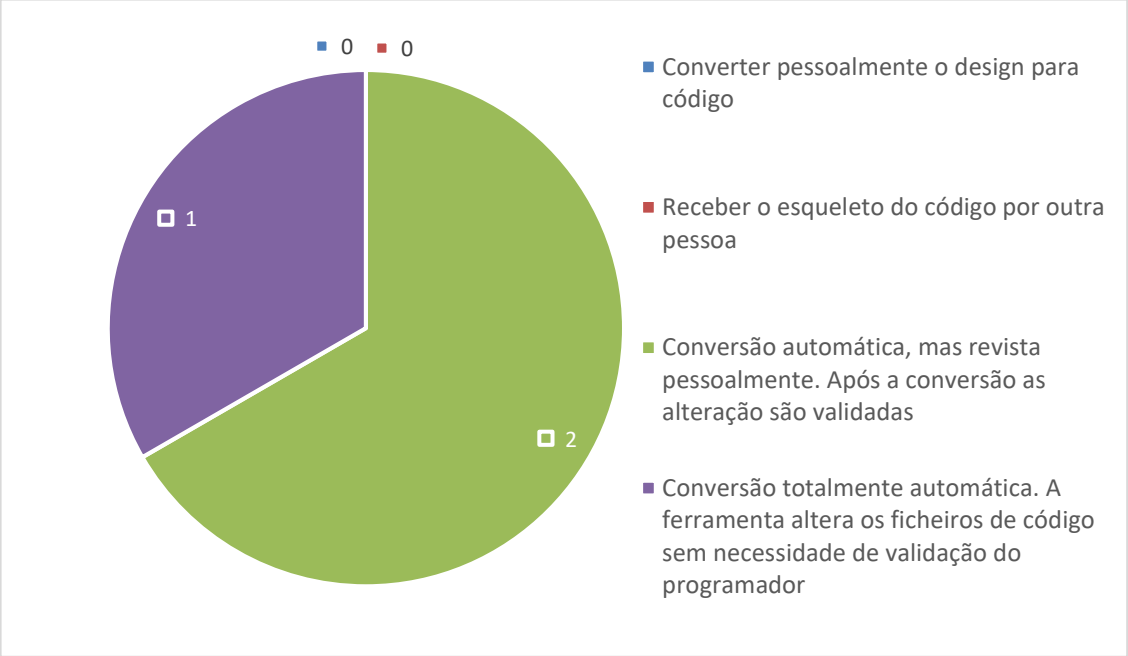


Figura 41 Caso a resposta à pergunta anterior tenha sido "Sim" ou "Não sei". Indique a sua preferência no uso de um sistema conversor de design para código

Os dados aparentam indicar uma conversão automática, contudo este não deverá ser totalmente automática. No limite poderá optar-se por suportar as duas utilizações, com possibilidade de escolha em qual usar.

7 Conclusão

O objetivo desta tese era conseguir reduzir o tempo gasto no desenvolvimento através de uma ferramenta que simplificasse o processo. Investigaram-se padrões de *design* que tornavam os mesmos semelhantes entre *designers*. Assim como aplicações de *design* utilizavam estes padrões. Foi interpretado o ficheiro da aplicação Sketch, escolhida como tecnologia desta tese, e o mapeamento entre os conceitos de *design* e componentes de código ReactJs. Este conhecimento foi integrado num sistema que converte *designs* realizados no Sketch para código ReactJs. O sistema foi avaliado por *designers* e programadores, quanto à redução de trabalho e simplicidade de uso. Devido às restrições ao trabalho do *designer* e da incapacidade de reutilização do código, o objetivo não pode ser considerado como concluído. Isto deveu-se à falta de acesso direto aos recursos, nomeadamente à aplicação Sketch. Na qual foi necessário pedir várias vezes pequenos projetos Sketch a um *designer* de forma a poder entender melhor a arquitetura do ficheiro e como esta organizava o conteúdo das bibliotecas. Esta dependência externa aumentou o tempo de compreensão da ferramenta. Contudo no desenvolvimento desta tese houve um trabalho de pesquisa realizado que abre caminho a desenvolvimento futuro. Nomeadamente o aumento de componentes de *design*, a disposição de elementos e as opções de personalização. Após o desenvolvimento destas funcionalidades deve ser novamente testado com *designers* e programadores de forma a averiguar se os resultados são os mesmos. Também se pode melhorar o próprio funcionamento do sistema, optando por uma base de dados comum ao *design* e ao código. E também adição de interação com a página, mesmo que simplificada.

Referências

Adobe XD (no date). Available at: https://www.adobe.com/products/xd.html?promoid=PYPVQ3HN&mv=other (Accessed: 28 January 2020).

Adobe XD Features (no date). Available at: https://www.adobe.com/products/xd/details.html (Accessed: 27 December 2019).

Bajammal, M., Mazinanian, D. and Mesbah, A. (2018) 'Generating Reusable Web Components from Mockups', in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery (ASE 2018), pp. 601–611. doi: 10.1145/3238147.3238194.

Borza, J. S. (no date) 'FAST Diagrams: The Foundation for Creating Effective Function Models', p. 10.

Chang, K. S.-P. and Myers, B. A. (2014) 'Creating interactive web data applications with spreadsheets', in *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. *the 27th annual ACM symposium*, Honolulu, Hawaii, USA: ACM Press, pp. 87–96. doi: 10.1145/2642918.2647371.

Chen, C., Su, T., Meng, G., Xing, Z. and Liu, Y. (2018) 'From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation', in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 665–676. doi: 10.1145/3180155.3180240.

Denuzière, L., Rodriguez, E. and Granicz, A. (2013) 'Piglets to the Rescue: Declarative User Interface Specification with Pluggable View Models', in *Proceedings of the 25th Symposium on Implementation and Application of Functional Languages*. New York, NY, USA: Association for Computing Machinery (IFL '13), pp. 105–115. doi: 10.1145/2620678.2620689.

Design kits for AdobeXD (no date). Available at: https://www.adobe.com/products/xd/resources.html?promoid=WXYGJ27F&mv=other#panel-3 (Accessed: 22 February 2020).

Human Interface Guidelines (no date). Available at: https://developer.apple.com/design/human-interface-guidelines/ (Accessed: 8 February 2020).

Introducing JSX – React (no date). Available at: https://reactjs.org/docs/introducing-jsx.html (Accessed: 14 June 2020).

Leiva, G., Maudet, N., Mackay, W. and Beaudouin-Lafon, M. (2019) 'Enact: Reducing Designer–Developer Breakdowns When Prototyping Custom Interactions', *ACM Trans. Comput.-Hum. Interact.*, 26(3). doi: 10.1145/3310276.

Material Design (no date) *Material Design*. Available at: https://material.io/design/introduction/ (Accessed: 8 February 2020).

Material Design design kits (no date) *Material Design*. Available at: https://material.io/resources/ (Accessed: 22 February 2020).

Material-UI (no date). Available at: https://material-ui.com/ (Accessed: 13 June 2020).

Meskens, J., Luyten, K. and Coninx, K. (2009) 'Plug-and-design: embracing mobile devices as part of the design environment', in *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems - EICS '09*. *the 1st ACM SIGCHI symposium*, Pittsburgh, PA, USA: ACM Press, p. 149. doi: 10.1145/1570433.1570461.

Meskens, J., Luyten, K. and Coninx, K. (2010) 'Jelly: a multi-device design environment for managing consistency across devices', in *Proceedings of the International Conference on Advanced Visual Interfaces - AVI '10*. *the International Conference*, Roma, Italy: ACM Press, p. 289. doi: 10.1145/1842993.1843044.

Outsystems UI (2019) *OutSystems*. Available at: https://success.outsystems.com/Documentation/11/Getting_Started/Create_Your_First_Reactive_Web_App (Accessed: 16 February 2020).

OutSystems UI Framework (no date). Available at: https://outsystemsui.outsystems.com/OutSystemsUIWebsite/HowItWorks (Accessed: 9 November 2019).

Rodriguez-Martinez, L., Duran-Limon, H., Mendoza-González, R. and Muñoz, J. (2015) 'Identifying common activities in the graphical user interface development process and their integration into the software-system development life cycle', *Computer Science and Information Systems*, 12(1), pp. 323–348. doi: 10.2298/CSIS140301002R.

Seichter, H., Looser, J. and Billinghurst, M. (2008) 'ComposAR: An intuitive tool for authoring AR applications', in *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pp. 177–178. doi: 10.1109/ISMAR.2008.4637354.

Sketch (no date) *Sketch*. Available at: https://www.sketch.com/ (Accessed: 28 January 2020).

Sketch Code Export (no date) *Sketch*. Available at: https://www.sketch.com/docs/exporting/code-export/ (Accessed: 9 November 2019).

Sketch File Format (no date) *Sketch Developers*. Available at: https://developer.sketch.com/file-format/ (Accessed: 14 June 2020).

Sketch Interface (no date). Available at: https://www.sketch.com/docs/the-interface/ (Accessed: 29 January 2020).

Sketch2React (no date). Available at: https://sketch2react.gitbook.io/sketch2react-io/learn/quick-guide (Accessed: 8 February 2020).

SketchAPI (no date) *Sketch Developers*. Available at: https://developer.sketch.com/reference/api/ (Accessed: 23 August 2020).

Sopin, I. and Hamza-Lup, F. G. (2010) 'Extending the Web3D: design of conventional GUI libraries in X3D', in *Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10*. *the 15th International Conference*, Los Angeles, California: ACM Press, p. 137. doi: 10.1145/1836049.1836070.

SwiftUI (no date). Available at: https://developer.apple.com/xcode/swiftui/ (Accessed: 2 February 2020).

Tassinari, O. (2020) *Material-UI for Sketch*, *Medium*. Available at: https://medium.com/material-ui/introducing-material-ui-for-sketch-b72f5f79f7f0 (Accessed: 14 June 2020).

Tidwell, J. (2006) *Designing Interfaces*. First Edition. O'Reilly Media Inc.

UI Fabric (no date). Available at: https://developer.microsoft.com/en-us/fabric#/ (Accessed: 8 February 2020).

Usage - Material-UI (no date). Available at: https://material-ui.com/getting-started/usage/ (Accessed: 14 June 2020).

Wang, J., Jing, F., Xia, Y., Xu, C. and Gao, X. (2019) 'Research on Definition of Digital Mockup and Application of Zoning Optimization', in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 2056–2059. doi: 10.1109/IAEAC47372.2019.8998036.

WWDC 2019 (2019). Steve Jobs Theater, Cupertino, Califórnia, EUA. Available at: https://youtu.be/psL_5RIBqnY?t=7601 (Accessed: 4 November 2019).

Zarras, A. V., Mamalis, G., Papamichail, A., Kollias, P. and Vassiliadis, P. (2018) 'And the Tool Created a GUI That was Impure and Without Form: Anti-Patterns in Automatically Generated GUIs', in *Proceedings of the 23rd European Conference on Pattern Languages of Programs - EuroPLoP '18*. *the 23rd European Conference*, Irsee, Germany: ACM Press, pp. 1–8. doi: 10.1145/3282308.3282333.

Zeidler, C., Lutteroth, C., Sturzlinger, W. and Weber, G. (2013) 'The auckland layout editor: an improved GUI layout specification process', in *Proceedings of the 26th annual ACM symposium on User interface software and technology - UIST '13*. *the 26th annual ACM symposium*, St. Andrews, Scotland, United Kingdom: ACM Press, pp. 343–352. doi: 10.1145/2501988.2502007.

8 Anexos

8.1 Sessão de experimentação

8.1.1 Instruções ao *designer*

Guia de desenvolvimento Sketch

O presente documento serve como um guia para o a fase de testes com o utilizador, no contexto da tese de mestrado em Engenharia Informática de Mário João Gonçalves Ferreiro no Instituto Superior de Engenharia do Porto, cujo tema é: Construtor de Interfaces Web. O objetivo da tese é converter *design* produzido na ferramenta Sketch para código React, correspondente a esse design, através de uma ferramenta desenvolvida pelo autor. A fase de testes encontra-se dividida em dois, uma primeira parte em que é desenvolvido o *design* na ferramenta Sketch, e uma segunda onde o código gerado pela ferramenta conversora é fornecido a programadores para eventuais correções necessárias.

Cada uma das partes está sujeita a restrições definidas na secção com o mesmo nome, no presente documento.

No final da resolução da tarefa deste guia deverá ser respondido ao inquérito disponível em:

https://forms.office.com/Pages/ResponsePage.aspx?id=KaZgeh_mDEOcYyZmU7K4mdwvCb8oQhZAmDzerMJodbNUNThPRDdPRlQwRjBLVjRWVDA5U0hYTk5RMy4u

O tempo total previsto é de 1 hora: 45 minutos para a tarefa, 15 para o inquérito.

Restrições

O sistema de conversão de design Sketch para código desenvolvido assenta no uso *symbols* de uma biblioteca específica. A biblioteca fornecida em conjunto com o guião e cujo nome é Blocos.sketch. Esta biblioteca apresenta os seguintes *symbols*:

- Botão
- Campo de texto
- Caixa de seleção
- Chips
- Cartão

O desenvolvimento do projeto deve seguir as seguintes restrições:

1. O projeto Blocos.sketch terá de ser adicionado como biblioteca externa ao projeto.
2. Só poderão ser utilizados os *symbols* disponíveis pela biblioteca, e texto livre, qualquer outro conteúdo de *design* será ignorado.
3. Não deverá haver composição de *symbols*, com isto denomina-se situações em que um *symbol* possa estar contido noutro.
4. Não deverão ser criados *symbols* locais.
5. Opções de personalização como cor e tamanho são desejadas, mas não prioritárias.
6. *Prototyping* e/ou navegação entre páginas não são necessárias.
7. Variações da mesma página não são necessárias.
8. Componentes que aparecem em resposta a ações devem ser apresentados como estando sempre presentes.

O sistema usa uma lógica de organização do design em Sketch na medida em que cada *artboard* corresponde a uma página web e cada página Sketch corresponde a pastas de forma a organizar os ficheiros de código. Esta estrutura terá de ser respeitada no desenvolvimento da tarefa.

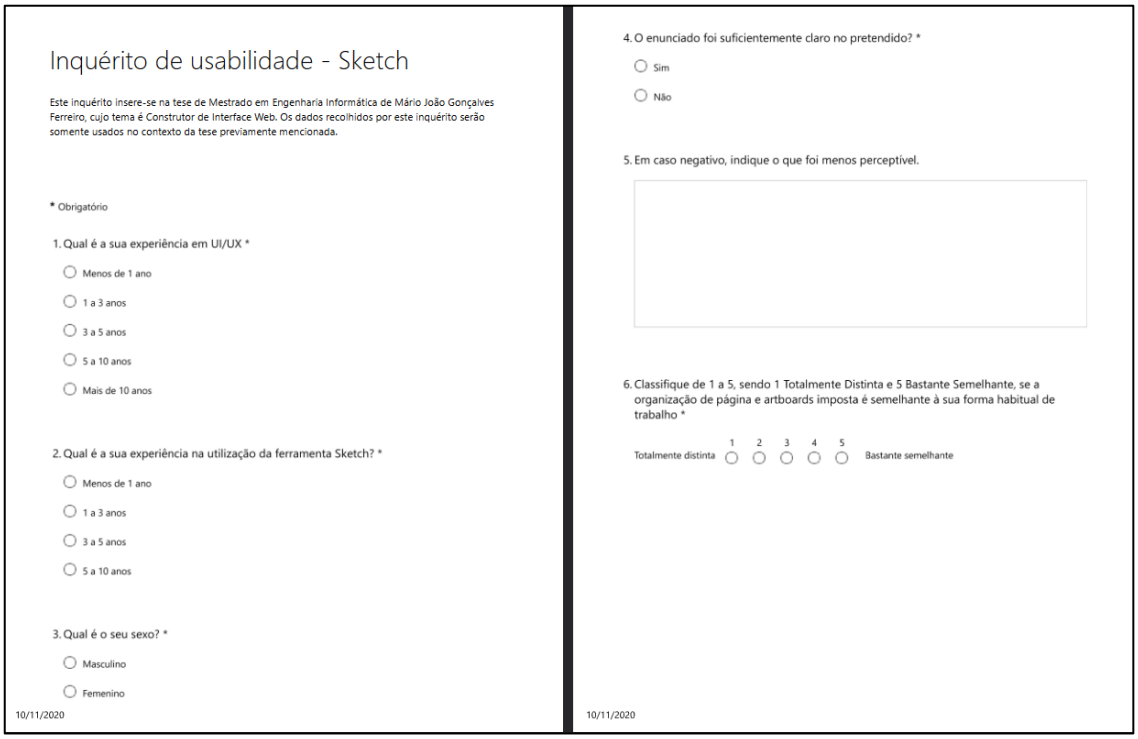
Enunciado

Neste projeto pretende-se desenvolver o design para uma aplicação web muito básica de registo de prática de desporto. Esta aplicação deve conter no mínimo três páginas distintas. Uma página inicial da aplicação indicativa do objetivo da mesma, registo de atividade desportiva. Uma ou mais páginas de recolha de informação sobre o utilizador como: como nome, idade, sexo e os possíveis desportos a selecionar (entre 5 a 10). E uma página final com o resumo do registo, o nome do utilizador e os desportos selecionados.

Publicação

Quando o *design* tiver sido concluído o ficheiro do projeto Sketch deverá ser enviado para o remetente. Assim como a exportação dos *artboards* para PDF.

8.1.2 Inquérito ao *designer*



Inquérito de usabilidade - Sketch

Este inquérito insere-se na tese de Mestrado em Engenharia Informática de Mário João Gonçalves Ferreiro, cujo tema é Construtor de Interface Web. Os dados recolhidos por este inquérito serão somente usados no contexto da tese previamente mencionada.

* Obrigatório

1. Qual é a sua experiência em UI/UX *
   - Menos de 1 ano
   - 1 a 3 anos
   - 3 a 5 anos
   - 5 a 10 anos
   - Mais de 10 anos

2. Qual é a sua experiência na utilização da ferramenta Sketch? *
   - Menos de 1 ano
   - 1 a 3 anos
   - 3 a 5 anos
   - 5 a 10 anos

3. Qual é o seu sexo? *
   - Masculino
   - Femenino

10/11/2020

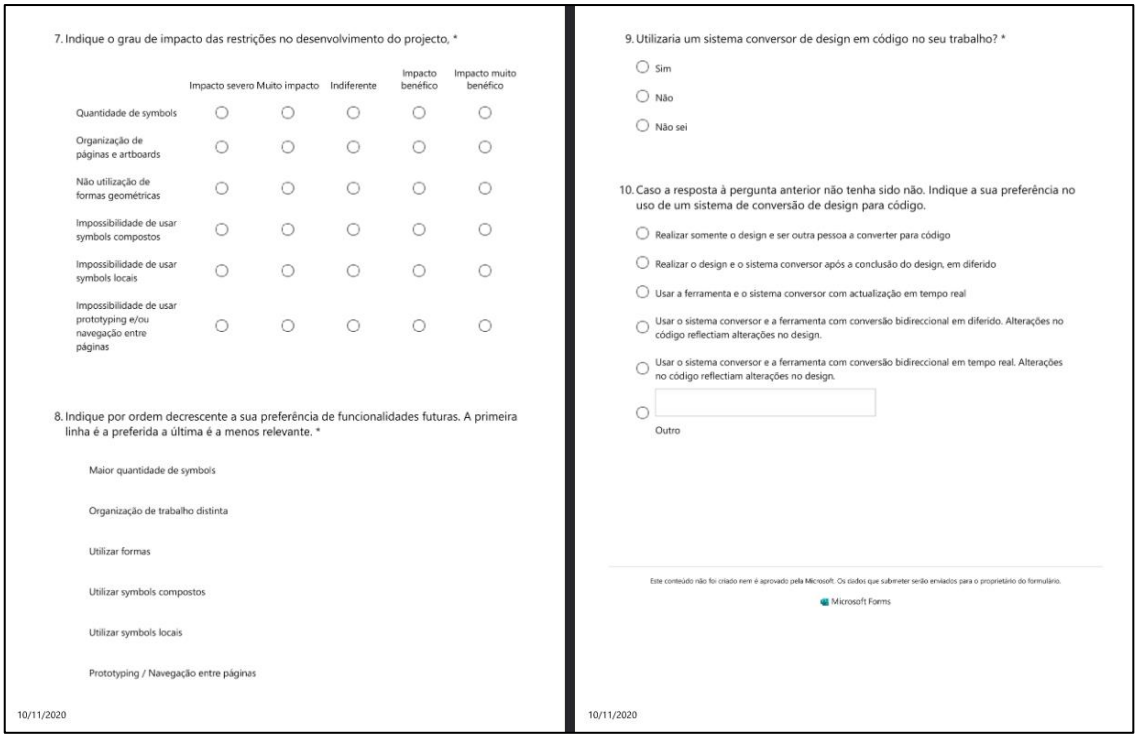
4. O enunciado foi suficientemente claro no pretendido? *
   - Sim
   - Não

5. Em caso negativo, indique o que foi menos perceptível.

6. Classifique de 1 a 5, sendo 1 Totalmente Distinta e 5 Bastante Semelhante, se a organização de página e artboards imposta é semelhante à sua forma habitual de trabalho *

	1	2	3	4	5	
Totalmente distinta	○	○	○	○	○	Bastante semelhante

10/11/2020



7. Indique o grau de impacto das restrições no desenvolvimento do projecto, *

	Impacto severo	Muito impacto	Indiferente	Impacto benéfico	Impacto muito benéfico
Quantidade de symbols	○	○	○	○	○
Organização de páginas e artboards	○	○	○	○	○
Não utilização de formas geométricas	○	○	○	○	○
Impossibilidade de usar symbols compostos	○	○	○	○	○
Impossibilidade de usar symbols locais	○	○	○	○	○
Impossibilidade de usar prototyping e/ou navegação entre páginas	○	○	○	○	○

8. Indique por ordem decrescente a sua preferência de funcionalidades futuras. A primeira linha é a preferida a última é a menos relevante. *

Maior quantidade de symbols

Organização de trabalho distinta

Utilizar formas

Utilizar symbols compostos

Utilizar symbols locais

Prototyping / Navegação entre páginas

10/11/2020

9. Utilizaria um sistema conversor de design em código no seu trabalho? *
   - Sim
   - Não
   - Não sei

10. Caso a resposta à pergunta anterior não tenha sido não. Indique a sua preferência no uso de um sistema de conversão de design para código.
   - Realizar somente o design e ser outra pessoa a converter para código
   - Realizar o design e o sistema conversor após a conclusão do design, em diferido
   - Usar a ferramenta e o sistema conversor com actualização em tempo real
   - Usar o sistema conversor e a ferramenta com conversão bidireccional em diferido. Alterações no código reflectiam alterações no design.
   - Usar o sistema conversor e a ferramenta com conversão bidireccional em tempo real. Alterações no código reflectiam alterações no design.
   - ○ Outro

Este conteúdo não foi criado nem é aprovado pela Microsoft. Os dados que submeter serão enviados para o proprietário do formulário.

Microsoft Forms

10/11/2020

8.1.3 Dados do inquérito ao *designer*

Qual é a sua experiência em UI/UX	Qual é a sua experiência na utilização da ferramenta Sketch?	Qual é o seu sexo?	O enunciado foi suficientemente claro no pretendido?	Em caso negativo, indique o que foi menos perceptível.
Mais de 10 anos	1 a 3 anos	Feminino	Sim	
1 a 3 anos	Menos de 1 ano	Feminino	Sim	
Mais de 10 anos	3 a 5 anos	Masculino	Sim	
Menos de 1 ano	Menos de 1 ano	Feminino	Sim	
3 a 5 anos	3 a 5 anos	Feminino	Sim	

Classifique de 1 a 5, sendo 1 Totalmente Distinta e 5 Bastante Semelhante, se a organização de página e artboards imposta é semelhante à sua forma habitual de trabalho	Quantidade de symbols
5	Muito impacto
4	Muito impacto
4	Muito impacto
4	Muito impacto
5	Impacto severo

Organização de páginas e artboards	Não utilização de formas geométricas	Impossibilidade de usar symbols compostos	Impossibilidade de usar symbols locais
Impacto muito benéfico	Indiferente	Indiferente	Indiferente
Indiferente	Impacto severo	Indiferente	Muito impacto
Indiferente	Impacto severo	Indiferente	Indiferente
Indiferente	Muito impacto	Impacto severo	Indiferente
Indiferente	Indiferente	Muito impacto	Muito impacto

Impossibilidade de usar prototyping e/ou navegação entre páginas	Indique por ordem decrescente a sua preferência de funcionalidades futuras. A primeira linha é a preferida a última é a menos relevante.	Utilizaria um sistema conversor de design em código no seu trabalho?
Indiferente	Maior quantidade de symbols; Prototyping / Navegação entre páginas; Organização de trabalho distinta; Utilizar formas; Utilizar symbols compostos; Utilizar symbols locais;	Sim
Impacto severo	Maior quantidade de symbols; Utilizar formas; Utilizar symbols locais; Prototyping / Navegação entre páginas; Utilizar symbols compostos; Organização de trabalho distinta;	Sim
Impacto benéfico	Maior quantidade de symbols; Utilizar formas; Utilizar symbols compostos;	Não sei

	Organização de trabalho distinta; Utilizar symbols locais; Prototyping / Navegação entre páginas;	
Muito impacto	Maior quantidade de symbols; Utilizar symbols compostos; Prototyping / Navegação entre páginas; Utilizar formas; Organização de trabalho distinta; Utilizar symbols locais;	Não
Indiferente	Maior quantidade de symbols; Utilizar symbols compostos; Utilizar symbols locais; Organização de trabalho distinta; Utilizar formas; Prototyping / Navegação entre páginas;	Sim

Caso a resposta à pergunta anterior não tenha sido não. Indique a sua preferência no uso de um sistema de conversão de design para código.
Realizar o design e o sistema conversor após a conclusão do design, em diferido
Usar o sistema conversor e a ferramenta com conversão bidireccional em tempo real. Alterações no código reflectiam alterações no design.
Realizar somente o design e ser outra pessoa a converter para código
Usar a ferramenta e o sistema conversor com actualização em tempo real

8.1.4 Instruções ao programador

Guia de desenvolvimento React

O presente documento serve como um guia para o a fase de testes com o utilizador, no contexto da tese de mestrado em Engenharia Informática de Mário João Gonçalves Ferreiro no Instituto Superior de Engenharia do Porto, cujo tema é: Construtor de Interfaces Web. O objetivo da tese é converter *design* produzido na ferramenta Sketch para código React, correspondente a esse design, através de uma ferramenta desenvolvida pelo autor. A fase de testes encontra-se dividida em dois, uma primeira parte em que é desenvolvido o *design* na ferramenta Sketch, e uma segunda onde o código gerado pela ferramenta conversora é fornecido a programadores para eventuais correções necessárias.

Cada uma das partes está sujeita a restrições definidas na secção com o mesmo nome.

No final da resolução da tarefa deste guia deverá ser respondido o inquérito disponível em:

https://forms.office.com/Pages/ResponsePage.aspx?id=KaZgeh_mDEOcYyZmU7K4mdwvCb8oQhZAmDzerMJodbNUMkIIME4xU0hBS0FWMlpJTTQ0WlNGV0Q1Mi4u

O tempo total previsto é de 1 hora: 45 minutos para a tarefa, 15 para o inquérito.

Restrições

Pela conversão do *design* para código não ser perfeita, as correções a aplicar deve respeitar a seguinte ordem decrescente de prioridade, em que o primeiro item é o mais prioritário:

1. Tornar a aplicação compilável
2. Adicionar navegação à aplicação (alteração entre páginas atribuindo funcionalidade, por exemplo, a botões)
3. Disposição de componentes igual ao design (colocar os componentes no local certo da página)
4. Lógica aplicacional, como transferência de dados entre páginas
5. Adição de possíveis componentes em falta
6. Atribuir propriedades estéticas cor, tamanho, entre outros

A localização dos ficheiros não deve ser altera, contudo, o nome dos mesmos pode ser corrigido de forma a tornar a solução compilável. É imperativo que a solução seja compilável, e esperado que haja navegação entre páginas. Não é esperado que todas as opções estéticas sejam corrigidas, mas deve ser realizado o máximo possível de forma a providenciar dados com qualidade ao estudo.

Antes de começar a tarefa é necessário instalar as dependências do projeto. Para isso basta correr o comando **npm install** na linha de comandos na raiz do projeto. Assume-se que as dependências das ferramenta React se encontram instaladas.

Deve ser gasto somente 45 minutos no desenvolvimento da tarefa, mesmo que não seja possível aplicar todas as correções.

Enunciado

O código foi gerado pelo utilitário create-react-app e usa a biblioteca Material-UI. Os ficheiros gerados automaticamente encontram-se localizados em pastas dentro da pasta src. Os ficheiros dentro destas pastas dizem respeito às diferentes páginas web da aplicação. Já as pastas servem para organizar as páginas que possam ter conceitos em comum, ou que possam estar relacionadas.

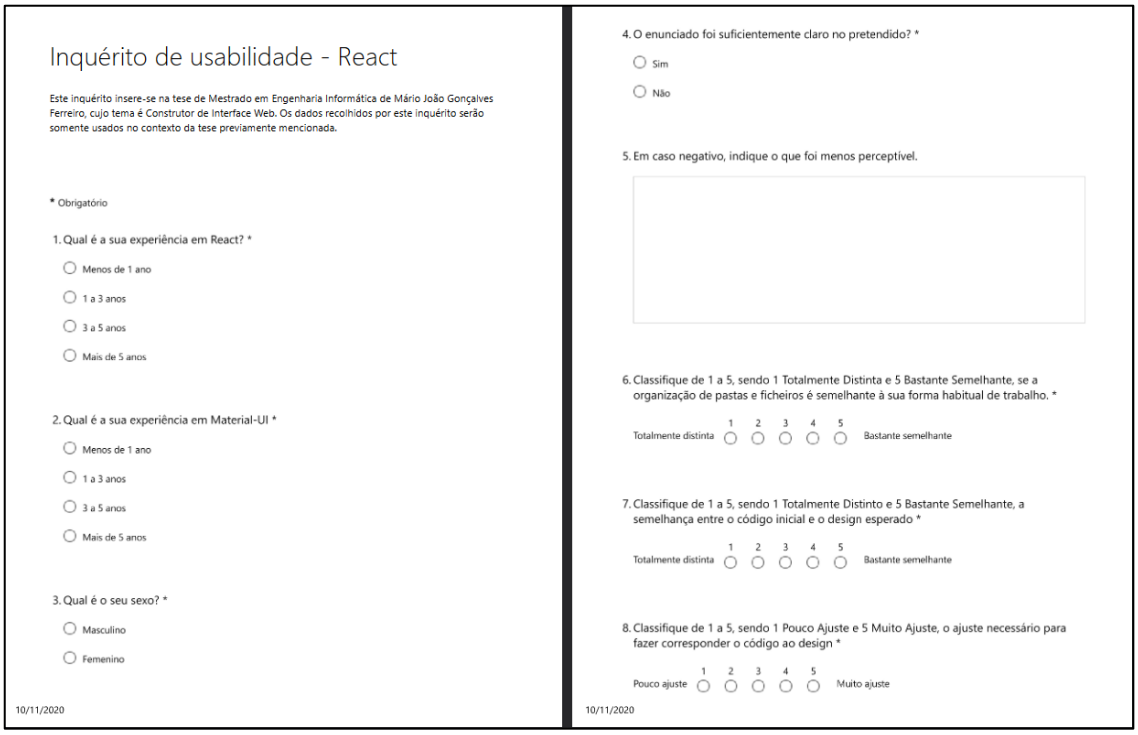
Com base no código fornecido, gerado pela aplicação conversora, pretende-se que sejam realizadas as correções necessárias de forma a tornar o resultado o mais parecido com o *design* fornecido.

Em conjunto com o código, foi enviado o *design* usado pela aplicação conversora. Este *design* deverá ser usado como objetivo a replicar. O *design* deve ser entendido como um *mockup* das páginas web.

Publicação

O código final deverá ser comprimido para uma pasta comprimida zip. E enviado de volta ao remetente.

8.1.5 Inquérito ao programador



Inquérito de usabilidade - React

Este inquérito insere-se na tese de Mestrado em Engenharia Informática de Mário João Gonçalves Ferreiro, cujo tema é Construtor de Interface Web. Os dados recolhidos por este inquérito serão somente usados no contexto da tese previamente mencionada.

* Obrigatório

1. Qual é a sua experiência em React? *
   - Menos de 1 ano
   - 1 a 3 anos
   - 3 a 5 anos
   - Mais de 5 anos

2. Qual é a sua experiência em Material-UI *
   - Menos de 1 ano
   - 1 a 3 anos
   - 3 a 5 anos
   - Mais de 5 anos

3. Qual é o seu sexo? *
   - Masculino
   - Femenino

10/11/2020

4. O enunciado foi suficientemente claro no pretendido? *
   - Sim
   - Não

5. Em caso negativo, indique o que foi menos perceptível.

6. Classifique de 1 a 5, sendo 1 Totalmente Distinta e 5 Bastante Semelhante, se a organização de pastas e ficheiros é semelhante à sua forma habitual de trabalho. *

	1	2	3	4	5	
Totalmente distinta	○	○	○	○	○	Bastante semelhante

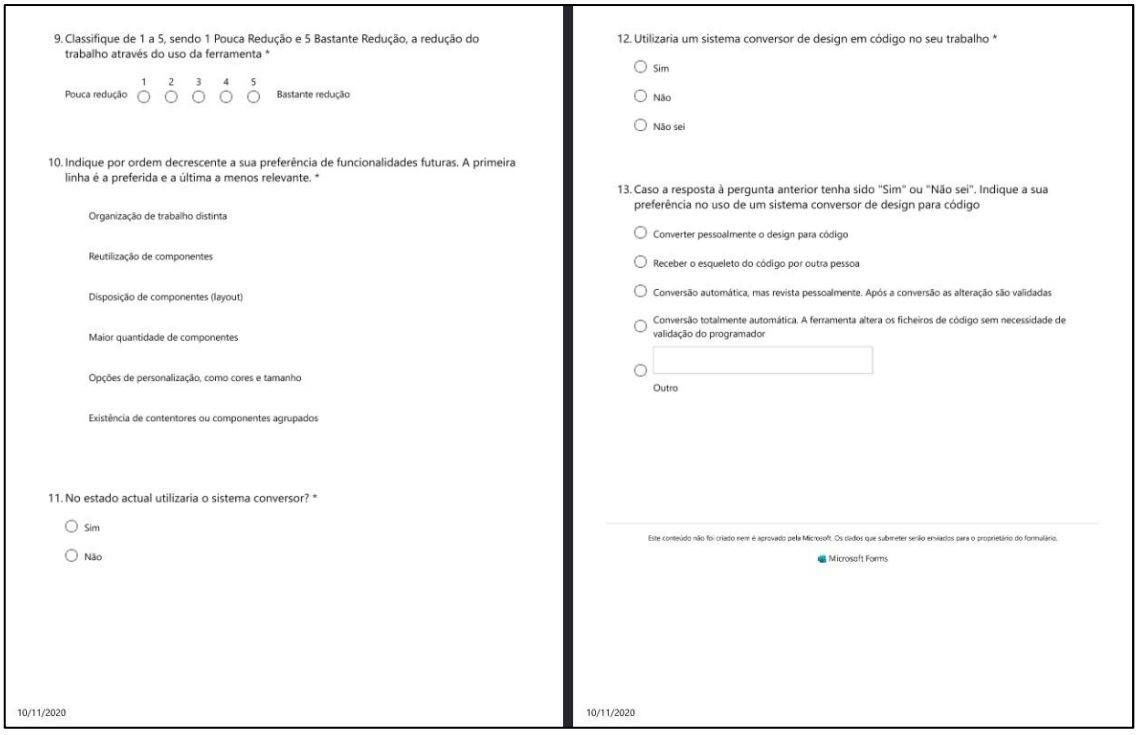
7. Classifique de 1 a 5, sendo 1 Totalmente Distinto e 5 Bastante Semelhante, a semelhança entre o código inicial e o design esperado *

	1	2	3	4	5	
Totalmente distinta	○	○	○	○	○	Bastante semelhante

8. Classifique de 1 a 5, sendo 1 Pouco Ajuste e 5 Muito Ajuste, o ajuste necessário para fazer corresponder o código ao design *

	1	2	3	4	5	
Pouco ajuste	○	○	○	○	○	Muito ajuste

10/11/2020



9. Classifique de 1 a 5, sendo 1 Pouca Redução e 5 Bastante Redução, a redução do trabalho através do uso da ferramenta *

	1	2	3	4	5	
Pouca redução	○	○	○	○	○	Bastante redução

10. Indique por ordem decrescente a sua preferência de funcionalidades futuras. A primeira linha é a preferida e a última a menos relevante. *
    - Organização de trabalho distinta
    - Reutilização de componentes
    - Disposição de componentes (layout)
    - Maior quantidade de componentes
    - Opções de personalização, como cores e tamanho
    - Existência de contentores ou componentes agrupados

11. No estado actual utilizaria o sistema conversor? *
    - Sim
    - Não

10/11/2020

12. Utilizaria um sistema conversor de design em código no seu trabalho *
    - Sim
    - Não
    - Não sei

13. Caso a resposta à pergunta anterior tenha sido "Sim" ou "Não sei". Indique a sua preferência no uso de um sistema conversor de design para código
    - Converter pessoalmente o design para código
    - Receber o esqueleto do código por outra pessoa
    - Conversão automática, mas revista pessoalmente. Após a conversão as alteração são validadas
    - Conversão totalmente automática. A ferramenta altera os ficheiros de código sem necessidade de validação do programador
    - Outro

Este conteúdo não foi criado nem é aprovado pela Microsoft. Os dados que submeter serão enviados para o proprietário do formulário.

Microsoft Forms

10/11/2020

8.1.6 Dados do inquérito ao programador

Qual é a sua experiência em React?	Qual é a sua experiência em Material-UI	Qual é o seu sexo?	O enunciado foi suficientemente claro no pretendido?	Em caso negativo, indique o que foi menos perceptível.
Menos de 1 ano	Menos de 1 ano	Masculino	Sim	
1 a 3 anos	Mais de 5 anos	Masculino	Não	Não entendi logo o que se pretendia fazer. Pensei que iria fazer o design no sketch e fazer a conversão até ao produto final. Após esclarecimento é que percebi que apenas se tratava apenas de tratar da conversão para o produto final (que no meu entender é um passo que não deveria existir).
Menos de 1 ano	Menos de 1 ano	Masculino	Sim	
3 a 5 anos	Menos de 1 ano	Masculino	Sim	

Classifique de 1 a 5, sendo 1 Totalmente Distinta e 5 Bastante Semelhante, se a organização de pastas e ficheiros é semelhante à sua forma habitual de trabalho.	Classifique de 1 a 5, sendo 1 Totalmente Distinto e 5 Bastante Semelhante, a semelhança entre o código inicial e o design esperado	Classifique de 1 a 5, sendo 1 Pouco Ajuste e 5 Muito Ajuste, o ajuste necessário para fazer corresponder o código ao design	Classifique de 1 a 5, sendo 1 Pouca Redução e 5 Bastante Redução, a redução do trabalho através do uso da ferramenta	Indique por ordem decrescente a sua preferência de funcionalidades futuras. A primeira linha é a preferida e a última a menos relevante.

5	4	4	4	Opções de personalização, como cores e tamanho; Disposição de componentes (layout); Maior quantidade de componentes; Reutilização de componentes; Existência de contentores ou componentes agrupados; Organização de trabalho distinta;
2	1	5	1	Reutilização de componentes; Maior quantidade de componentes; Opções de personalização, como cores e tamanho; Organização de trabalho distinta; Disposição de componentes (layout); Existência de contentores ou componentes agrupados;

	1	1	5	1	Reutilização de componentes; Disposição de componentes (layout); Organização de trabalho distinta; Maior quantidade de componentes; Opções de personalização, como cores e tamanho; Existência de contentores ou componentes agrupados;
	2	3	4	2	Disposição de componentes (layout); Reutilização de componentes; Opções de personalização, como cores e tamanho; Existência de contentores ou componentes agrupados; Maior quantidade de componentes; Organização de trabalho distinta;

No estado actual utilizaria o sistema conversor?	Utilizaria um sistema conversor de design em código no seu trabalho	Caso a resposta à pergunta anterior tenha sido "Sim" ou "Não sei". Indique a sua preferência no uso de um sistema conversor de design para código
Sim	Sim	Conversão automática, mas revista pessoalmente. Após a conversão as alteração são validadas
Não	Não sei	Conversão totalmente automática. A ferramenta altera os ficheiros de código sem necessidade de validação do programador
Não	Não	
Não	Não sei	Conversão automática, mas revista pessoalmente. Após a conversão as alteração são validadas