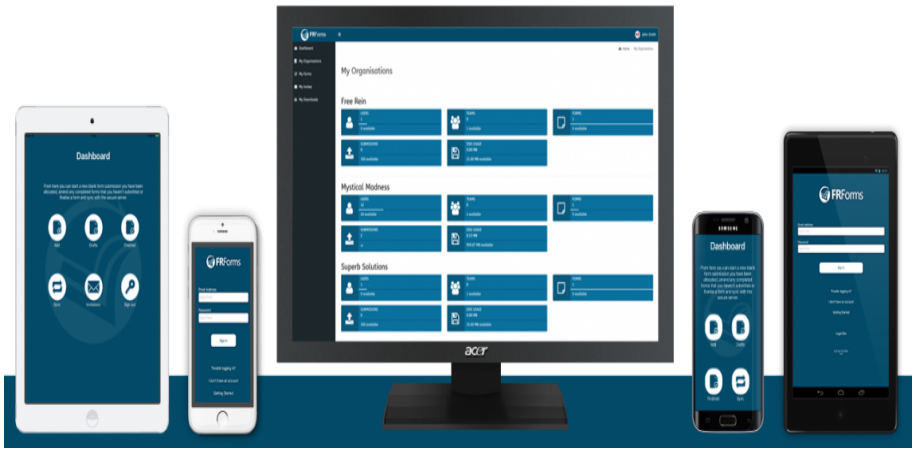
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



Plataforma mobile para captura de inquéritos de informação online/offline

Cláudia Filipa Lino Soares

Licenciatura em Engenheira Informática e de Computadores

Relatório Final para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Professor Doutor José Manuel de Campos Lages Garcia Simão

Júri:

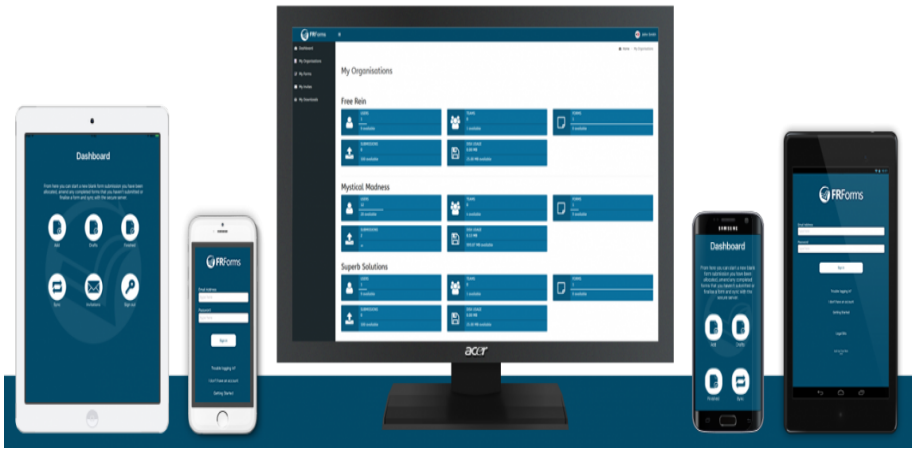
Presidente: Professor Doutor Tiago Miguel Silva Dias

Vogais: Professor Doutor Nuno Miguel Soares Datia
Professor Doutor José Manuel de Campos Lages Garcia Simão

Setembro, 2021

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



Plataforma mobile para captura de inquéritos de informação online/offline

Cláudia Filipa Lino Soares

Licenciatura em Engenheira Informática e de Computadores

Relatório Final para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Professor Doutor José Manuel de Campos Lages Garcia Simão

Júri:

Presidente: Professor Doutor Tiago Miguel Silva Dias

Vogais: Professor Doutor Nuno Miguel Soares Datia
Professor Doutor José Manuel de Campos Lages Garcia Simão

Setembro, 2021

Agradecimentos

Todo o trabalho desenvolvido neste projeto teve diversas contribuições, por parte de algumas instituições e de variadas pessoas. Como tal, gostaria de expressar os meus sinceros agradecimentos a todos os que tornaram possível este trabalho.

Ao ISEL, que me proporcionou as condições necessárias de aprendizagem, e aos professores que nesta instituição lecionam, que ao disponibilizarem alguns dos seus conhecimentos me tornaram capaz de desenvolver o projeto.

À Fordesi por propor este projeto e pela confiança depositada ao oferecer a possibilidade de desenvolver este projeto com a sua parceria.

Ao engenheiro José Simão que ao longo dos meses foi incansável ao se disponibilizar sempre que necessitava de orientações, por todo o seu apoio e dedicação. Também gostaria agradecer ao engenheiro José Alexandre Pedregal Tavares, pessoa responsável por parte da empresa envolvente, por todo o conhecimento disponibilizado acerca da empresa e dos detalhes do projecto.

Aos meus pais, à minha irmã e ao meu namorado pela compreensão, incentivo e apoio incondicional que sempre me deram.

Resumo

Hoje em dia existem muitas empresas que necessitam da recolha de informação de uma forma estruturada através de um dispositivo móvel, no formato de inquéritos.

É importante para estas empresas que seja possível guardar os dados introduzidos quer online, como offline, caso exista cobertura de internet ou não.

No caso de se encontrar no modo offline, a informação recolhida seria guardada localmente no dispositivo móvel, procedendo à actualização do repositório remoto, assim que exista as condições necessárias para o fazer.

A maioria das aplicações existentes para obtenção de dados apenas funciona em modo online, sendo uma desvantagem para as empresas, pois nem sempre existe internet nos locais onde as empresas pretendem utilizar os inquéritos, portanto seria útil realizar uma extensão destas aplicações já existentes para suportar o modo offline.

Para realizar esta funcionalidade foi necessário o planeamento de uma arquitectura global do sistema de informação, assim como o estabelecimento de um meta-modelo para definição dos formulários. A construção da aplicação móvel e do servidor aplicacional são outros dois pontos necessários á plataforma a desenvolver.

Palavras-chave: Arquiteturas para empresas, Sistemas Operativos, Princípios de aplicações web, Aplicações móveis, Formulários estruturados, Sincronização de dados.

Abstract

Nowadays many companies require information to be collected in a structured way via mobile devices, in the form of surveys.

It's important for these companies to be able to store the supplied information either online or offline, in case there is network connectivity or not. When offline, the collected information should be stored locally in the mobile device, and then when there is connectivity, it should proceed to transfer to a remote storage.

Most of the available applications can only work online for data gathering, and this is a disadvantage for those companies, because they can't resort to surveys' as a way of gathering data in locations where there isn't internet coverage, therefore it would be useful to provide an extension that works offline with those applications.

To create such functionality it was necessary to plan a global architecture for the information gathering system, as well as to establish a meta-model for the definition of the forms. The construction of the mobile application and applicational server are two major items needed to implement while the application is developed.

Keywords: Enterprise architectures, Operating systems, Organizing principles for web applications, Mobile Applications, Structed Forms, Data Synchronization.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Glossário

MVC Model-View-Control. xv, 13, 32, 35

XML Extensible Markup Language. xvi, 1, 8, 11, 13, 14, 16, 18, 25, 34, 60, 62, 63, 64, 71

UX User Experience. 2, 26

ODK Open Data Kit. 5, 6, 26

API Application Programming Interface. 6, 8

REST Representational State Transfer. 6

SSL Secure Sockets Layer. 7

HTML HyperText Markup Language. 13, 26

W3C World Wide Web Consortium. 13

SQL Structured Query Language. 17, 20, 21, 22, 23, 26

JSON JavaScript Object Notation. 18, 23, 25, 31, 33, 37, 41, 44, 71

ACID Atomicidade, Consistência, Isolamento, Durabilidade. 20

OLAP Online Analytical Processing. 20

OLTP Online Transaction Processing. 20

ORM Mapeamento Relacional a Objectos / Object-relational mapping. 20, 22

BD Base de dados. 21, 25

CSS Cascading Style Sheets. 26, 43

UI User Interface. 26

CASE Computer-aided software engineering. 26

HTTP Hypertext Transfer Protocol. 41, 70

1

Introdução

Atualmente um dos problemas das atividades empresariais é a recolha de informação de uma forma estruturada através de um dispositivo móvel, no formato de formulários. Estes são usados em dispositivos moveis, como na recolha de dados de marketing, censos, campanhas para angariação de clientes, trabalhos de campo científicos e laboratórios móveis.

Esta recolha de informação por vezes é realizada em locais onde existe uma ligação instável à *internet*. Por esta razão, torna-se difícil a utilização de dispositivos móveis, para guardar a informação. Por exemplo, nas aldeias do interior do nosso país é difícil que exista uma ligação estável à *internet*, ocasionalmente esta é mesmo inexistente [1, 38].

De modo a que recolha de dados seja feita de forma digital e estruturada, é necessário garantir que o preenchimento dos formulários poderá ser realizado em equipamentos móveis, mesmo que estes estejam em modo *offline*. Se o dispositivo móvel estiver no modo *offline* a informação recolhida será guardada localmente, no dispositivo móvel, procedendo à atualização do repositório remoto assim que existam as condições necessárias para o fazer, ou seja, existência de uma conexão estável.

Entende-se por formulários, toda a informação estruturada em ecrãs ou em ficheiros, sendo que estes podem ser pré-definidos numa linguagem, neste caso XML e instanciados através da aplicação móvel. Existe a necessidade de explorar mecanismos de definição de ecrãs para dispositivos móveis, restrição e controlo de informação, e por fim sincronismo com repositórios centrais de dados.

A necessidade de predefinição de formulários deve-se ao facto das empresas terem necessidades diferentes e assim dá-se a hipótese que esta seja livre de construir os seus próprios inquéritos. Por este motivo apenas existe a hipótese destes serem interpretados dinamicamente em tempo de execução, pois não existem *templates* para formulários na aplicação.

O sistema a desenvolver usa competências na definição de arquiteturas de sistemas distribuídos, desenvolvimento de aplicações moveis e aplicações *web*. O desenvolvimento de aplicações para dispositivos móveis é realizado de diferentes formas, umas mais complexas que outras. Para se desenvolver uma aplicação para um dispositivo móvel é necessário saber à partida qual o sistema operativo para o qual estamos a desenvolver a aplicação. A aplicação desenvolvida é baseada em aplicações já existentes de recolha de dados através de formulários. Muitas destas aplicações já existentes para obtenção de dados apenas funcionam em modo *online*, porque são tecnologicamente mais simples e baratas de desenvolver, sendo uma desvantagem para as empresas. É uma desvantagem, pois como foi mencionado anteriormente, nem sempre existe *internet* nos locais onde as empresas pretendem utilizar os formulários. Portanto, seria útil realizar aplicação semelhante a estas, mas suportando operações no modo *offline*.

Um dos grandes desafios desta aplicação é encontrar uma linguagem onde seja possível descrever formulários e que o dispositivo móvel consiga interpretar de tal modo que apresente num determinado ecrã aquilo que foi descrito no formulário. Esta linguagem deve ser independente do sistema operativo do dispositivo móvel, suportando qualquer tipo de descrição de *widgets* [3]. A interpretação do formulário recorrerá a um motor próprio de modo a identificar o *widget* que tem que ser criado dinamicamente. A instanciação destes componentes será realizada recorrendo à reflexão da classe do *widget* correspondente. Pretende-se ainda com esta solução guardar os dados preenchidos nos formulários caso o dispositivo móvel tenha ligação à *internet* ou não.

A criação dinâmica de *widgets* pressupõe que o tempo de geração de formulários seja elevado. Com esta aplicação pretende-se que esse tempo não seja muito demorado de modo a não comprometer a *User Experience* (UX) do utilizador.

A descrição dos formulários devem poder ser partilhados por diversos utilizadores da aplicação, por isso esta aplicação deve de conter um repositório onde seja armazenado esses inquéritos de modo a que todos consigam observá-los e utilizá-los caso necessitem. Os formulários são utilizados através do dispositivo móvel de modo a recolher os dados necessários.

1.1 Objetivos

O objetivo principal deste trabalho é desenvolver um sistema onde seja possível fazer a gestão de formulários, onde através de uma aplicação móvel seja possível interpretar os formulários descritos numa determinada linguagem. Os objetivos são:

- Analisar, escolher ou definir uma linguagem para especificação de formulários.
- Disponibilizar um repositório de formulários, possibilitando a sincronização entre o repositório e as aplicações móveis.
- Gerar dinamicamente e de forma eficiente os componentes gráficos correspondentes com base na representação textual dos mesmos.
- Usar um mecanismo de persistência e de sincronização entre repositórios de dados *online* e *offline*.

1.2 Organização do Documento

O presente documento encontra-se organizado em sete capítulos, cujos âmbitos se detalham de seguida.

O capítulo 2, expõe várias tecnologias ponderadas para o desenvolvimento de aplicação.

O capítulo 3, aborda o tema das bases de dados e mostra a arquitetura da plataforma desenvolvida.

O capítulo 4, descreve como é que a plataforma *web* e a aplicação móvel foram desenvolvidas de modo a dar suporte aos objetivos propostos, sendo também abordada a vertente de acesso a dados.

O capítulo 5, é demonstrado um caso prático da utilização desta plataforma.

O capítulo 6, são expostas as conclusões que se obtiveram durante a realização do projeto e descreve o trabalho futuro que pode ser realizado de forma a tornar a aplicação desenvolvida ainda mais rica.

2

Trabalho Relacionado

Neste capítulo aborda-se alguns dos trabalhos desenvolvidos no contexto do problema já mencionado anteriormente, analisando assim algumas aplicações para representação de formulários de forma dinâmica. Os trabalhos mencionados de seguida quando desenvolvidos, inicialmente não suportavam a recolha de dados sem ligação à *internet*.

As ferramentas que surgiram após os trabalhos relacionados foram:

- ODK
- Kobotoolbox
- SurveyCTO
- InfoPath

Estas ferramentas são usadas para recolher, analisar e utilizar os dados em áreas tão diversas como a saúde, agricultura ou educação [38].

2.1 ODK

O software ODK [1, 17], permite a recolha de variadíssimos dados, desde imagens a perguntas de resposta aberta. Este mecanismo ainda permite a construção de fluxos de trabalho, assim como, a sua partilha.

Este software é constituído por várias ferramentas interligadas entre si, estas são:

- **ODK Collect** [21]

  É uma aplicação *Android* e o seu código é *open-source*. A aplicação oferece uma ampla variedade de tipos de perguntas e respostas, sendo desenhada para funcionar bem sem conetividade da rede, mas não foi implementada inicialmente.

  O *ODK Collect* renderiza formulários através comandos de entrada que aplicam lógica de formulário, restrições de entrada e subestruturas de repetição. Os formulários podem ser guardados a qualquer momento e só após submeter o formulário é que este é enviado para o servidor.

- **ODK XLSForm** [43]

  O *XLSForm* é a norma utilizada para construir *forms* em *excel*. Esta é uma forma simples de começar e pode-se criar formulários complexos através deste padrão.

  As pessoas que não possuem *excel* instalado na suas máquinas podem sempre utilizar as ferramentas na web, por exemplo o *Google Sheets* ou o *Excel Web*.

- **ODK Build** [19]

  Esta ferramenta é utilizada para construir formulários através de uma interface com o utilizador de *drag-and-drop*.

  Esta é uma ferramenta web que utiliza *HTML5*, sendo que esta funciona melhor em formulários mais simples.

- **ODK Briefcase** [18]

  É uma aplicação *desktop* que pode ser executada em *macOS*, *Windows* ou *Linux*. A ferramenta tem como utilidade fazer *pull*, *push* e exportar formulários dos servidores *ODK*, como **ODK Central** e **ODK Aggregate**. Também se pode usar como repositório de formulários obtidos directamente do **ODK Collect**.

- **ODK Central** [20]

  É um servidor *ODK* que serve para gerir contas e permissões de utilizadores, armazena definições de formulários e permite que o **ODK Collect** se ligue para realizar o *download* e *upload* de formulários. Este servidor pode ser utilizado para implementar novas funcionalidade utilizando APIs programáticas *REST*, *OpenRosa* e *OData*.

- **ODK Validate** [23]

  ODK Validate garante que um arquivo *XML XForms* está em conformidade com a especificação *XForms*. Esta ferramenta deve ser utilizada para verificar se os

XForms foram editados manualmente. Se se usar as ferramentas de criação de formulários como o *XLSForm* ou *ODK Build* não é necessário usar o **ODK Validate**, a menos que se edite esses formulários manualmente após criá-los.

- **ODK JavaRosa** [22]

  ODK JavaRosa é uma biblioteca *Java* que serve renderizar formulários que são compatíveis com a especificação *ODK XForms*.

No ano corrente foi lançado o **ODK-X** que é uma versão mais actualizada da ferramenta ODK, que tem como características principais as seguintes: [24]

- Sincronização de dados bidireccional
- Recolha de dados em modo *offline*
- Pesquisas nos formulários
- Visualização de dados no dispositivo
- Controlo de acessos do utilizador
- Alteração de aparência da aplicação

2.2 Kobotoolbox

O *KoBotoolbox* [12] é constituído por três ferramentas, uma para criar o formulário, outra para recolher dados, e a terceira ferramenta tem como utilidade analisar e gerir dados.

O *Form Builder*, ferramenta de criação de formulários, é apresentada como uma ferramenta rápida e fácil criação de formulários. Esta ferramenta ainda permite a reutilização de perguntas ou de blocos de perguntas já existentes, é possível criar *workflows* de perguntas com alguma variedade de perguntas. É ainda possível a partilha dos formulários assim como também há a possibilidade de importação e exportação de *XLSForms*.

A recolha de dados é feita através da ferramenta *Collect Data*, esta permite que a colheita de dados seja realizada em modo *online* e *offline*. Esta operação pode ser executada em diversos dispositivos, telemóveis, *tablets* ou computadores. Os dados depois de recolhidos, caso haja ligação à *internet* são sincronizados por via *SSL*. Os dados ficam disponíveis assim que haja a sincronização.

Por último, esta aplicação ainda contém uma ferramenta de análise e gestão de dados. Esta permite criar resumos de relatórios que contem gráficos e tabelas. Nesta ferramenta também é possível visualizar através de um mapa onde a informação foi recolhida, também dá para exportar os dados. Existe ainda uma API que a partir dela consegue-se obter todos os dados.

Esta ferramenta também é *open-source* e gratuita, foi desenvolvida após existir uma necessidade de recolha de informações rápidas numa crise humanitária, especialmente após um desastre natural.

2.3 SurveyCTO

O *SurveyCTO* [36] é apenas mais uma ferramenta que faz o mesmo que as outras mencionadas anteriormente, porém, com algumas melhorias ao nível do servidor, do software de *desktop* e umas pequenas diferenças nos formulários de pesquisa [37].

Esta ferramenta permite criação de formulários complexos através de *drag-and-drop*, pré-carregamento de dados e o versionamento de formulários.

A recolha de dados pode ser feita de forma *offline* caso seja usado com as aplicações *SurveyCTO* para *Android* e *iOS* ou *online* com a interface da web universal. Os dados são enviados para o servidor de forma segura devido ás várias camadas de criptografia e redundância.

Nesta ferramenta ainda é capaz de monitorizar os dados em tempo real e detectar prováveis problemas. Depois dos dados recolhidos é possível fazer a análise dos mesmo através de um *dashboard*.

Esta já não é gratuita nem *open-source*.

2.4 InfoPath

O *InfoPath* [10] é uma aplicação desenvolvida pela *Microsoft*, faz parte dos produtos do *office* e é utilizado para gerar dados em XML, mas sendo que esta nunca vem instalada por defeito no computador.

O utilizador deve desenvolver um modelo no *InfoPath* antes de usar o mesmo para preencher um formulário.

Os dados dos formulários do *InfoPath* são armazenados em formato XML. Esta plataforma fornece vários componentes controladores (por exemplo: *Textbox*, *Radio Button*, *Checkbox*, etc.) para apresentar os dados a utilizadores finais.

As acções complexas podem ser desenvolvidas através da validação de dados, contudo, também é possível ser feito através da programação em *VisualBasic* dentro de um documento do *Microsoft Word* ou *Excel*.

Atualmente, esta aplicação foi substituída pela *Power Apps* que interage com as versões mais recentes das aplicações Microsoft. Esta aplicação é um pouco diferente das outras porque esta apenas permite a criação de formulários e não a recolha de dados.

3

Arquitetura Proposta

Neste capítulo abordar-se-á toda a informação necessária para a resolução do problema apresentado anteriormente, assim como, a sua arquitetura. Neste trabalho analisam-se quais as linguagens que se adaptam melhor à construção de formulários. O uso de formulários na *internet* é tão comum que a maioria das interacções do utilizador com o *browser* envolve uma forma de formulário.

A linguagem comum de se usar para construir um formulário é o *HTML*[7], porém posteriormente ao *HTML*, foram criadas duas novas formas de desenvolver formulários: *XForm* [42] e *InfoPath* [10].

Estas duas formas de desenvolver formulários utiliza como linguagem o XML, sendo assim uma alternativa aos formulários *HTML*.

Ao existir esta ligação com XML, a especificação *XForm* permite assim a criação de formulários flexíveis para introdução de dados por parte do utilizador, sendo que estes podem ser aplicados a uma ampla variedade de plataformas, incluindo computadores e dispositivos móveis.

Há que ter em conta, que se precisará de se lidar com bases de dados seleccionando assim as que se adequam mais ao que pretendemos solucionar. Neste caso o armazenamento de dados tem que ser feito caso exista ou não uma ligação estável à *internet*.

A estrutura de dados que a base de dados suporta também é importante, de modo a facilitar a leitura e escrita de dados, isto é, como os formulários tem diversos campos e podem variar em termos de número de *widgets* e de tipo de dados, a base de dados terá

```
<!DOCTYPE html>
    <html>
    <body>
        <h2>Document</h2>
        <form action="xxxx">
          <label>Whats your name?</label><br>
          <input type="text" name="Whats your name?"><br>
          <label>How old are you?</label><br>
          <input type="text" name="How old are you?"><br><br>
          <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

Listagem 3.1: Código de um formulário HTML.

que ser flexível a esse ponto de modo a permanecer a informação recolhida através do formulário. Quando se refere o tipo de dados é se o valor que se pretende guardar é um texto, um número, uma data ou um binário, por exemplo. O tipo de sincronização, também é algo relevante, pois tem que se escolher alguma base de dados que seja suportada pelo dispositivo móvel e que seja possível ser sincronizada para uma base de dados remota e vice-versa.

3.1 Linguagens para definição de formulários

3.1.1 Formulários *HTML*

A linguagem *HTML* sofreu algumas melhorias após ser criada no início dos anos 90. Uma das melhorias foi a adição da definição de formulários, após esta evolução a navegação na *internet* tornou-se totalmente diferente. A Listagem 3.1 mostra um exemplo da definição de um formulário em *HTML* No mundo da *internet* a utilização de formulários *HTML* é essencial para qualquer site, a recolha de informação feita através dos formulários poderá ser usada para fins futuros. À medida que os criadores aumentaram os limites da tecnologia de formulários, os sites tornaram-se iterativos e personalizáveis, com alguma troca de informação a experiência de navegação foi melhorada. [8]

A linguagem *HTML* foi criada para funções especificas: [40]

- Permite o controlo sobre a exibição do documento.
- Fornece flexibilidade para descrever diferentes tipos de dados.

- Transmite informações de uma variedade de media e em vários formatos.

A linguagem HTML, não é considerada uma linguagem de programação pois não é possível gerar funcionalidade programaticamente.[9]

Um formulário na linguagem *HTML* é uma secção de um documento que contém conteúdo normal, *tags*, elementos chamados controladores (dropdows, checkboxes, menus, entre outros.). Os utilizadores geralmente preenchem um formulário utilizando as opções de inserir texto, seleccionar itens do menu etc., antes de enviá-lo de modo a ser processado.

Este elemento que define formulários *HTML* é designado de *form*. Esta *tag* descreve alguns aspetos importantes do formulário, incluindo o *script* que servirá para recolher os dados presentes no formulário e enviar para o servidor *Web*, de modo a que a informação seja guardada.

Devido às dependências que a linguagem *HTML* tem com linguagens de *script*, para guardar dados foi criado duas outras formas de recolher dados sem utilizar *scripts*, usando apenas a linguagem XML: *XForms* ou *InfoPath*.

3.1.2 Formulários *XForms*

O *XForm* foi criado pelo W3C para gerar formulários. Esta linguagem veio a corrigir dois aspectos do *HTML*, a separação entre a camada de dados e a camada de interface com o utilizador, e a da dependência de *scripts* para a validação e processamento dos formulários.

O *XForm* baseia-se em *XML Schema*, *XPath* e *XML Events*, este tenta resolver algumas das limitações do *HTML*. Visto que esta linguagem foi construída de acordo com o padrão MVC, com esta separação de responsabilidades as pessoas que estão a desenvolver ou preencher os formulários podem desenvolve-lo com melhor produtividade. Na Figura 3.1 pode-se visualizar um exemplo da separação do modelo *XForm* da forma de apresentação. Um mesmo formulário pode ser apresentado em diversos dispositivos.

Ao utilizar *XForm* para recolher dados em XML tem várias vantagens comparado com os formulários HTML, como por exemplo:

- Os dados do formulário podem ser preenchidos e enviados em XML.
- Documentos em linguagem XML podem ser carregados como dados iniciais do formulário.

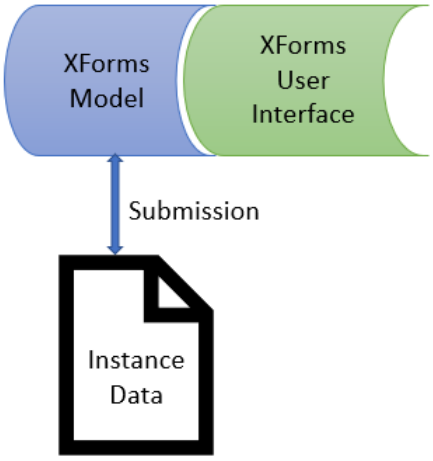


Figura 3.1: Camadas do XForms.

- Expressões *XPath* e tipos de dados *XML Schema* podem ser usados no *XForm*.
- *XForms* podem ser integrados com serviços da *internet*.
- *XForms* é independente de dispositivo - a mesma estrutura XML pode ser usada em qualquer dispositivo.

Os dados adquiridos no *XForms* são armazenados em instâncias de documentos XML, que seguem uma especificação inicialmente definida no modelo. Através de um protocolo de submissão, o *XForms* é capaz de enviar e receber dados, como se pode verificar na Figura 3.2.

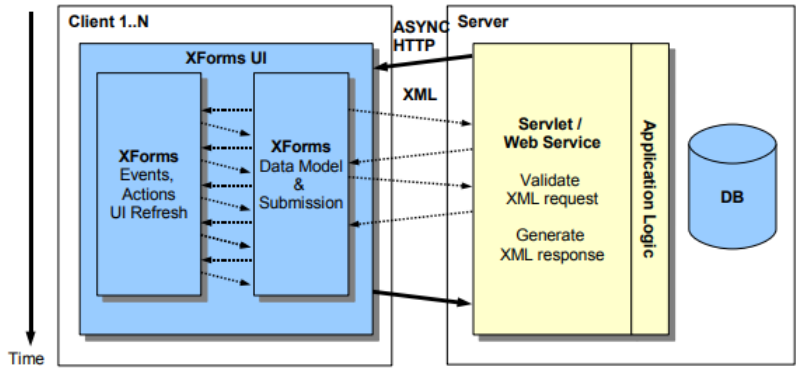


Figura 3.2: Modelo de interação do *XForm*.

Na Listagem 3.2, podemos ver um exemplo de um formulário *XForm*.

```
<h:html xmlns="http://www.w3.org/2002/xforms" xmlns:h="http://www.w3.org
    /1999/xhtml" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:jr="http
    ://openrosa.org/javarosa">
  <h:head>
    <h:title>Untitled Form</h:title>
    <model>
      <instance>
        <data id="build_Untitled-Form_1600794406">
          <meta>
            <instanceID/>
          </meta>
          <Whats_your_name/>
          <How_old_are_you/>
        </data>
      </instance>
      <itext>
        <translation lang="English">
          <text id="/data/Whats_your_name:label">
            <value>Whats your name?</value>
          </text>
          <text id="/data/How_old_are_you:label">
            <value>How old are you?</value>
          </text>
        </translation>
      </itext>
      <bind nodeset="/data/meta/instanceID" type="string" readonly="true()"
          calculate="concat('uuid:', uuid())"/>
      <bind nodeset="/data/Whats_your_name" type="string"/>
      <bind nodeset="/data/How_old_are_you" type="int"/>
    </model>
  </h:head>
  <h:body>
    <input ref="/data/Whats_your_name">
      <label ref="jr:itext('/data/Whats_your_name:label')"/>
    </input>
    <input ref="/data/How_old_are_you">
      <label ref="jr:itext('/data/How_old_are_you:label')"/>
    </input>
  </h:body>
</h:html>
```

Listagem 3.2: Código de um formulário XForm.

3.1.3 Formulários *InfoPath*

O *InfoPath* é uma tecnologia da Microsoft, não gratuita. Esta aplicação é um editor de formulários onde se pode criar um formulário através de um documento XML ou através da parte design da aplicação.

Os formulários do *InfoPath* podem ser preenchidos com ou sem ligação estável à *internet*. No entanto, os utilizadores devem ter o *InfoPath* instalado nos computadores para usar esses formulários, e só é possível esta instalação em computadores. A tecnologia *InfoPath* foi considerada por os seguintes motivos:

- O *InfoPath* é fácil de usar e não requer nenhum conhecimento de XML.
- Os dados do formulário do *InfoPath* podem ser exportados para outras aplicações da Microsoft.
- O *InfoPath* permite seleccionar vários formulários e combinar os dados num só formulário.
- Pode-se usar um documento XML existente e criar automaticamente um formulário baseado na estruturado documento existente.
- O *InfoPath* pode ser integrado com serviços de *internet*.
- O *InfoPath* inclui validação de dados.

O *InfoPath* não é uma boa tecnologia para se usar neste projeto, visto que a aplicação tem de estar instalada num computador e do software não ser gratuito.

3.2 Base de dados

A aplicação desenvolvida neste trabalho tem necessidade de armazenamento de dados. Este armazenamento tem duas categorias de dados distintos, formulários e informação recolhida através dos mesmos.

O armazenamento de dados tem de ser feito localmente no dispositivo móvel uma vez que pode não existir ligação à *internet*. No entanto, assim que haja uma ligação estável à *internet* terá que ser possível a sincronização dos dados para a base de dados remota.

3.2.1 Diferenciação de tipo de base de dados

As bases de dados armazenam conjuntos de dados organizados, segundo um modelo de dados, sendo que estes podem ou não estar relacionados entre si. Estas podem ser encontradas em diversos sistemas informáticos, como por exemplo: *smartphones*, servidores, computadores pessoais, entre outros; pois estas guardam as informações necessárias para o funcionamento dos sistemas.

A forma como os dados são inseridos e organizados e relacionados é que fazem com que a base de dados seja ou não relacional.

3.2.1.1 Base de dados relacional

As bases de dados relacionais usam SQL, e a forma de representar os dados é através de tabelas. Estas tabelas, onde são armazenados dados, as colunas são as propriedades dos dados a armazenar e as linhas são os registos de dados que guardamos na tabela. Os dados que são inseridos numa tabela normalmente são únicos, pois contém uma chave única que os distingue.

Quando são necessários detalhes ou dados adicionais, é necessário a criação de uma nova tabela, sendo que os dados precisam de ser associados entre as duas tabelas, a esta associação é chamada de relação. Uma chave comum é estabelecida entre as duas ou mais tabelas e depois é usada para essa associação.

Exemplos de base de dados relacionais:

- Microsoft SQL Server[15]
- SQL do Microsoft Azure [35]
- PostgreSQL [28]
- MariaDB [14]
- MySQL [16]
- Oracle [25]

3.2.1.2 Base de dados não relacional

As bases de dados não relacionais podem ser repositórios de chave-valor [2], documentais, orientados a colunas e baseadas em grafos. Os repositórios chave-valor, tratam os

dados como uma única colecção tendo uma string como chave. Este tipo de armazenamento geralmente utilizam menos memória enquanto guardam a mesma quantidade de dados, aumentando assim o desempenho.[4] Normalmente, este tipo de armazenamento é adoptado na gestão da sessão por parte das aplicações Web.

As base de dados documentais como o nome indica o armazenamento de dados é feito através de um documento, este documento permite a existência de várias "categorias" de dados. Devido à falta de estrutura destas bases de dados, estas tornam-se facilmente escaláveis. Os documentos deste tipo de base de dados podem armazenar dados em vários tipos de linguagem, com por exemplo: JSON e XML.

As bases de dados orientadas a colunas organizam os dados por campo, em vez do nome do atributo estar na horizontal estão dispostos na vertical.

Por último, as bases de dados orientadas a grafos utilizam-se quando existe uma necessidade muito grande de armazenar, mapear e procurar dados. Este tipo de bases dados utiliza conceitos como relações, conexões, nós e arestas.

Um nó representa uma entidade ou um dado. Quando dois nós se ligam designa-se de relação ou conexão, que representa uma aresta. Por isso, um nó pode ter uma aresta de entrada ou de saída, ou ambas, sendo definidos por uma seta a sua direcção. Todas estas bases de dados permitem flexibilidade e adaptabilidade à medida que é criada a aplicação. Se os requisitos de dados não estiverem claros desde o início, ou se se estiver a lidar com grandes quantidades de dados não estruturados, pode não ser possível desenvolver uma base de dados relacional com relacionamentos entre tabelas bem definidos. Em vez destas bases de dados usarem tabelas, usam documentos que permitem armazenar dados não estruturados num único documento. O armazenamento não será tão organizado como uma base de dados relacional. [13]

Base de dados não-relacionais:

- MongoDB
- Oracle NoSQL
- Apache CouchDB
- Redis
- Cassandra DB
- Neo4j

Deve-se usar uma base de dados relacional quando:

- Os princípios *atomicidade, consistência, isolamento, durabilidade* se aplicam , o que reduz anomalias e reforça a integridade.
- A estrutura de dados não muda. Se o *design* da aplicação é sólido e não se espera que mude com os requisitos futuros.

Usar base de dados não-relacionais quando:

- Se pretende fazer um desenvolvimento rápido de aplicações. Uma base de dados não-relacional oferece suporte a projetos em rápida mudança e "*sprints* de codificação"e é perfeito para configurações mais ágeis, onde os requisitos mudam frequentemente.
- O armazenamento de dados é realizado em quantidades elevadas e com pouca ou nenhuma estrutura. Assim como expresso no ponto anterior, se seus requisitos de dados não estiverem claros, sabe-se que é necessário armazenar muitos dados e, de alguma forma, poder-se-á usar esse tipo de base de dados, que pode ser transformada rapidamente para corresponder ao requisito.

Na Tabela 3.1 tem-se um comparativo das caraterísticas das bases de dados relacionais e não relacionais com suas capacidades e pontos fortes.

Caraterística	**SQL**	**NoSQL**
Operações ACID	Sim	Não
OLAP / OLTP	Sim	Não
Análise dos dados	Sim	Não
Modelo consistente	Sim	Não
Flexibilidade formato de dados	Não	Sim
Computação distribuída	Sim	Sim
Escalamento / Dimensionamento	Sim	Sim
Desempenho com crescimento de dados	Menor	Maior
Popularidade / Suporte comunidade	Sim	Crescente
Modelo de Armazenamento	Tabelas com colunas e linhas fixas	Documentos JSON, Chave-Valor e outros tipos
Histórico	Desenvolvido nos anos 70, com foco em redução de dados duplicados	Desenvolvido em 2000 com o foco em escalabilidade e mudança rápida de desenvolvimento
Exemplos	Oracle, MySQL, Microsoft SQL Server, e PostgreSQL	Documento: MongoDB e CouchDB, Chave-Valor: Redis e DynamoDB, Wide-column: Cassandra e HBase, Graph: Neo4j e Amazon Neptune
Escalonamento	Vertical (Com mais poder de processamento na mesma máquina)	Horizontal (Escala distribuindo em duas ou mais máquinas)
Transações	Suportado	A maioria não suporta, no entando o MongoDB sim
Mapeamento de Dado para Objeto	Requer um ORM	Pode não precisar de um ORM. Os documentos no MongoDB mapeiam diretamente para dados de estrutura das maiorias das linguagens

Tabela 3.1: Caraterísticas das bases de dados relacionais e não relacionais. [39]

3.2.1.3 Base de dados em dispositivos móveis

A forma mais comum de armazenar e gerir dados é através das bases de dados. As bases de dados são manipuladas no servidor ou na nuvem em que os dispositivos móveis somente comunicam pela rede. No entanto, as aplicações de hoje em dia são mais interativas, a tendência das aplicações será diminuir cada vez mais a dependência da conetividade da rede, utilizando o modo *offline*.

Atualmente, as aplicações mantêm a base de dados localmente ou fazem uma cópia da BD para a *cloud* no dispositivo local e sincronizam uma vez por dia ou sempre que houver uma conetividade de rede.

As bases de dados para dispositivos móveis precisam de ser:

- Optimizadas, pois o armazenamento é limitado.
- Independentes do requisito do servidor.
- Em forma de biblioteca sem dependências ou muito limitada para que possa ser usada quando necessário.
- O acesso a dados e a inserção não pode demorar.
- Deve assegurar a segurança dos dados.
- Fácil de manusear através do código e ter a opção de torná-la privada ou partilhada com outras aplicações.
- Económicas, isto é, ter um consumo baixo de memória e energia.

Existem muitas bases de dados móveis a chegar ao mercado, mas nem todas atendem aos requisitos mencionados acima.

De seguida, foram seleccionadas algumas bases de dados que cumprem parte ou a totalidade dos requisitos descritos anteriormente. Estas bases de dados são utilizadas em aplicações móveis.

SQLite

SQLite [34] é uma biblioteca que fornece um sistema de gestão de base de dados SQL, implementando um mecanismo transaccional independente, sem depender de um servidor e sem ter que haver qualquer tipo de configuração. O *Lite* significa pouco peso em termos de configuração, administração e recursos necessários.

Esta biblioteca suporta todos os recursos de bases de dados relacionais e é compacta, sendo que o seu código é Open-Source.

O SQLite está presente quer no sistemas operativo *Android*, quer no *iOS*.

O *SQLite* pode ser armazenado tanto em disco como em memória e cada arquivo de base de dados é um único arquivo e pode ser usado em várias plataformas. Esta é muito rápida e precisa de pouca memória para operar.

Realm DB

Realm DB [30] é um sistema de gestão de base de dados relacional que é como as bases de dados convencionais, na qual os dados podem ser consultados, filtrados e persistidos.

A base de dados *Realm* é desenvolvida para ser executada em dispositivos móveis. Como o *SQLite*, o *Realm* também não tem dependência de um servidor remoto.

O Realm DB tem muitas vantagens sobre o *SQLite* nativo, como:

- Os objetos no *Realm* são objetos nativos. Não é preciso copiar objetos da base de dados para alterá-los e guardá-los.
- Os objetos permanecem sempre sincronizados.
- O *Realm* é muito mais rápido que o *SQLite*. O *Realm* pode consultar até 57 registos/s, enquanto o *SQLite* pode fazer apenas até 20 registos/s. [31]
- Os dados podem ser encriptados.
- O *Realm* tem uma arquitetura que permite ligação diretamente à interface do utilizador, se os dados forem alterados, estes serão atualizados automaticamente.
- Existe sincronização automética com o servidor remoto caso existe uma ligação estável à *internet*.
- Uma aplicação pode ter dois tipos de bases de dados *Realm*, locais e remotas.
- Pode definir permissões diferentes para utilizadores diferentes.
- Disponível para *Android*, *iOS*, Java*script* etc.

ORMLite

ORMLite [26] é uma versão optimizada do ORM, que fornece algumas funcionalidades simples para a persistência de objetos **java** nas bases de dados SQL.

Esta base de dados é usada para simplificar operações SQL mais complexas, este também fornece um construtor de consultas flexível. O *ORMLite* é útil em aplicações amplas com consultas de dados complexas porque lida com instruções SQL "compiladas". Esta também suporta a configuração de tabelas e campos sem anotações e suporta chamadas nativas para *APIs* de base de dados *SQLite* do *Android*.

No geral, o *ORMLite* é um bom substituto do *SQLite* se considerarmos que a aplicação tem uma dimensão considerável e complexa, em termos de uso da base de dados.

Couchbase Lite

O *Couchbase Lite* [5] é uma base de dados que utiliza JSON incorporada no *NoSQL*, é uma base de dados escalável e é considerada segura. Os dados na *Couchbase Lite* são armazenados como documentos JSON, cada documento pode ter um ou mais anexos.

Esta solução é fornecida para aplicações móveis através da *Couchbase Lite*. Este componente é composto por três componentes diferentes: *Couchbase Lite*, uma base de dados *NoSQL*, *Sync Gateway*. A *Couchbase* primeiramente é uma base de dados *offline* e é necessário sincroniza-la quando a rede estiver disponível.

A *Couchbase Lite* é executada localmente no dispositivo e persiste os dados em JSON. Todas as operações, *create*, *read*, *update* e *delete*, são executadas na base de dados local. O programador não precisa de escrever código de sincronização, para sincronizar a base de dados local com a remota, uma vez que é tratado pelo *Sync Gateway*. O *Couchbase Lite* possui um mecanismo de resolução de conflitos.

Através desta base de dados existem *APIs* nativas para *Android* e *iOS*, portanto, se houver a necessidade de qualquer base de dados *NoSQL* no sistema operativo, o *Couchbase Lite* tem a maioria dos requisitos mencionados acima.

Firebase Database

Firebase database [6] é uma base de dados não relacional que está hospedada na *cloud*. Os dados são armazenados em JSON. Estes dados são armazenados e sincronizados em tempo real quando o dispositivo não está em modo *offline*. No caso do dispositivo móvel esteja no modo *offline*, os dados são persistidos localmente em cache no dispositivo, mas assim que exista a possibilidade os dados são sincronizados automáticamente.

Esta base de dados é utilizada por dispositivos *iOS*, *Android* e aplicações desenvolvidas em *JavaScript*, todos os seus clientes compartilham uma instância recebendo assim as atualizações automaticamente.

3.2.2 Sincronização entre base de dados

Quando ocorre uma alteração de algum dado e este é gravado tem que haver propagação do valor para a outra base de dados, a esta propagação de valor designa-se de sincronização de dados. Nem todas as bases de dados fazem a sincronização da mesma forma. Esta sincronização é possível devido a ambas as bases de dados terem o mesmo modelo definido. Para que esta sincronização de dados ocorra é necessário que haja **replicação**.

3.2.2.1 Replicação

O termo replicação é aplicado, sempre que é reproduzido um conjunto de operações que tenha ocorrido numa base de dados e tenha sido realizado numa outra. Caso não tenha sido reproduzido o mesmo conjunto de operações poderá ocorrer inconsistências nas bases de dados.

Uma replicação pode ser executada de forma parcial, onde apenas alguns dos dados é que são replicados em outra(s) bases de dados, ou de forma total, em que todos os dados são replicados. Se a disponibilidade de uma das cópias estiver acessível, na replicação total, a fiabilidade dos dados aumenta, visto que o utilizador não depende apenas dos dados disponibilizados numa única base de dados. Caso ocorra algum problema no sistema, o utilizador poderá solicitar uma réplica desses dados.

Ao fazer uma replicação, existe alguns tipos de vantagens, como por exemplo: [33]

- Disponibilidade - Pode ser significativamente melhorada, pelo menos para leituras. O sistema pode continuar a operar desde que um dos locais esteja ativo.
- Desempenho - Pode ser melhorado, pois o resultado de uma pergunta pode ser obtido localmente, em vez de ser necessário aceder a um sistema remoto.
- Segurança - Pode-se proteger melhor a versão original dos dados. A existência de réplicas reduz a possibilidade de se perderem os dados em caso de catástrofe.
- Autonomia local - Podem-se realizar os processamentos locais sem necessidade de acesso remoto.

Existem dois tipos de replicação: Assíncrona e Síncrona. A replicação assíncrona as réplicas podem apresentar valores diferentes durante períodos de tempo mais ou menos longo. A replicação síncrona, através de transacções, todas as réplicas apresentam os mesmos valores. As transacções podem ser distribuídas ou com cópias físicas.

3.2.2.2 Tipos de Sincronização

Existem dois tipos de sincronização que podem ser divididas em *one-way*, onde a transferência de dados é transmitida num único sentido, e *two-way*, onde as modificações ocorrem nos dois sentidos.[32]

A sincronização *one-way* pode ser efetuada na forma de Cliente/Servidor e vice-versa. Na sincronização Cliente/Servidor o dispositivo cliente é responsável por enviar as modificações das tabelas da base de dados ao servidor, que irá atualizar os seus registos conforme o que enviado pelo transmissor. Se o registo possuir a mesma chave primária do servidor, a sincronização irá subscrever o registo do servidor. Caso seja na forma de Servidor/Cliente o dispositivo servidor envia todos os registos contidos nas tabelas ao dispositivo cliente. No final da sincronização o dispositivo cliente possui a mesma versão da base de dados do servidor. As alterações executadas no dispositivo cliente não serão capturadas pelo servidor.

A sincronização *two-way*, como o nome indica, ocorre nos dois sentidos. As modificações são efetuadas tanto no dispositivo cliente quanto no servidor. O processo de sincronização inicia-se pelo cliente, enviando ao servidor os registos, o servidor por sua vez identifica e trata os conflitos existentes retornando ao dispositivo cliente os dados consolidados. No final do processo os dois dispositivos possuem a mesma versão do BD, sem haver qualquer tipo de perda.

Existe pelo menos duas formas de fazer a sincronização entre base de dados: a sincronização total e a sincronização baseada em delta. A sincronização total é o processo que envia todos os dados, mesmo que estes não tenham sido modificados e copiar para a base de dados de destino. Tecnicamente, porém, esse padrão pode ser implementado com qualquer tipo de base de dados, incluindo conjuntos de dados codificados em JSON ou XML. Na sincronização baseada em delta o sistema periodicamente verifica o que foi alterado e sincroniza apenas os dados modificados.

3.2.3 Custos de utilização das bases de dados

Das bases de dados mencionadas anteriormente, *SQLLite*, *Real DB*, *ORMLite*, *Berkeley DB*, *Couchbase Lite* e *Firebase Database*, três destas são de utilização gratuita as outras não. As bases de dados *SQLLite*, *ORMLite* e *Couchbase* são gratuitas, sendo que as outras tem sempre um custo associado, são elas *Berkeley DB*, *Firebase Database*, *Real DB*.

3.2.4 Plataformas de desenvolvimento

Existem múltiplas plataformas de desenvolvimento para dispositivos móveis, mas nem todas são simples de utilizar para chegar a uma solução baseada na aplicação ODK.

Após pesquisa no âmbito da plataforma a utilizar pode-se verificar a existência de plataformas como, o *Xamarin* , o *IONIC*, o *Outsystems*, *Android Studio* e *React Native*, tendo estas vantagens e desvantagens, as quais irão ser abordadas de seguida.

A plataforma *Xamarin* é utilizada para desenvolver aplicações para a maioria dos dispositivos móveis com a variedade de sistemas operativos conhecidos. Nesta plataforma é possível desenvolver para *Android*, *iOS*, *tvOs*, *whatchOs* e *windows*, utilizando a linguagem *C#*.

Ao desenvolver na plataforma *Xamarin*, os criadores da mesma garantem que, a interface com o utilizador comportar-se-á da mesma maneira para todos os sistemas operativos.

A plataforma *IONIC* é um kit de ferramentas de código aberto, que permite a criação de aplicações para dispositivos móveis e para *desktop*. O *IONIC* utiliza tecnologias como, o HTML, o CSS e o *JavaScript*. A plataforma *IONIC* concentra-se na experiência do utilizador a nível de UI ou UX de uma aplicação (controladores, interacções, gestos e animações). A estrutura das bibliotecas disponibilizadas por esta plataforma vai ao encontro das disponibilizadas pela plataforma *Android*. Esta pode ser utilizada para desenvolver aplicações quer para *Android* quer para *iOS*.

A *OutSystems* é uma empresa de *software* nascida em Portugal com sede em Atlanta, EUA. Foca o seu negócio numa plataforma de ferramentas CASE - software de apoio à concepção e implementação de outro software. A empresa disponibiliza uma plataforma, cujo nome também é *outsystems*, que permite desenvolver de forma eficiente tanto a nível do *backend*, como do *frontend*. Esta plataforma tem uma dependência (lock-in) do *Xamarin*, mas o código gerado é *standard*, isto é, HTML, CSS, *JavaScript*, *Java*, *C#* e SQL. Esta plataforma não permite gerar ecrãs de uma forma dinâmica, ele apenas permite que os ecrãs sejam previamente definidos no desenvolvimento das aplicações.

A plataforma *Android Studio* é o ambiente de desenvolvimento integrado, oficial, para o desenvolvimento de aplicações *Android* e é baseado no *IntelliJ IDEA*. O *Android Studio* permite três linguagens de programação, *Java*, *Kotlin* eC++. Esta plataforma disponibiliza a funcionalidade de gerar novos ecrãs programaticamente, lançando uma nova *activity* com os dados necessários e desejáveis.

	Caso de Utilização	Ator	
		Utilizador	Administrador
Aplicação Web	Aceder à página de Login	X	
	Aceder à página de Registo	X	
	Criar Empresas		X
	Criar Categorias		X
	Gestão de Utilizadores		X
	Upload de Formulários	X	X
	Ver lista de formulários preenchidos	X	X
	Exportação de dados recolhidos pelos formulários	X	X
Aplicação Móvel	Aceder à aplicação	X	
	Sincronizar os formulários que provêm do servidor	X	
	Preencher Formulários	X	
	Configurar o servidor	X	
	Submeter Formulários	X	
	Envio automático para a base de dados remota	X	

Tabela 3.2: Casos de Utilização.

A *framework React Native* é baseado em *React*, como o nome indica, foi desenvolvido pela equipa do *Facebook*, que possibilita o desenvolvimento de aplicações móveis, tanto para *Android*, como para *IOS*, utilizando apenas *JavaScript*. Esta *framework* converte todo o código desenvolvido para a linguagem nativa do sistema operacional, esta caraterística é que torna-a diferente das já existentes. [29]

3.3 Análise e desenho

De modo a desenvolver uma plataforma móvel para captura de inquéritos de informação *online/offline* para demonstrar o funcionamento da plataforma e dos conceitos aplicados, foi necessário elaborar diagramas para que o desenvolvimento se torne mais simples.

Esta plataforma é constituída por um conjunto de funcionalidades. Como foi referido ao longo do trabalho, as funcionalidades estão associadas a casos de utilização que expressam os requisitos funcionais da aplicação.

Nesta fase foi efetuado o levantamento dos requisitos e dos casos de utilização que estão associados a este projeto. A Tabela 3.2 apresenta os casos de utilização que foram considerados para o protótipo e que serão explicados de seguida.

3.3.1 Contexto geral

O diagrama de contexto representa todo o sistema como um único processo e é composto por fluxos de dados que mostram as relações entre o sistema e as entidades (atores).

O diagrama de contexto é a representação geral do sistema, destacando [41]:

- **Entidades** – agentes externos ao sistema que interagem com ele, gerando estímulos e recebendo respostas.
- **Fluxos** – identificação dos tipos de estímulos e respostas entre as entidades do sistema.

A Figura 3.3 ilustra o diagrama de contexto que se propôs realizar.

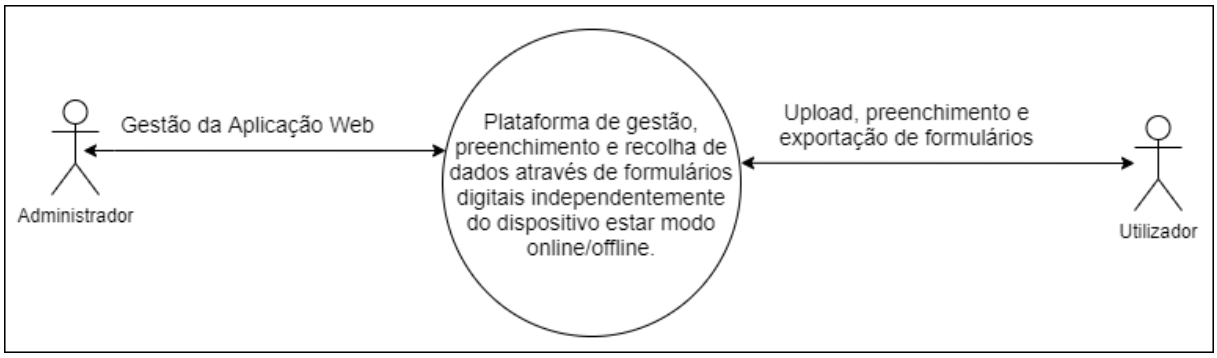


Figura 3.3: Diagrama de Contexto da Plataforma de gestão, preenchimento e recolha de dados através de formulários digitais.

3.3.2 Casos de Utilização

O Diagrama de Casos de Utilização especifica o comportamento e os aspetos envolventes do sistema em termos de casos de utilização e de atores. Este diagrama fornece um modo de descrever a visão externa do sistema e suas interacções com o mundo exterior, representando uma visão de alto nível da funcionalidade do sistema.

Tendo em conta que se pretende criar uma plataforma para um dispositivo móvel a fim de capturar informação através de inquéritos em modo *online/offline*, foram identificados apenas dois atores e algumas acções que se efetuam no sistema e procedeu-se à exposição dos casos de utilização recorrendo aos diagramas de casos de utilização.

De seguida, são apresentados dois casos de utilização, um para a aplicação *web*, que serve como interface para fazer toda a gestão e outro para a aplicação móvel, dedicada à recolha dados.

Na Figura 3.4 foi colocado um ator, nomeando de Utilizador/Administrador, sabendo que o administrador tratará da gestão da plataforma e o utilizador terá as funções já referidas anteriormente na Tabela 3.2. Na Figura 3.5, o ator que é apresentado é um

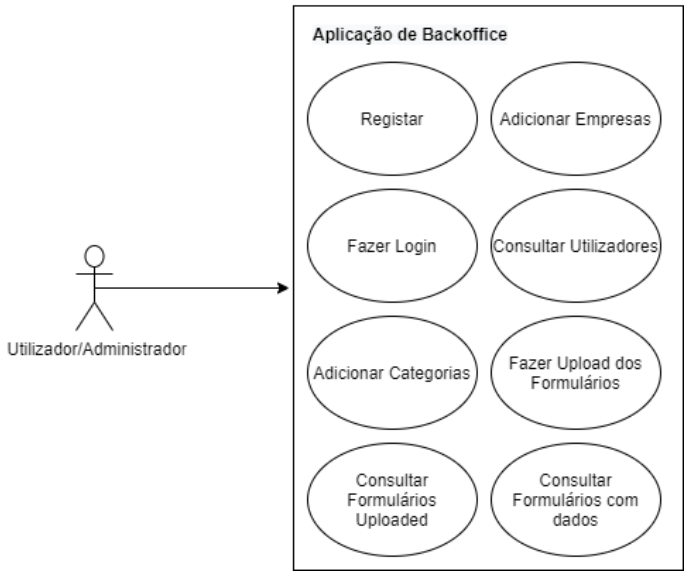


Figura 3.4: Modelo de Casos de Utilização da aplicação servidora.

utilizador dito normal, não tem permissões de administração.

3.3.3 Especificação de Casos de Utilização

Um caso de utilização descreve o que faz um sistema (ou parte deste), mas não como tal é realizado. O foco é, portanto, na visão externa do sistema, ou seja, na visão que os utilizadores têm dele. A especificação de casos de utilização detalha a operação dos casos de utilização anteriormente identificados.

De seguida, são especificados casos de utilização da plataforma móvel para captura de inquéritos de informação *online/offline*, nomeadamente "Aceder à página inicial", "Fazer Upload de Formulários no Servidor"e "Preenchimento de um Formulário da aplicação móvel".

De seguida são indicadas as operações associadas ao caso de utilização "Carregamento de Formulários", este caso de utilização ocorre na aplicação web. Na Figura 3.7 é exibido o diagrama de atividade e na Figura 3.6 são descritas as operações.

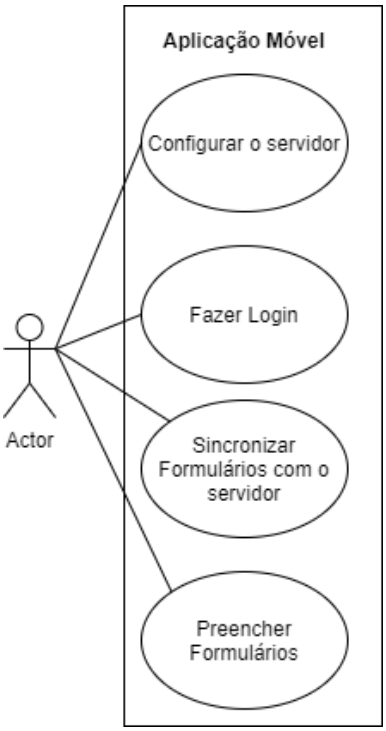


Figura 3.5: Modelo de Casos de Utilização da aplicação móvel.

Por último são apresentadas as operações associadas ao caso de utilização "Preenchimento de um Formulário da aplicação móvel", este caso de utilização ocorre na aplicação móvel, na Figura 3.9 é exibido o diagrama de atividade e na Figura 3.8 são descritas as operações deste caso de utilização.

3.4 Arquitetura

A arquitetura proposta está dividida em três partes distintas, como se pode verificar na Figura 3.10, o bloco do lado esquerdo designado de **aplicação de *backoffice***, o bloco do lado direito nomeado de **Aplicação Móvel** e ainda a parte da nuvem que contém a base dados, *Firebase*.

O sistema é composto por duas bases de dados distintas *Firebase* e *SQLite*.

O *SQLite* foi a opção para guardar os dados que são essenciais ao funcionamento da aplicação. Era desejável que estes dados não fossem acedidos através de outras aplicações, por esta razão optou-se por não se usar o armazenamento interno nem o externo do dispositivo. Uma outra razão por se ter escolhido este tipo de armazenamento, é por este suportar operações transacionais e ter a capacidade de tratar falhas de memória e erros de disco. O *SQLite* é a base de dados que é suportada pela maioria dos dispositivos móveis.

Nome: Carregamento de formulários na aplicação *web*.

Descrição: Os formulários são armazenados numa base de dados, mas para isso teve de se criar uma interface com o utilizador para que este conseguisse fazer a gestão de todos os formulários, sobretudo o carregamento dos mesmos.

Cenário Principal:

1. O utilizador acede à página de Upload *Forms*.
2. O utilizador introduz a categoria, a empresa, a descrição, a versão e o(s) formulário(s).
3. O utilizador carrega no botão de "*submit*".
4. O formulário é guardado no repositório remoto.

Cenário alternativo 1:

5. No passo 2 os dados não existem, por exemplo, a categoria e a empresa.
6. O administrador tem de criar para que os dados possam ser escolhidos corretamente.

Figura 3.6: Caso de utilização - Carregamento de Formulários.

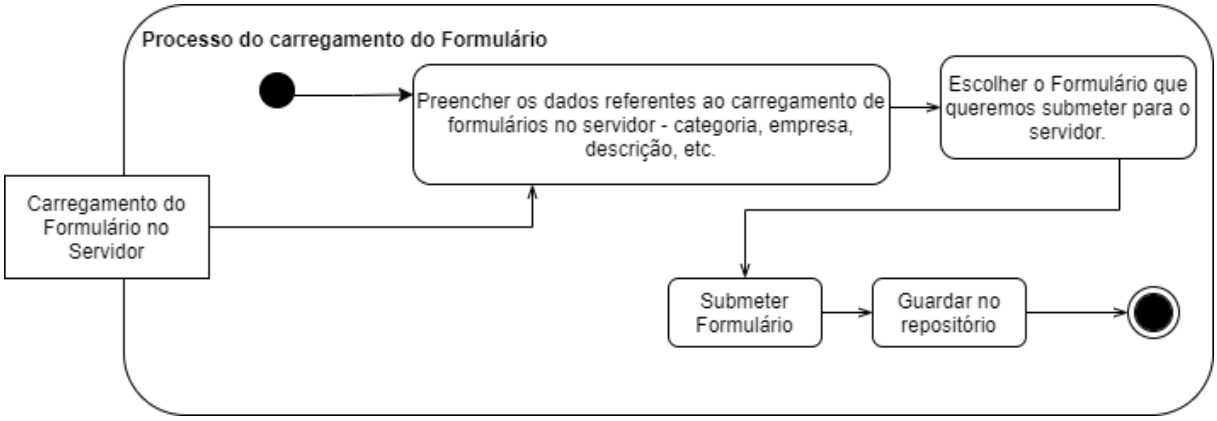


Figura 3.7: Diagrama de atividade - Carregamento de Formulários.

Ao longo do preenchimento dos formulários existe a necessidade de ir guardando os dados que vão sendo recolhidos. Como a informação presente nos vários formulários não é constante, isto é, o número de campos presentes nos formulários assim como o seu tipo alteram, não é possível criar uma base de dados relacional, então recorreu-se a uma base de dados não relacional, a *Firebase*.

Esta base de dados foi a escolhida, pois permite o armazenamento de objetos JSON resolvendo assim o problema do número e tipo de campos descrito anteriormente, assim como a sincronização de dados entre a base de dados local e a base de dados remota. Toda a gestão, se existe ou não uma conexão estável à *internet*, é feita de forma transparente para o utilizador.

A *Firebase* é que decide se existem condições para fazer a sincronização de dados sem que haja perda de informação.

Nome: Preenchimento de um Formulário da aplicação móvel.

Descrição: Quando o utilizador acede à lista de formulários no dispositivo móvel e carrega num *item* da lista, seleciona o formulário, o formulário é mostrado numa outra página, nova *activity*.

Cenário Principal:

1. O utilizador acede à página de *login* da aplicação móvel.
2. Acede ao ecrã com a lista de formulários, a lista apenas contém os nomes dos ficheiros que foram inseridos pela aplicação *web* no servidor.
3. Sincroniza a lista de formulários com o servidor, caso haja conectividade com a *Internet*. Todos os dados referentes aos formulários são sincronizados.
4. O utilizador escolhe o formulário que pretende responder.
5. É criada uma instância do formulário que o utilizador demonstrou a intenção de preencher. Este formulário continuará disponível para ser escolhido quantas vezes o utilizador desejar.
6. A descrição do formulário é carregada para memória construindo assim um objeto. Este objeto é interpretado pela aplicação móvel construindo os *widgets* dinamicamente, consoante os campos descritos no objeto.
7. Mediante o preenchimento dos dados, estes são guardados localmente na base de dados. A sincronização para a base de dados remota é executada quando o mecanismo de sincronização disponibilizado pela base de dados achar que é a altura certa para o fazer.

Cenário alternativo 1:

8. No passo 1 os dados são inválidos.
9. O sistema mostra uma mensagem a informar o utilizador desse facto e o utilizador deve-se registar primeiramente na aplicação *web*, de modo a que consiga efetuar o *login* na aplicação móvel.

Cenário alternativo 2:

10. No passo 3 não existe ligação com a *Internet*.
11. Apenas consegue usufruir dos formulários sincronizados previamente.

Figura 3.8: Caso de utilização - Preenchimento de um Formulário da aplicação móvel.

3.4.1 Aplicação para *upload* e gestão de formulários e utilizadores

A aplicação de *backoffice* tem que disponibilizar uma *API* para que este possa ser utilizados por diferentes tipos de cliente, neste caso para que haja comunicação com a aplicação móvel.

O desenvolvimento de uma aplicação de backoffice web, deveu-se ao facto de ser necessário fazer a administração de utilizadores da aplicação móvel, assim como, a gestão dos formulários que poderiam ser usados e a exportação de dados já adquiridos através do preenchimento dos inquéritos.

O desenvolvimento da aplicação de *backoffice* teve em conta o padrão *MVC*, Figura 3.11. Houve a necessidade, na parte do modelo, de criar uma camada de acesso a dados, onde contém uma camada responsável por comunicar com a base de dados e outra de transferir objectos para o controlador.

A *API* disponibilizada tem o objectivo da aplicação móvel conseguir adquirir os formulários carregues na aplicação web assim como sincronizar os utilizadores registados na mesma.

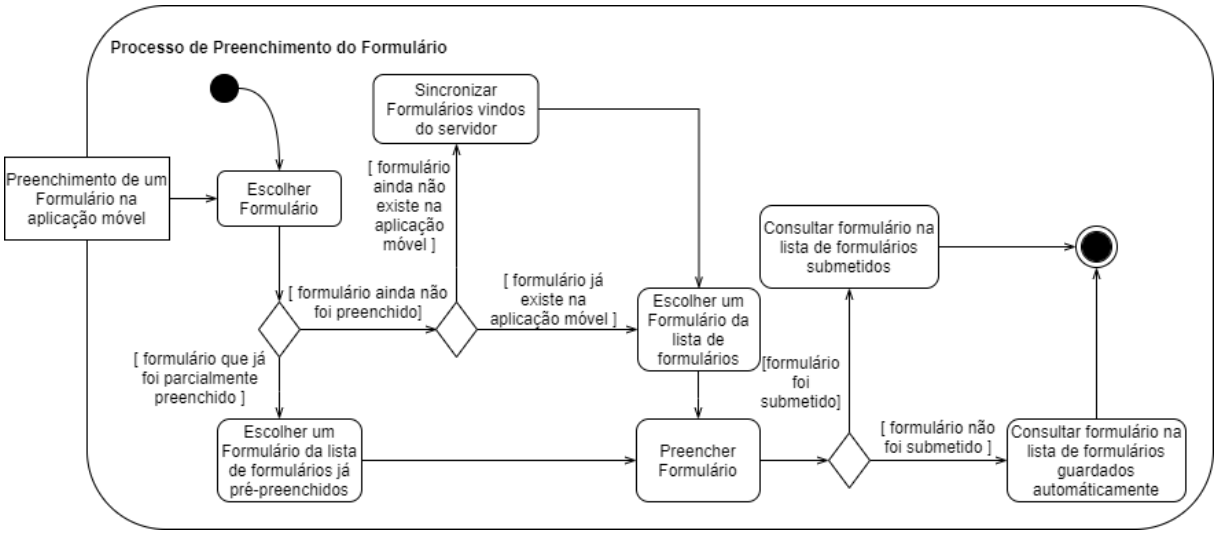


Figura 3.9: Diagrama de atividade - Preenchimento de Formulários.

Esta aplicação apenas interage com a base de dados *Firebase*, sendo que esta está alojada na *cloud*. Posteriormente, será abordada a estrutura de dados utilizada na base de dados, assim como outros aspectos.

Na Figura 3.12, são apresentados dois tipos de interação com a aplicação web: Registo/Login de utilizadores e carregamento de formulários.

Registo/Login de utilizadores

- (1) O utilizador, que nunca entrou na aplicação, expressa a intenção de se registar na aplicação preenchendo o utilizador, palavra-passe e empresa. Entra dentro da rota do controlador.
- (2) O controlador invoca a função de criação de um utilizador.
- (3) É verificada a existência do utilizador. Dependendo da resposta do ponto (4) é criado o utilizador ou não, isto é, se o utilizador ainda não existir é criado na base de dados.
- (5) O utilizador que se acabou de registar é instanciado na função inserção.
- (6) É criado um objecto JSON com a informação do utilizador e com a mensagem de sucesso ou insucesso.
- (7) A informação é passada de modo a que seja possível decidir se é apresentado um mensagem de alerta de insucesso ou se o utilizador é reencaminhado para a página principal da aplicação.

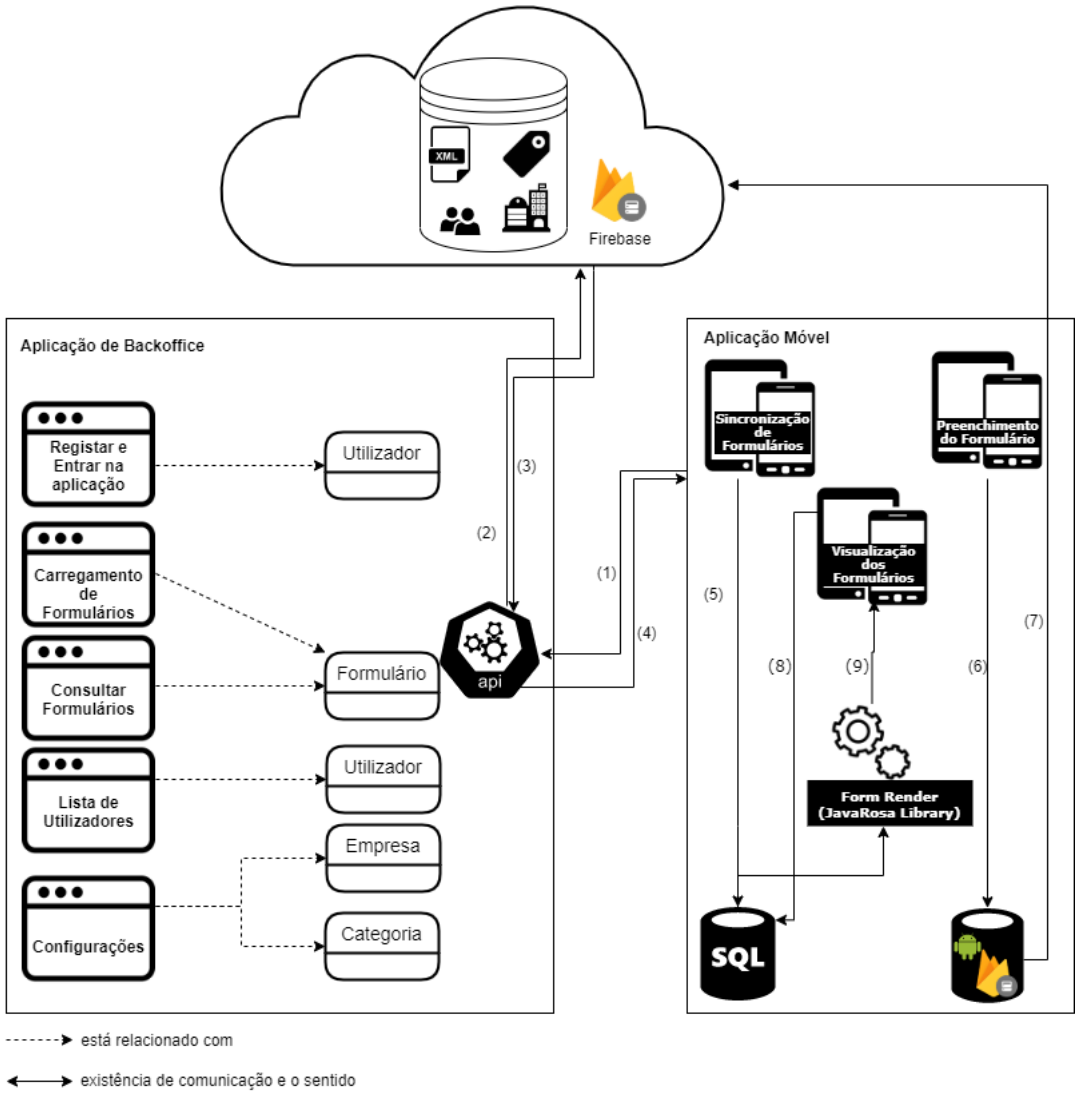


Figura 3.10: Diagrama da arquitetura proposta.

Carregamento de Formulários

É necessário ter em conta que, antes de se fazer o carregamento de formulários para o servidor, se existe a informação essencial para o fazer. Os dados que são indispensáveis são os campos empresas e categorias, estes devem ser criados previamente. Os formulários são definidos recorrendo à linguagem XML tendo em conta a especificação *XForms*. Estes formulários são definidos em duas partes:

- o modelo - descrevendo os dados e relações que possam existir com outros campos.
- a interface com o utilizador - descrição de conteúdo e valores.

Se o utilizador verificar que os dados presentes não são suficientes então os passos

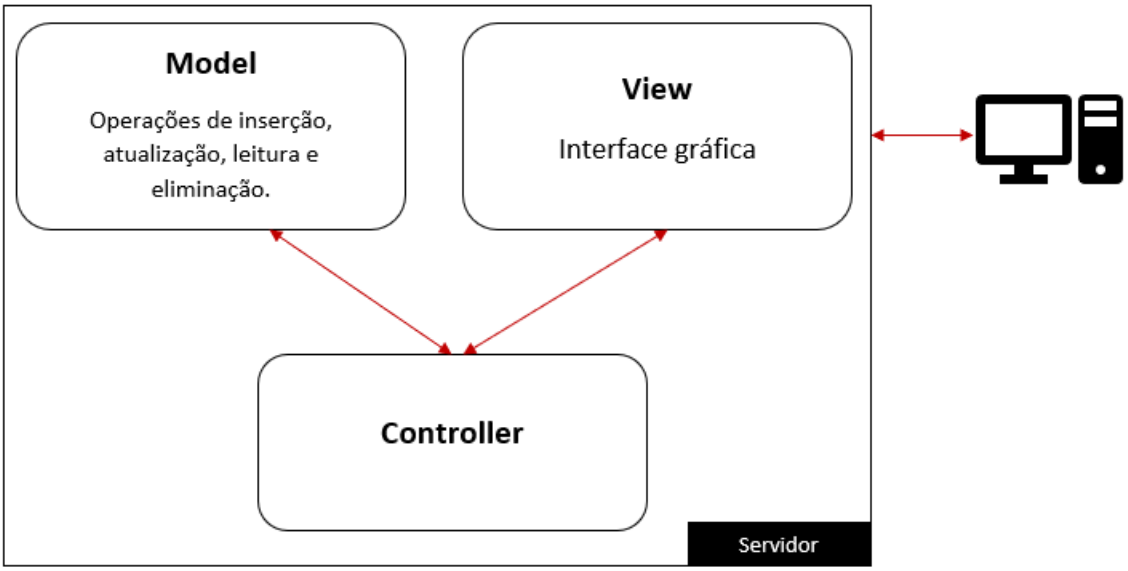


Figura 3.11: Padrão de desenvolvimento MVC.

executados são:

- (8) Envio de categoria preenchida para o controlador.
- (9) O controlador invoca a função de criação.
- (10) Insere na base de dados.
- (11) Retorna a categoria inserida.
- (12) Cria objeto que é retornado pela base de dados.
- (13) Passagem do objecto criado para o controlador.
- (14) O controlador envia a informação para ser disponibilizada ao utilizador.

Este sistema apresentado anteriormente é idêntico para a criação de empresas o que diferencia é só mesmo a entidade.

Após esta pequena configuração o utilizador já pode executar o carregamento de formulários para o servidor este processo é feito da seguinte forma:

- (15) Depois de preencher os dados necessários e carregar *submit* o pedido é reencaminhado para o controlador especifico.
- (16) É invocada a função de criação do formulário.

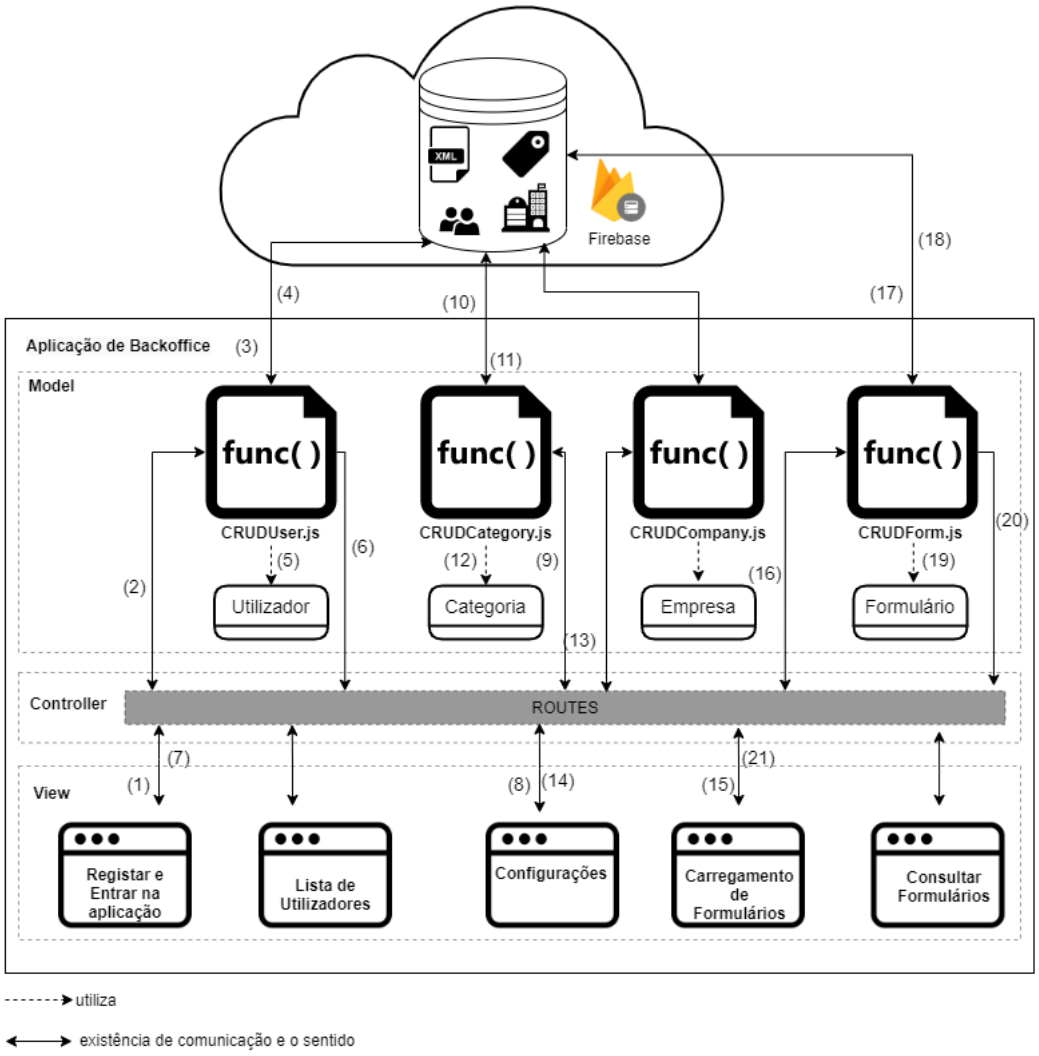


Figura 3.12: Diagrama da arquitetura da aplicação de *backoffice*.

- (17) Os dados referentes ao formulário são guardados na base de dados.
- (18) Resposta por parte da base de dados sobre o sucesso do pedido realizado.
- (19) É instanciado um objecto *Form* com a informação retornada.
- (20) O objecto é passado ao controlador e é disponibilizado depois na lista de formulários, que se pode consultar num outro menu da aplicação.

3.4.2 Aplicação Móvel

Na aplicação móvel são usadas duas bases de dados, *SQLite* e *Firebase*, como já foi mencionado anteriormente. A base de dados *SQLLite* conterá os utilizadores que já fizeram *login* na aplicação naquele dispositivo no momento que houve conexão à *internet*.

Neste repositório também é guardada a descrição do formulário, esta permanece na base de dados *SQLite* porque assim, no caso da conexão à *internet* ser instável ou inexistente, conseguimos iniciar o preenchimento dos formulários já existentes no dispositivo móvel.

A preocupação inicial do preenchimento de formulários em dispositivos móveis era como guardar os dados, devido a não saber quantos campos existia no formulário e o tipo dos mesmos. Visto que, os formulários podem conter inúmeros campos de variadíssimos tipos, não seria fácil de construir uma estrutura de dados na base de dados *SQLite* de modo a suportar esta situação, então recorreu-se à base de dados *Firebase*.

Esta base de dados, assim como o *SQLite*, está disponível para serem usada em dispositivos móveis, a diferença é que a *SQLite* já está incorporada no sistema operativo e a *Firebase* tem que se instalar. Este repositório de dados, guardará a informação recolhida pelos formulários, fazendo a sua própria gestão, de quando é que é possível enviar os dados para a base de dados remota. Se o dispositivo estiver no modo *offline*, os dados irão ser guardados localmente sem haver a sincronização com a base de dados que está na *cloud*.

Os formulários, *XForm*, até serem expostos para o utilizador é necessário que haja transformações. Na aplicação móvel é utilizada a biblioteca *javarosa* [11], que permite descobrir qual o tipo do campo, deste modo consegue-se proceder à renderização do formulário *XForm*, de acordo com a sua especificação. O formulário é carregado para memória onde é efetuado um conjunto testes para que o campo seja renderizado de forma correta.

Na Figura 3.13, existem algumas setas, estas estão numeradas de 1 a 9. Segue-se uma breve explicação sobre estas interacções.

Sincronização de formulários

- (1) Faz um pedido à API, realizada na aplicação de *backoffice*, assim obtém-se todos os formulários carregados no sistema para a empresa a que o utilizador pertence.
- (2) A API é responsável pela execução de pedidos à *Firebase*, estes são realizados através das operações disponibilizadas por parte da base de dados.
- (3) A base de dados retorna um objeto JSON com a informação.
- (4) A informação obtida é enviada para a aplicação móvel, esta mostra a informação em forma de lista.

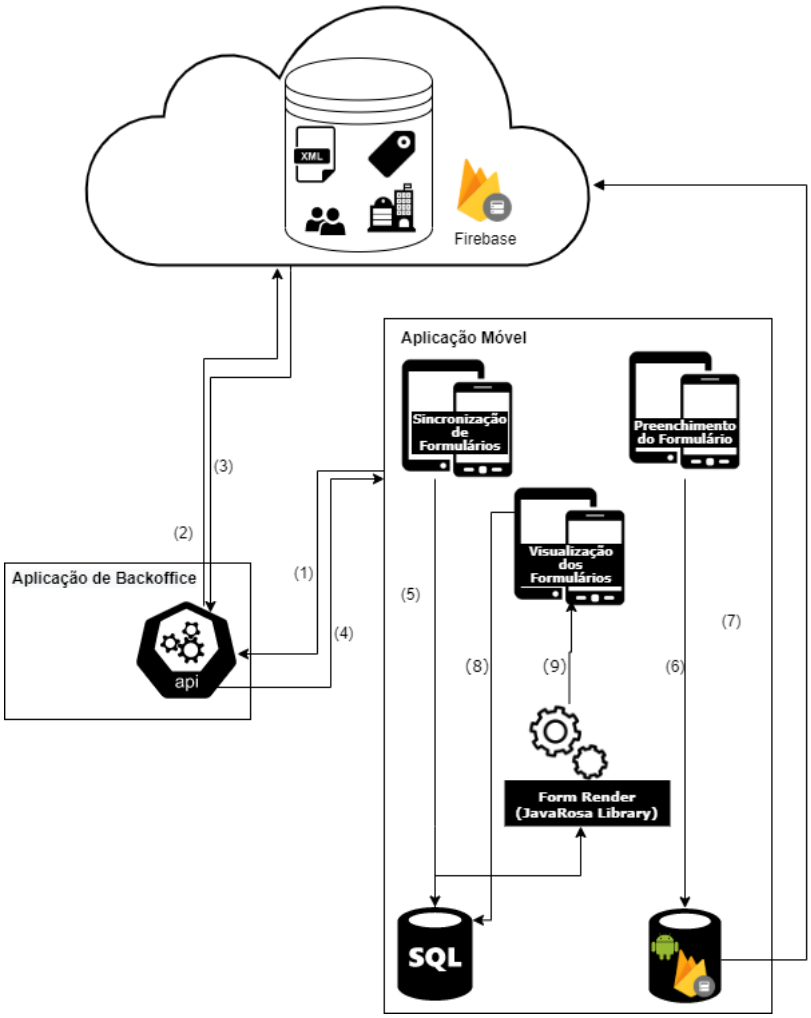


Figura 3.13: Diagrama da arquitetura da Aplicação Móvel.

- (5) A aplicação móvel guarda a descrição dos formulários na base de dados *SQLite*, para quando não existe conexão à *internet* ser possível preencher formulários na mesma.

Visualização dos formulários

- (8) é realizado um pedido à base de dados *SQLite* para obter a descrição do formulário. Este pedido é feito a esta base de dados porque o dispositivo móvel pode não ter conexão à *internet*. Desta forma não se corre o risco de ter de realizar um novo pedido.

- (9) Neste ponto existe um mapeamento entre a descrição do documento *XForm* com os *widgets* produzidos no dispositivo móvel. Através da biblioteca *JavaRosa*, é possível este mapeamento e gerar um formulário com vários *widgets*.

Preenchimento dos formulários

- (6) À medida que o formulário é preenchido os valores dos campos são guardados localmente no dispositivo móvel. Após terminar o preenchimento do formulário os eventos existentes na *Firebase* irão averiguar se existe todas as condições necessárias para sincronizar o repositório remoto.
- (7) Envio dos dados para o repositório remoto.

3.4.3 Base de Dados na *Cloud - Firebase*

A base de dados *Firebase*, tanto é usada no **aplicação de *backoffice*** como na **aplicação móvel**.

Esta base de dados, que está alojada na *cloud*, também guarda a descrição *XForm*, os utilizadores registados, as categorias de formulários, as empresas que possam usar a aplicação e por último os dados recolhidos através dos formulários, estes conceitos foram abordados na Secção 3.3.

Optou-se pela *Firebase* pois esta é adequada, tendo em conta que manipula os dados tanto no modo *offline* como no modo *online*.

Nesta base de dados só se guardará uma identificação para o formulário que foi criado e os dados que já foram preenchidos, e quando houver necessidade far-se-á uma junção entre a descrição do formulário guardado no *SQLite* e os dados armazenados na *Firebase*, visto que o repositório tem custos, desta maneira, não duplicando dados conseguimos reduzir recursos. A junção apenas será necessária quando queremos voltar a apresentar o formulário já com os dados que foram preenchidos.

A base de dados mencionada anteriormente suporta a comunicação com a aplicação móvel como seria pretendido, tendo uma chave associada a um documento, o acesso é direto. A base de dados disponibiliza um conjunto de operações, que permite manipular, criar, atualizar e eliminar dados.

As aplicações que utilizam a base de dados *Firebase*, normalmente é porque se deparam com o problema de interrupções temporárias na rede ou devido ao modelo de dados, que neste caso é orientado ao documento e por isso é mais flexível quando ao tipo e número dos campos.

Todas as operações na aplicação, que são realizadas através da *Firebase*, são mantidas na instância da *Firebase* alojada no dispositivo móvel, enquanto o dispositivo estiver *offline*, quando o dispositivo recupera a conetividade, todas as operações são enviadas

para o servidor da base de dados. A *Firebase* contém duas bases de dados disponíveis para se usar a *Firebase Storage* e a *Firebase Realtime Database*, nesta aplicação a utilizada foi a *Firebase Realtime Database* [27].

A comunicação estabelecida entre a base de dados na *cloud* e do dispositivo móvel, é através das operações disponibilizadas pela *Firebase*, mas a sincronização entre elas é totalmente transparente para o utilizador e em questões programáticas só se tem que invocar uma função a referir que as bases de dados devem-se manter sincronizadas.

4

Implementação e Configuração

Este capítulo descreve os aspectos de implementação que foram utilizados ao longo do desenvolvimento da aplicação. No final do capítulo, são apresentados os passos das configurações necessárias para o funcionamento de toda a aplicação.

A aplicação de *backoffice* está dividido em três partes, *frontend*, *backend* e aplicação móvel. Para desenvolver a interface com o utilizador, parte cliente, decidiu-se utilizar a *framework Angular8* e para a parte servidora utilizou-se *JavaScript*. A comunicação entre o *frontend* (client browser) e o *backend* é realizada através de uma API HTTP, com respostas na forma de objetos JSON.

4.1 Aplicação de *Backoffice*

A aplicação de *backoffice*, é uma aplicação *web*. Esta aplicação *web* é dividida em duas componentes como mencionado anteriormente.

A aplicação *web*, serve para registar utilizadores, fazer *upload* de formulários, criar categorias para os formulários e por fim criar empresas de modo a que várias empresas possam utilizar esta aplicação.

4.1.1 Registo de clientes e armazenamento de credenciais

O utilizador que realize o primeiro registo na aplicação para uma determinada empresa é atribuído o *role* de administrador da aplicação. Um administrador na aplicação

de *backoffice* consegue aceder ao tabulador de *"**Settings**"*, de modo a criar *"**Companies**"* e *"**Categories**"*. Estes atributos serão aplicados quando for feito o carregamento de formulários para o sistema. Este tipo de utilizador também tem o privilégio de fazer a gestão dos restantes utilizadores da empresa.

Um utilizador dito normal, isto é, não contém o *role* de administrador, consegue fazer o carregamento dos formulários e exportar os dados preenchidos por formulário. Este consegue ainda visualizar a lista de utilizadores mas sem fazer a sua gestão.

A distinção entre estes dois tipos de utilizadores foi implementada apenas adicionando um atributo do tipo *booleano* que se tiver o valor *true* é administrador, caso contrário é um utilizador normal. Esta diferenciação só existe na aplicação *web*.

Um utilizador para registar-se na aplicação precisa de conceder três dados: o *"username"*, a *"company"* e a *"password"*. Depois da submissão destes dados, a aplicação verifica se existe algum utilizador naquela *"company"* com aquele *"username"*. Se existir é exibida uma mensagem a reportar isso mesmo e não cria nenhum utilizador.

Caso contrário, cria-se esse utilizador, tendo que fazer o *hash* da *password*. O *hash* é uma sequência de bits que tem como objectivo identificar algo. Este *hash* servirá para não guardar a *password* em claro na base de dados, sendo que assim poder-se-á descartar a *password* inserida pelo utilizador. Ao guardar a *password* em claro na base de dados seria uma falha de segurança, por essa razão decidiu-se recorrer a uma função de *hash* para sua proteção. Ao utilizar um *hash* não há a necessidade de chaves e, existe a garantia da consistência, visto que, se introduzir a mesma *password* será gerado o mesmo *hash*.

Um dos problemas do *hash* é ser vulnerável a ataques de dicionário. Um ataque de dicionário é onde um atacante tem uma base de dados com *passwords* prováveis com seus respectivos *hashs*. A solução existente para este problema é colocar um *salt*, isto é, uma porção aleatória de texto que é concatenado com a senha original, e assim sendo torna-se mais difícil para o atacante descobrir a *password*.

O algoritmo utilizado para este *hash* é o *SHA-512*, o que diferencia este algoritmo de todos os outros existentes é o tamanho que é suportado da mensagem de entrada.

Neste caso optou-se por utilizar um *hash*, pois o maior erro que pode ser cometido é guardar as *password* dos utilizadores em texto puro, o que faz com que um possível roubo de dados acarrete problemas ainda maiores.

Todos os dados recolhidos através desta aplicação *web* são guardados na base de dados *Firebase*.

4.1.2 Aplicação Cliente

Designa-se aplicação cliente a toda a interface que o utilizador visualiza. Esta componente da aplicação de *backoffice* foi desenvolvida utilizando *Angular8*. O *Angular8* junta *TypeScript*, *HTML* e CSS numa só plataforma.

A interface com o utilizador é constituída por um menu lateral contendo todas as funcionalidades disponíveis da aplicação, como se pode observar na Figura 4.1. Nesta mesma figura tem-se as funcionalidade numeradas de um a cinco com diferentes cores. O menu designado por **"My Dashboard"**, foi pensado para conter todos os formulários

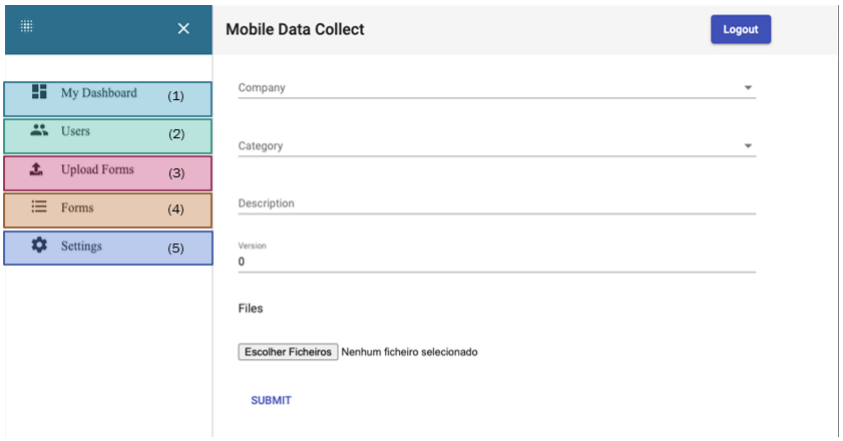


Figura 4.1: Interface com o utilizador da aplicação WEB.

que já tenham sido submetidos extraindo os dados preenchidos. Então, nesta área foi

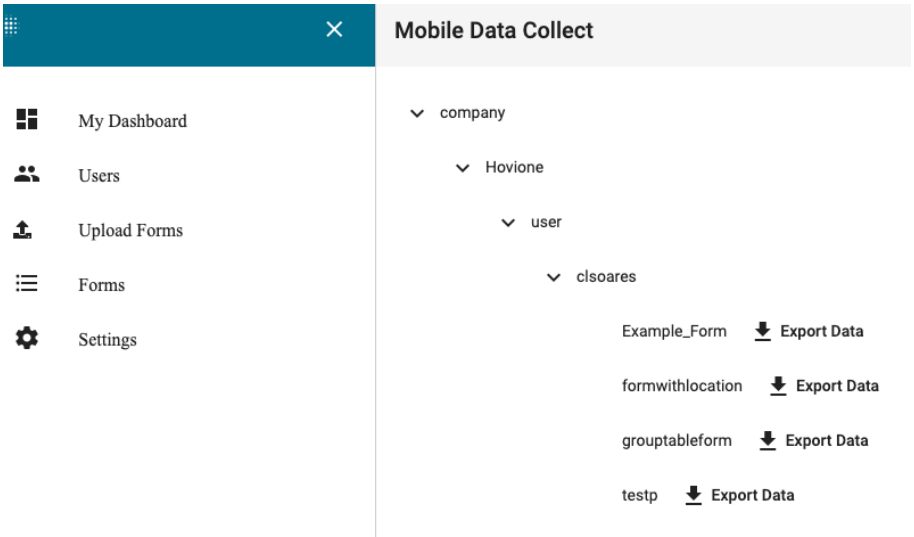


Figura 4.2: Exibição de formulários para extracção de informação recolhida.

criada uma estrutura que representasse todos os formulários que já tivessem sido preenchidos através dos dispositivos móveis de modo a que quando o utilizador quisesse

```
{
   "data":[
      {
         "Age":"26",
         "Algarve":false,
         "Lisbon":true,
         "Name":"claudia",
         "Porto":true,
         "createdOn":"Fri Nov 27 19:27:29 GMT+00:00 2020"
      },
      {
         "Age":"22",
         "Name":"ola",
         "createdOn":"Thu Nov 26 22:47:22 GMT+00:00 2020"
      },
      {
         "Age":"22",
         "Name":"claudia",
         "createdOn":"Thu Nov 26 22:51:29 GMT+00:00 2020"
      },
      {
         "Age":"22",
         "Name":"claudia",
         "createdOn":"Thu Nov 26 23:04:29 GMT+00:00 2020"
      }
   ]
}
```

Listagem 4.1: Dados exportados do formulário grouptableform.

exportar os dados dos formulários fosse necessário apenas carregar no botão *"**Export Data**"*, como podemos ver na Figura 4.2. Para a implementação desta funcionalidade, optou-se por se mostrar um estrutura semelhante à que existe na base de dados. O nó principal será sempre designado por *"company"*, de seguida o nó é o nome da empresa e depois a identificação do utilizador com os respectivos formulários. A formatação escolhida para obter os dados dos formulários foi JSON, podemos ver um exemplo na Listagem 4.1.

O segundo menu, **"Users"** apresenta uma lista de todos os utilizadores registados na aplicação. Este registo é feito antes de fazer o *login* na aplicação, pois só depois de fazer o registo é que é possível entrar na aplicação, quer *web* quer móvel, como já foi referido na subsecção 4.1.1.

O terceiro menu, como o nome indica, **"Upload Forms"**, serve para o utilizador introduzir os formulários que pretender. Neste menu consegue-se introduzir mais do que um formulário, caso estes tenham a mesma categoria, a mesma versão e que pertença a uma só empresa, pois a partir do nome do ficheiro que contem o formulário consegue-se distingui-los um dos outros.

Antes de introduzir qualquer tipo de formulário tem de se ter em conta que se deve criar as empresas e as categorias, para conseguir preencher estes dados do formulário. Estas configurações são feitas no quinto menu cujo nome é **"Settings"**.

Por último, o menu *"Forms"* contém a lista de todos os formulários existentes na aplicação.

4.1.3 Aplicação Servidora

A aplicação servidora é a parte da aplicação que comunica com a base de dados, guardando os dados que foram introduzidos pelo utilizador, e também, tem como funcionalidade disponibilizar a informação, proveniente da base de dados, que é pedida pelos utilizadores.

Como se pode observar na Figura 4.3, existe uma camada de acesso a dados constituída por outras duas camadas: *Data Access Object* e *Data Transfer Object*.

A camada *Data Access Object* comunica diretamente com a base de dados, isto é, esta é a camada que invoca operações da *Firebase*, quer para obter dados, quer para guardá-los. É de salientar, que a comunicação feita com a base de dados *Firebase*, é realizada através da chamada a funções disponibilizadas pela mesma, é assim que o repositório de dados disponibiliza a sua *API*.

Caso o objetivo seja obter dados e disponibilizá-los para o utilizador, existe uma transformação do objeto retornado pela base de dados, transformação essa realizada através da camada *Data Transfer Object*. A camada *Data Transfer Object* comunica diretamente com a camada de apresentação.

A aplicação servidora também disponibiliza uma *REST API*, esta tem como objectivo estabelecer a comunicação entre a aplicação móvel e o servidor, de modo a obter informação sobre os utilizadores e formulários.

4.2 Aplicação Móvel

A aplicação móvel desenvolvida destina-se apenas para dispositivos com o sistema operativo *Android*.

Nesta secção 4.2, é abordado o mecanismo de comunicação com o servidor e as técnicas utilizadas para um formulário *XForm* ser transformado num formulário digital, neste caso num ecrã de um dispositivo móvel com o sistema operativo *Android*.

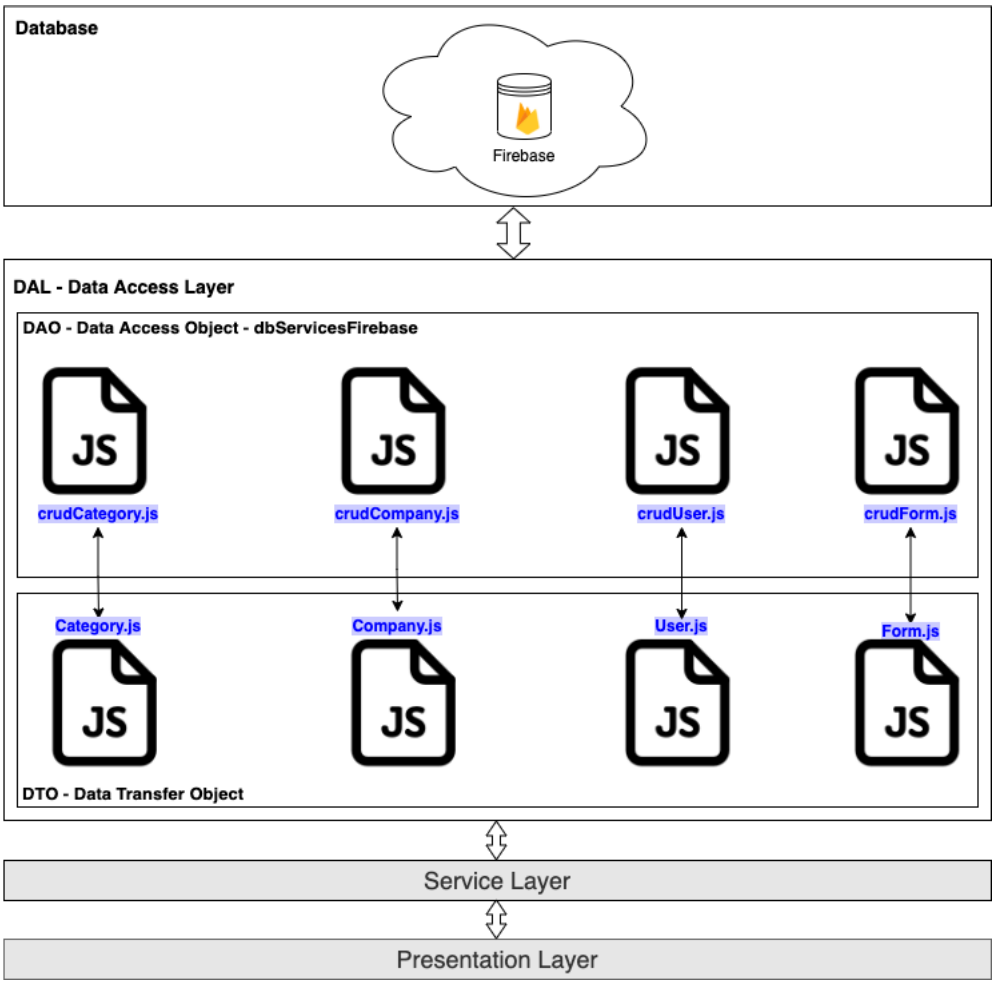


Figura 4.3: Modelo de camadas da aplicação

4.2.1 Comunicação com o servidor

Nesta subsecção são descritos os passos necessários para configurar a comunicação entre a aplicação móvel com o servidor aplicacional.

O servidor, que irá conter a aplicação servidora, carece da instalação do *nodeJS*, que por sua vez já contém o *npm*, que servirá para instalar todos os *packages* necessários ao funcionamento da aplicação servidora.

A comunicação entre a aplicação móvel e a aplicação de *backoffice* é essencial, por isso, foi implementado na aplicação móvel um local onde é possível configurar o endereço do servidor onde está alojada a API que se usará. Para configurar o endereço basta aceder ao menu que é disponibilizado no canto superior direito da aplicação móvel, clicar *Settings* e preencher o campo que tem a *label* de *Connection Server*, como se observa na Figura 4.4. É através desta configuração, que posteriormente se consegue fazer qualquer pedido ao servidor, seja sobre utilizadores ou formulários.

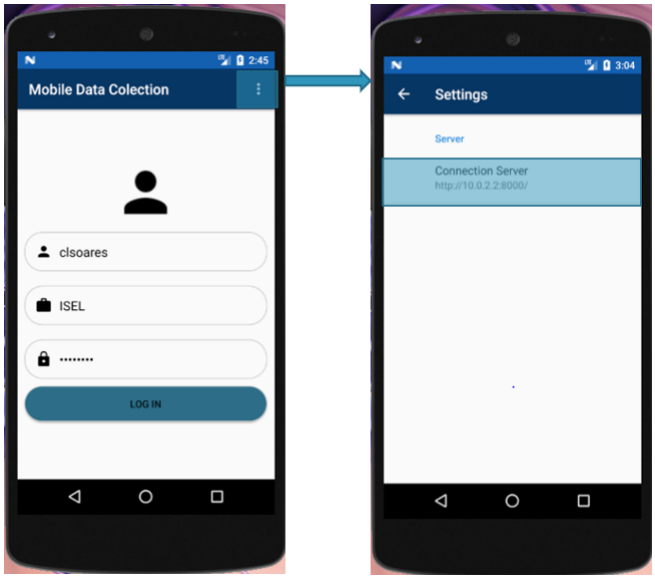


Figura 4.4: Aplicação móvel - Menu "Settings".

4.2.2 Camada de acesso a dados e sincronização

Como se pode constatar na Figura 4.5, para aceder à aplicação móvel é necessário fazer *login*. O primeiro *login* na aplicação num determinado dispositivo móvel deve ser efetuado tendo conexão à *internet*, pois é necessário ler todos os utilizadores registados na aplicação *web*, e que por sua vez, como será o primeiro acesso, também será necessário fazer sincronização dos formulários existentes.

Após este primeiro *login*, apenas será preciso possuir a ligação com a *internet* quando se pretender fazer algum tipo de actualização de utilizadores ou formulários, ou no momento que haja a necessidade de enviar os dados, recolhidos pelos formulários, para a base de dados alojada na *cloud*, pois os utilizadores são armazenados na base dados *SQLLite*, assim como a descrição dos formulários.

4.2.3 Construção dinâmica de formulários

Quando já existe a lista de formulários, como se pode ver na Figura 4.6, para começar o preenchimento de um determinado formulário, então é necessário clicar no formulário que se pretende preencher.

Quando se expressa essa intenção, de começar o preenchimento de um inquérito, a descrição do formulário é carregada para memória através da biblioteca *JavaRosa*, isto pode ver-se na Figura 4.2.

Este código obterá o ficheiro que foi selecionado e o carregamento para memória é

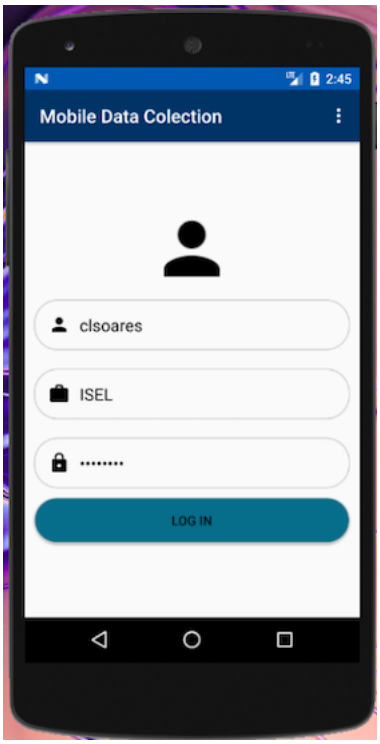


Figura 4.5: Login na aplicação móvel.

```
File formXml = new File(pathNameFile);
String lastSavedSrc = FileUtils.getOrCreateLastSavedSrc(formXml);
formDef = XFormUtils.getFormFromFormXml(pathNameFile, lastSavedSrc);
model = new FormEntryModel(formDef);
FormIndex index = model.getFormIndex();
FormEntryCaption fec = new FormEntryCaption(formDef,index);
FormEntryController form = new FormEntryController(model);
```

Listagem 4.2: Carregamento do formulário para memória.

feitos através da função *getFormFromFormXml*, de seguida é criado um objecto que contem o modelo do formulário que foi carregue, *new FormEntryModel(formDef)*. De seguida, de uma forma recursiva, constrói-se os elementos descritos pelo formulário.

Para saber qual é o *widget* a desenhar, a biblioteca *JavaRosa* tem definido constantes, para cada um dos elementos do *XForm* e através destas constantes consegue-se saber qual é o *widget* que se tem de apresentar.

Este mapeamento é feito com a ajuda de *HashMaps*, para cada *Control Type*, como é designado pela biblioteca *javarosa*, temos associado um *HashMap* diferente, Figura 4.3.

Todas estas coleções são criadas estaticamente, servindo assim toda a aplicação. Para os *HashMaps* associados a cada *Controlo Type* temos como chave a constante *Data Type*, definida também pela mesma biblioteca, e para essa chave temos como valor a classe do *widget* correspondente e assim consegue-se descobrir qual a classe que se tem de

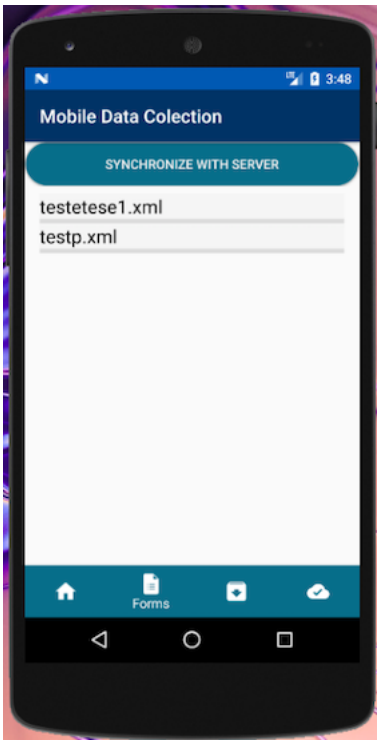


Figura 4.6: Lista de formulários na aplicação móvel.

instanciar, como se pode ver na Listagem 4.4.

A classe que permite a criação do *widget* é instanciada recorrendo à recursividade, como se vê na Figura 4.5.

Primeiramente, ao saber a classe que se quer instanciar, consegue-se obter todos os construtores daquela classe, o "contracto"a seguir por qualquer *widget* que se venha a desenvolver será implementar um construtor com seis argumentos, então consegue-se instanciar invocando o método *newInstance* passando como parâmetro um *array* com os dados necessários.

```
1 contantsControlType.put(CONTROL_INPUT,contantsDataTypeInput);
2 contantsControlType.put(CONTROL_FILE_CAPTURE,contantsGetDataFileCapture);
3 contantsControlType.put(CONTROL_IMAGE_CHOOSE,contantsGetDataImageChoose);
4 contantsControlType.put(CONTROL_OSM_CAPTURE,contantsGetDataOmsCapture);
5 contantsControlType.put(CONTROL_AUDIO_CAPTURE,contantsGetDataAudioCapture);
6 contantsControlType.put(CONTROL_VIDEO_CAPTURE,contantsGetDataVideoCapture);
7 contantsControlType.put(CONTROL_SELECT_ONE,contantsGetDataSelectOne);
8 contantsControlType.put(CONTROL_SELECT_MULTI,contantsGetDataSelectMulti);
9 contantsControlType.put(CONTROL_RANK,contantsGetDataRank);
10 contantsControlType.put(CONTROL_TRIGGER,contantsGetDataTrigger);
11 contantsControlType.put(CONTROL_RANGE,contantsGetDataRange);
```

Listagem 4.3: *HashMap* utilizado para o mapeamento entre as constantes *Control Type* e os *HashMaps*.

```
contantsDataTypeInput.put(DATATYPE_DATE_TIME,DateTimeWidget.class);
contantsDataTypeInput.put(DATATYPE_DATE,DateWidget.class);
contantsDataTypeInput.put(DATATYPE_TIME,TimeWidget.class);
contantsDataTypeInput.put(DATATYPE_DECIMAL,DecimalWidget.class);
contantsDataTypeInput.put(DATATYPE_INTEGER,IntegerWidget.class);
contantsDataTypeInput.put(DATATYPE_GEOPOINT,GeopointWidget.class);
contantsDataTypeInput.put(DATATYPE_GEOSHAPE,GeoshapeWidget.class);
contantsDataTypeInput.put(DATATYPE_GEOTRACE,GeotraceWidget.class);
contantsDataTypeInput.put(DATATYPE_BARCODE,BarcodeWidget.class);
contantsDataTypeInput.put(DATATYPE_TEXT,TextWidget.class);
```

Listagem 4.4: Exemplo de um *HashMap* utilizado para o mapeamento entre as constantes *Data Type* e os *widgets*.

```
private IFormElement recursividade(IFormElement form, FormEntryController
    fec, FormIndex index, DatabaseReference databaseReference){
    List<IFormElement> childrens = form.getChildren();
    if( childrens == null){
        ...
        Class c = hash.get(qd.getControlType()).get(datatype);
        Constructor constructor[] = c.getConstructors();
        QuestionDetails questionDetails = new QuestionDetails(fep);
        Object[] intArgs = new Object[]{this , ll, qd, fep, version,
            databaseReference};
        Class noparams[] = {};
        try {
           Object instance = constructor[0].newInstance(intArgs);
           Method method = c.getDeclaredMethod("getElement", noparams);
           ll = (LinearLayout) method.invoke(instance);
        } catch(Exception e){
           Log.e("Form Activity", e.getMessage());
        }
        return null;
    }
    for(IFormElement child : childrens){
     index = model.incrementIndex(index,true);
     recursividade(child,fec,index, databaseReference);
    }
    return null;
}
```

Listagem 4.5: Recursividade utilizada para criar *widgets*.

```
public class TextWidget{
  private final DatabaseReference databaseReference;
  private LinearLayout screen;
  private TextView tv1;
  private EditText edq;
  long delay = 1000; // 1 seconds after user stops typing
  long last_text_edit = 0;
  Handler handler = new Handler();
  @SuppressLint("ResourceAsColor")
  public TextWidget(Context context, LinearLayout screen, QuestionDef form,
      FormEntryPrompt fep, int version,DatabaseReference databaseReference)
      {
    this.screen = screen;
    this.databaseReference = databaseReference;
    String name = fep.mTreeElement.getName();
    tv1 = new TextView(context);
    tv1.setTextColor(Color.BLACK);
    tv1.setTypeface(Typeface.DEFAULT_BOLD);
    tv1.setText(name);
    edq = new EditText(context);
...
```

Listagem 4.6: Text Widget - exemplo de código.

Após adquirir a instância, tem de se obter o método que se pretende chamar de modo a ter o *widget* pronto a adicionar ao *layout* principal.

Todos os *widgets* são construídos com a mesma lógica. A Figura 4.6 apresenta um exemplo do desenvolvimento de um *widget* para a aplicação *Android*. Quando se tem mais do que um *widget* na descrição do formulário, é necessário ter em consideração que estes tem que ser posicionados de forma a aparecer no ecrã do dispositivo móvel. Então a solução utilizada para esta situação é colocar de forma sequencial no ecrã todos os *widgets*, tendo assim que o *layout* principal possuir um *scroll* para se conseguir ver todos os *widgets*.

De seguida são apresentados todos os *widgets* desenvolvidos, recorrendo sempre aos *widgets Android*, assim como uma breve explicação sobre cada um deles, como foram pensados e desenvolvidos para poderem ser usados:

- *Text*

  Este *widget*, como o seu nome indica é um *input* que permite a introdução de texto. Então, para identificação do campo, é introduzida uma *label* com o nome do mesmo, e depois para usufruir da utilidade do *widget* teve que se introduzir um espaço onde se possa editar o determinado *input*, o componente *EditText* já existente no sistema operativo *Android*. O exemplo deste *widget* pode ser visto no Figura 4.7.

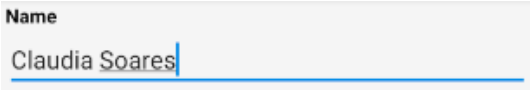


Figura 4.7: *Widget* para introdução de texto.

- *Integer*

  Este componente foi pensado da mesma forma que o *widget Text*, explicado no ponto anterior, a única diferença é que foi adicionada a restrição para só se poder inserir números através da constante **TYPE_CLASS_NUMBER** passada como parâmetro ao método **setInputType**, exemplo na Figura 4.8.

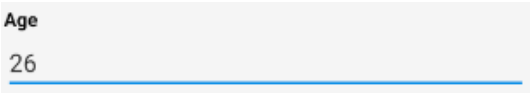


Figura 4.8: *Widget* para introdução de números.

- *Decimal*

  Este componente tem uma implementação idêntica ao *widget Integer* com a diferença da constante passada ao método **setInputType**. A constante passada como parâmetro é **TYPE_NUMBER_FLAG_DECIMAL**.

- *Date Time*

  Este *widget* também contem a *label* para a sua identificação mas depois foi desenvolvido um botão. Quando se clica neste botão é apresentado o **DatePickerDialog**. Este componente é um simples *dialog* que contem um *DatePicker*. Depois, quando se fecha este dialog é despoletado o TimePicker, e quando ambos estão com o dados desejados os dados do formulário deste *widget* são alterados, tanto a data como a hora.

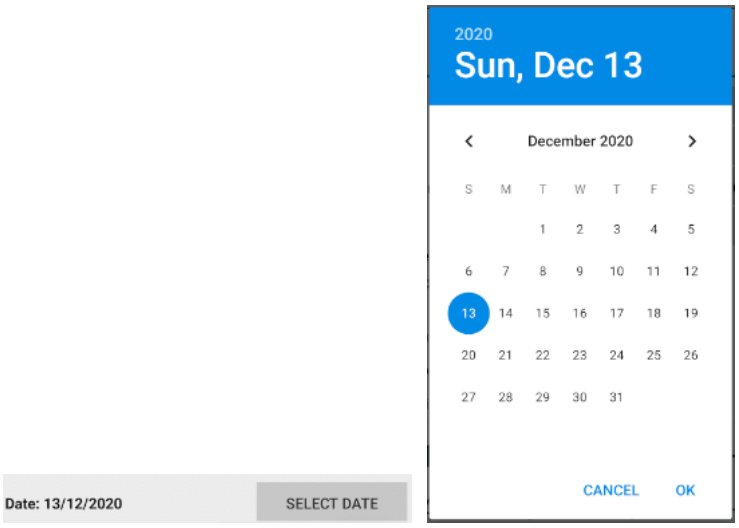
- *Date*

  Este *widget* utiliza parte do *Date Time* explicado anteriormente. A parte utilizada é **DatePickerDialog**. Este *widget* pode ser observado na Figura 4.9.

- *Time*

  Este *widget* também utiliza parte do *Date Time* explicado anteriormente. A parte utilizada é **TimePickerDialog**. A Figura 4.10 é um exemplo deste *widget*.

- *Geopoint*



(a) (b)

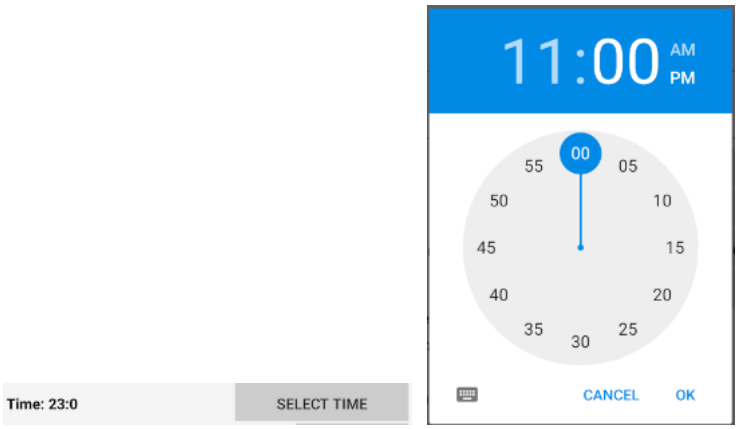
Figura 4.9: (a) Representação do *widget* para introdução de uma data - Botão e informação seleccionada. e (b) Representação do *widget* para escolher a data - Calendário.

Este *widget* serve para fazer a localização actual do telemóvel que está a fazer o preenchimento do formulário. Este utiliza *AlertDialog* onde se pode personalizar a aparecia do mesmo, neste caso com recorreu-se a *api* do *Google Maps* para esta funcionalidade. Depois do lugar estar escolhido são preenchidas as coordenadas de latitude e longitude no formulário, a informação é passada através de um *bundle*. Este *widget* pode ser visualizado na Figura 4.11.

- *SelectOne* - Este *widget* é como se fosse um resposta binária, só existe dois valor possíveis. Por essa razão foi desenvolvido este componente recorrendo à *Checkbox* presente no *andorid*, como se pode ver na Figura 4.12.

- *MultipleItems* - Este componente foi baseado no anterior, só que em vez de se ter uma opção pode-se ter n valores possíveis, sendo que podem estar todos seleccionados ou não. O exemplo deste *widget* está representado na Figura 4.13.

- *SelectMulti* - Este componente também foi baseado no anterior, só que em vez de se poder seleccionar n valores possíveis só se pode seleccionar um, então foi utilizado o **RadioGroup** adicionando quantos **RadioButtons** necessários, com estes elementos garantimos que apenas um é seleccionado, este *widget* está representado na Figura 4.14.



(a) (b)

Figura 4.10: (a) Representação do *widget* para introdução de horas - Botão e informação seleccionada. e (b) Representação do *widget* para escolher as horas - Relógio.

4.3 Configuração das Bases de Dados no projeto

Como já foi referido anteriormente, a aplicação servidora depende de uma base de dados e a aplicação móvel depende de duas bases de dados. A aplicação servidora depende da *Firebase* e a aplicação móvel da *SQLite* e *Firebase*.

4.3.1 Bases de Dados *Firebase*

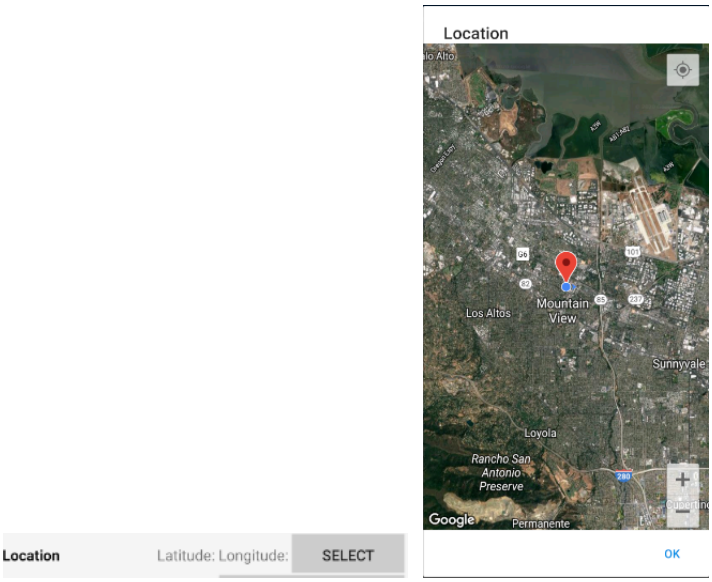
A configuração da base de dados *Firebase* na aplicação *web* é feita em três etapas. Para adicionar a base de dados ao projeto, primeiramente teve de se criar o projeto na *Firebase*, só depois de criar este projeto é que é possível adiciona-lo à aplicação *web*.

A segunda etapa é registar a aplicação *web* na *Firebase*, tendo que navegar para página de visão geral do projeto da Consola do *Firebase* e clicar no ícone da *web* (<>) para iniciar o fluxo de trabalho de configuração, registar a aplicação, introduzir o código exibido na Listagem 4.7, instalar a *Firebase CLI*.

Para instalar o *SDK* do *Firebase* para o *JavaScript* recorreu-se ao *package npm* para a instalação.

De modo a utilizar-se o módulo do *Firebase* teve que se usar o *require* e depois fazer a inicialização com a instrução seguinte: **firebase.initializeApp(firebaseConfig);** o parâmetro *firebaseConfig* é o *object* que se pode visualizar na Listagem 4.7, que contem a *apiKey*, *authDomain* e *databaseURL*, entre outros parâmetros.

Esta base de dados tanto foi usada tanto para a aplicação *web* como para a plataforma



(a) (b)

Figura 4.11: (a) Representação do *widget* para introdução da localização actual - Botão e informação seleccionada, Latitude e Longitude. e (b) Representação do *widget* da localização - Mapa do Google.

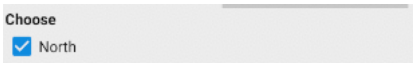


Figura 4.12: Representação do *widget* de resposta binária.

móvel, assim sendo, teve que se fazer a configuração específica para o dispositivo móvel.

A configuração para a aplicação *Andoid*, começa por se ter que fazer um registo da mesma na consola da base de dados e depois aplicação a *Android* só terá contato com a base de dados se se colocar o ficheiro *google-services.json* na diretoria indicada na Figura 4.15, caso contrário, quando se fizer o *build* da aplicação *Android* o *gradlew* não deteta a configuração necessária.

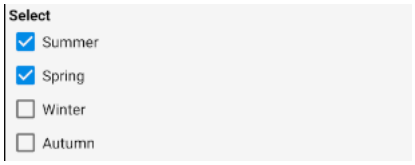


Figura 4.13: Representação do *widget* de escolha de várias uma opções.

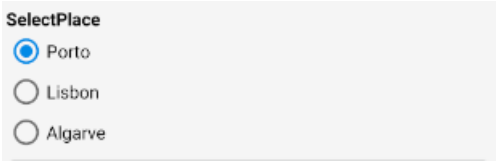


Figura 4.14: Representação do *widget* de escolha de apenas uma opção.

```
const firebaseConfig = {
  apiKey: "xxxxxx",
  authDomain: "mobiledatacolection.firebaseapp.com",
  databaseURL: "https://mobiledatacolection.firebaseio.com",
  projectId: "mobiledatacolection",
  storageBucket: "mobiledatacolection.appspot.com",
  messagingSenderId: "474281143454",
  appId: "1:474281143454:web:a1e75472eb03d102785f8f",
  measurementId: "G-9XBZPW5EDE"
};
```

Listagem 4.7: Objecto para configurar a base de dados - Firebase.

4.3.1.1 Base de Dados SQLite

O sistema operativo *Android* tem embutido a base de dados *SQLite*.

Quando a aplicação é executada pela primeira vez, neste ponto ainda não existe base de dados, tem que se criar as tabelas, índices e relações.

Para se criar, ou até mesmo atualizar o modelo da base de dados, é necessário criar uma classe que estenda de *SQLiteOpenHelper*, sendo que esta classe é responsável por aplicar toda a lógica necessária para criar a base de dados. Desta classe tem que se implementar pelo menos três métodos: **construtor**, **onCreate(SQLiteDatabase db)** e **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)**. O método **onCreate(SQLiteDatabase db)** é chamado quando a base de dados é criada pela primeira vez e o método **onUpgrade(...)** é chamado quando a base de dados precisa de ser actualizada.

Na implementação deve-se usar este método para eliminar tabelas, adicionar tabelas ou atualizar para a nova versão o modelo. Quando se está a implementar a classe e o construtor da classe *SQLiteOpenHelper* é chamado, este consegue saber se a base de dados já existe ou se a versão é diferente da existente. Se a base de dados não existir, a função *onCreate* é invocada, se existir e a versão for diferente é invocada a função *onUpgrade*, e assim garante-se que a base de dados não está a ser construída sempre de novo.

Após ter esta classe implementada é necessário definir as operações de *insert*, *update*, *read* e *delete* da entidades criadas anteriormente.

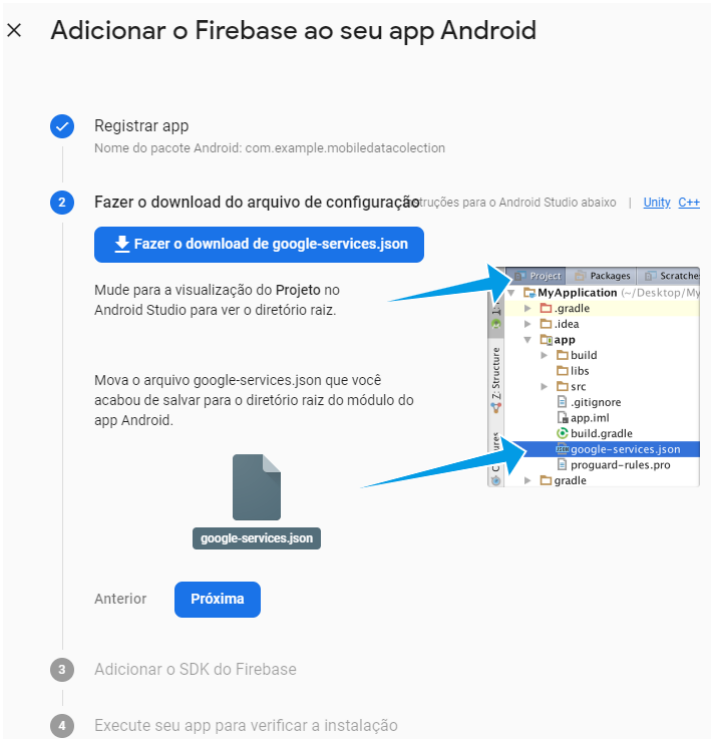


Figura 4.15: Fluxo de configuração da base de dados Firebase - Plataforma Móvel.

5

Estudo de caso e Resultados

Neste capítulo é evidenciado o desenvolvimento da plataforma para dispositivos móveis, de modo a ser possível a captura de informação através de inquéritos, sendo que o dispositivo pode estar tanto no estado *online* (ligação estável à *internet*), como no estado *offline* (sem ligação). São apresentados também todos os componentes e passos que são necessários para atingir o objectivo.

Foi considerado um caso prático onde estes inquéritos podem ser aplicados a uma empresa de qualquer área de negócio que tenha interesse em recolher informação de forma estruturada, por exemplo poderia ser aplicado sempre que fosse preciso realizar estudos em diversas áreas geográficas tendo ou não conexão com à *internet*, concebendo o caso prático: inquéritos à população escolar em zonas geográficas com conectividade intermitente, de modo a recolher as idades das pessoas em idade escolar por zona geográfica.

5.1 Geração do descritivo do formulário

Primeiramente, o utilizador deve criar o seu próprio formulário recorrendo à aplicação externa ODK Build, o utilizador necessita de ter uma conta criada. O inquérito realizado nesta aplicação deve de ir ao encontro das necessidades que o utilizador tem em recolher informação, neste caso recolha de informação sobre a população escolar, nomeadamente as idades dos estudantes.

Como se observa na Figura 5.1, através do *drag-and-drop*, foram colocados quatro elementos, sendo eles dois campos de texto, um numérico, e um de data.

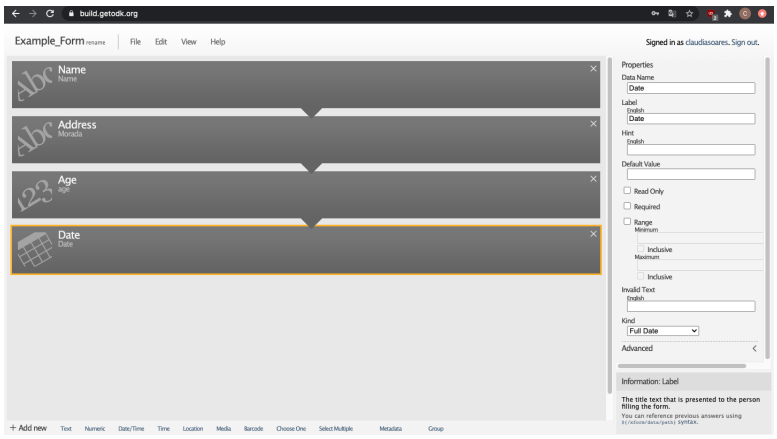


Figura 5.1: Configuração do Formulário através da ferramenta *ODK Build - online*.

Quando esta fase for dada como concluída basta carregar no menu *File* e de seguida *Export to XML*, exportando assim o formulário que foi desenvolvido para um ficheiro XML como se pode constatar na Figura 5.1.

5.2 Upload do formulário no servidor

Após ter este documento XML na máquina, Listagem 5.1, deve-se colocar este formulário no servidor, através da aplicação de *backoffice* desenvolvida. Deste modo o formulário fica disponível para utilização na aplicação móvel. Para colocar o formulário descrito em XML disponível, é necessário aceder à aplicação *web*, caso o utilizador não tenha acesso a esta aplicação deve-se registar, caso contrário basta apenas fazer *login*, Figura 5.2.

```
<h:html xmlns="http://www.w3.org/2002/xforms" xmlns:h="http://www.w3.org
    /1999/xhtml" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:jr="http
    ://openrosa.org/javarosa">
  <h:head>
    <h:title>Example_Form</h:title>
    <model>
      <instance>
        <data id="build_Example-Form_1600277332">
          <meta>
            <instanceID/>
          </meta>
          <Name/>
          <Morada/>
          <age/>
          <Date/>
        </data>
      </instance>
      <itext>
        <translation lang="English">
          <text id="/data/Name:label">
            <value>Name</value>
          </text>
          <text id="/data/Morada:label">
            <value>Address</value>
          </text>
          <text id="/data/age:label">
            <value>Age</value>
          </text>
          <text id="/data/Date:label">
            <value>Date</value>
          </text>
        </translation>
      </itext>
      <bind nodeset="/data/meta/instanceID" type="string" readonly="true()"
          calculate="concat('uuid:', uuid())"/>
      <bind nodeset="/data/Name" type="string"/>
      <bind nodeset="/data/Morada" type="string"/>
      <bind nodeset="/data/age" type="int"/>
      <bind nodeset="/data/Date" type="date"/>
    </model>
  </h:head>
  <h:body>
    <input ref="/data/Name">
      <label ref="jr:itext('/data/Name:label')"/>
    </input>
    <input ref="/data/Morada">
      <label ref="jr:itext('/data/Morada:label')"/>
    </input>
    <input ref="/data/age">
      <label ref="jr:itext('/data/age:label')"/>
    </input>
    <input ref="/data/Date">
      <label ref="jr:itext('/data/Date:label')"/>
    </input>
  </h:body>
</h:html>
```

Listagem 5.1: Documento XML resultante da configuração na ferramenta exposta na Figura 5.1.

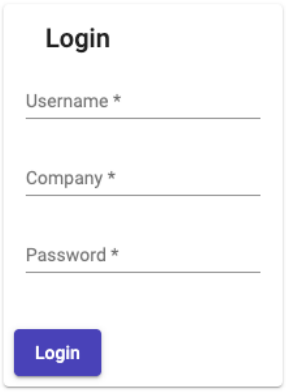


Figura 5.2: Ecrã onde se realiza ou *login*/registo da aplicação *web*.

Assim que é realizado o *login* na aplicação, o utilizador deve aceder ao menu lateral e carregar em *Upload Forms* para fazer carregamentos da descrição XML do inquérito. O utilizador que está a fazer o carregamento deve preencher a empresa, a categoria, a versão e a descrição, Figura 5.3. Esta informação é necessária para que haja organização de formulários dentro da aplicação.

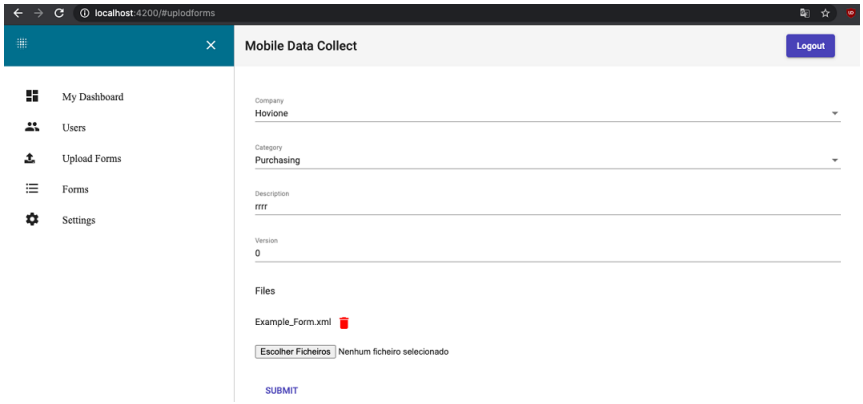


Figura 5.3: Realização do *upload* do documento XML com a descrição do formulário.

Após fazer o *upload* do documento, pode verificar-se se o *upload* correu como esperado. Esta verificação é possível acedendo ao menu que foi designado de *Forms*, onde é mostrado uma lista de todos os formulários presentes no servidor, como se pode observar na Figura 5.4.

Depois de terminar estes dois passos importantes ao nível da aplicação *web*, pode-se passar para a aplicação do dispositivo móvel.

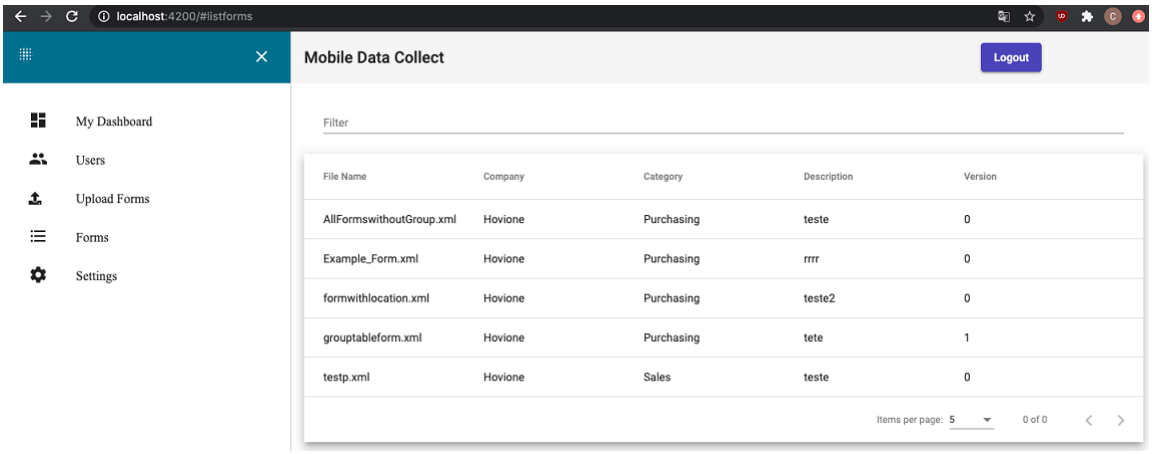


Figura 5.4: Lista de documentos XML carregues no servidor.

5.3 Avaliação qualitativa

Como primeiro passo deve-se configurar a aplicação em si para esta comunicar com o servidor, para isso basta ir aos *settings* e colocar o endereço do servidor, neste caso particular o endereço é `http://10.0.2.2:8000/`, como se pode observar na Figura 5.5.

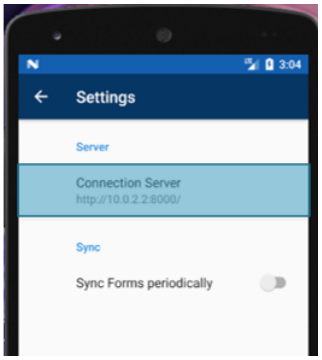


Figura 5.5: Ecrã de configuração, para comunicação com o servidor.

Esta configuração é essencial para que consiga-se obter os formulários que foram carregues na aplicação *web*, assim como, se o utilizador que está a efectuar o *login* está registado na aplicação *web*.

Logo após esta configuração inicial, já é possível fazer *login* na aplicação móvel, colocando o utilizador, a empresa e por fim a palavra-passe, clicando de seguida no botão de *login*, como é exibido na Figura 5.6.

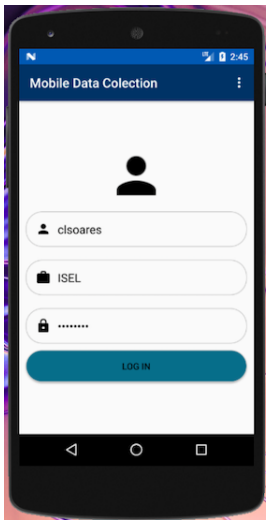


Figura 5.6: Ecrã da aplicação móvel para realizar o *login*.

Quando é realizado o *login* todos os utilizadores são sincronizados para o dispositivo móvel, de modo a que quando não haja ligação á *internet* se possa utilizar a aplicação móvel com todas as funcionalidades que a aplicação disponibiliza.

Ao ser efectuado o *login* com sucesso, o utilizador é encaminhado para um ecrã onde é possível fazer a sincronização com os formulários do servidor, ou escolher um já existente, como se visualiza na Figura 5.7.

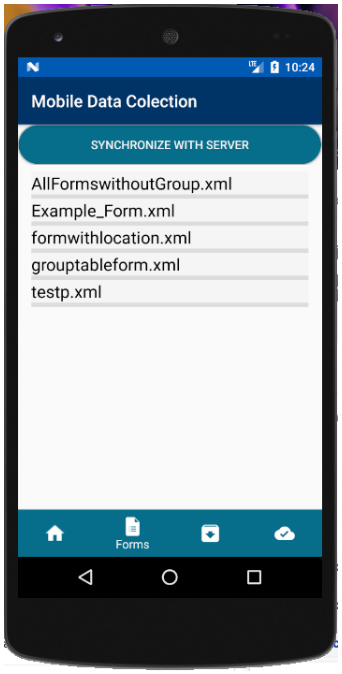
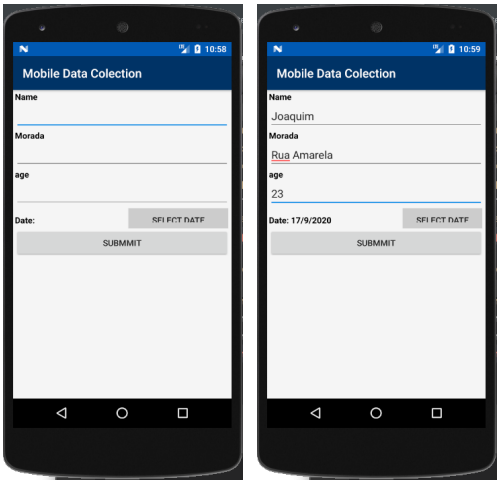


Figura 5.7: Lista de formulários sincronizados com os documentos XML do servidor.

Ao seleccionar o formulário desenvolvido nos passos anteriores, `Example_Form.xml`,

obtém-se o *design* mostrado na Figura 5.8. O formulário foi gerado de forma dinâmica, recorrendo aos *widgets* criados programaticamente.



(a) (b)

Figura 5.8: (a) *Layout* do formulário desenvolvido na ferramenta *ODK Build* e (b) Formulário preenchido.

Na Figura 5.8, é exibido o formulário preenchido, sendo que estes dados ficaram guardados na *Firebase* remota, visto que havia ligação estável à *internet*, estes dados são armazenados seguindo a estrutura que se pode ver na Figura 5.9.

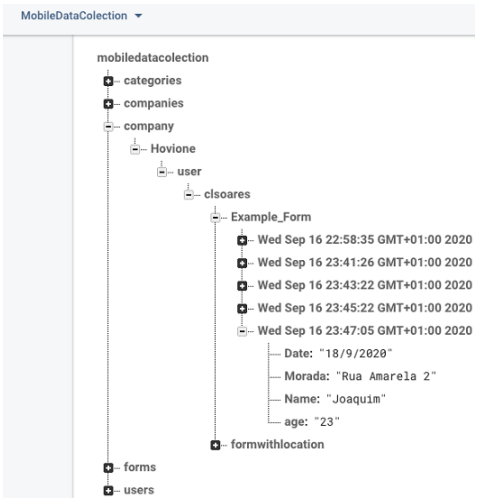


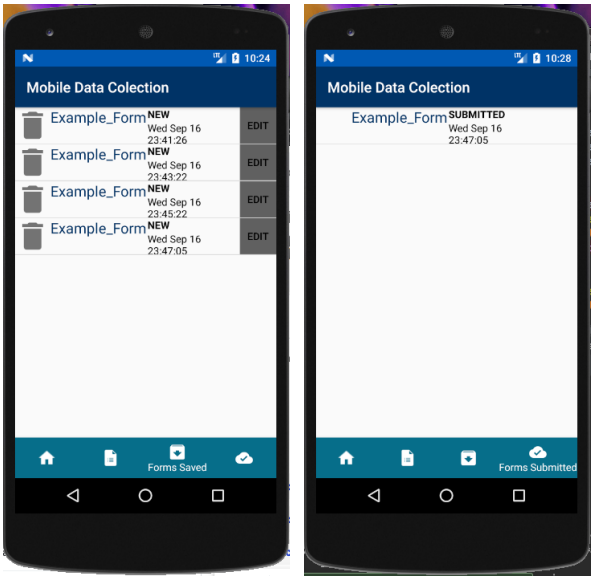
Figura 5.9: Estrutura do objecto guardado na base de dados com os dados recolhidos através do formulário.

Esta estrutura apresenta os dados guardados quer das categorias existentes na aplicação, como as empresas que aderiram à aplicação, a descrição de formulários submetidos, os utilizadores registados, e por fim os dados recolhidos dos formulários.

Os dados recolhidos dos formulários segue sempre a mesma estrutura, começando sempre pela propriedade *company* e de seguida o nome da empresa, depois a propriedade *user* e o utilizador que preencheu o formulário e por ultimo a propriedade do nome do formulário onde é guardado o objecto com os dados recolhidos sendo que a chave é a data de começo do preenchimento do formulário.

No caso da ligação ser interrompida a qualquer momento o SDK da *Firebase Realtime Database* aborta a transferência de dados para a base de dados remota permanecendo com os dados localmente. No momento que o *SDK* detecte o restabelecimento de conectividade os dados serão novamente sincronizados com a base de dados remota. Caso não exista de todo ligação à *internet*, os dados permanecem localmente na memória do dispositivo móvel, sendo possível continuar a operar a aplicação para recolha de dados e preenchimento dos formulários já persistidos na aplicação móvel.

Por fim, se este formulário for submetido assim que seja dado como terminado este irá fazer parte da lista que é mostrada na Figura 5.10 a) , caso contrário este formulário integrar a lista de formulários apresentado na Figura 5.10 b).



(a) (b)

Figura 5.10: (a) Lista de formulários guardados automaticamente, mas não submetidos. (b) Lista de formulários submetidos.

A diferença destas duas listas é que enquanto o formulário não for submetido dá para editar e no outro caso, se o formulário estiver no estado de submetido, não dá para editar. A lista de formulários submetidos apenas serve para o utilizador consultar os formulários que já preencheu.

5.4 Avaliação quantitativa

De modo a avaliar o desempenho da aplicação desenvolvida, optou-se por medir o tempo de renderização dos formulários no dispositivo móvel. Entende-se por tempo de renderização o tempo que a ferramenta desenvolvida leva a ler a descrição do formulário e a interpretar todos os *widgets* que o formulário contém, até à apresentação do mesmo.

Para realizar a avaliação quantitativa é necessário ter em conta as características do computador e do dispositivo móvel. O computador onde foram executados os testes tem como processador um Intel Core i5 de 1.3GHz de núcleo duplo e tem como memória RAM 4GB de 1600MHz DDR3. O dispositivo móvel que é executado através de um emulador tem 2GB de *RAM* sendo que o processador é um dual core.

Como se pode observar na Figura 5.11, o gráfico que é mostrado o eixo dos x representa o número de *widgets* por formulário e no eixo dos y o tempo que demorou a renderização do formulário, em milissegundos.

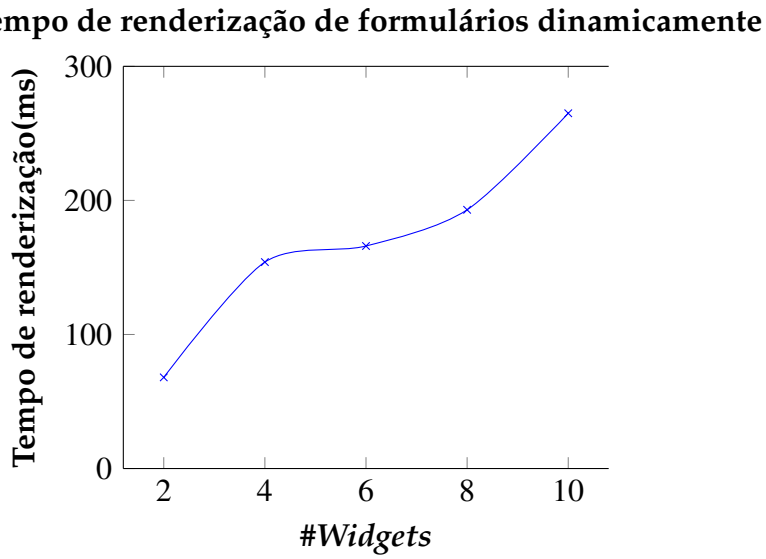


Figura 5.11: Tempo de renderização dinâmica de formulários, apenas com *widgets* de texto.

O tempo foi medido três vezes consecutivas, isto é, por cada formulário obteve-se o tempo de renderização por três vezes e depois aplicou-se uma média aritmética.

Os formulários utilizados para obter os valores do gráfico da Figura 5.11 apenas contêm *widgets* de texto.

Em análise ao gráfico da Figura 5.11, pode-se concluir que o tempo de renderização é influenciado pelo o número de *widgets* que o formulário tem, isto é, quantos mais *widgets* o formulário contiver mais tempo demora a renderizar, não chegando a ser diretamente proporcional.

6

Conclusões

Este projeto teve como objetivo desenvolver uma plataforma através da qual seja possível recolher informação na forma de formulários pré-definidos e eliminando a dependência da conexão estável com a *internet*, sem perder qualquer tipo de informação já recolhida.

A plataforma foi desenvolvida em duas partes: (a) aplicação *web*, que tem como funcionalidade a gestão da plataforma em si e carregamento de formulários para serem usados na aplicação móvel, (b) aplicação móvel, que ajuda a recolher dados através de inquéritos sem a preocupação de haver ou não uma ligação estável à *internet*. Esta separação torna possível o desenvolvimento independente de cada plataforma e ainda facilita a divisão de responsabilidades. Com a divisão de responsabilidades torna possível a implementação de restrições de acesso por funcionalidade.

De momento, na aplicação *web* de *backoffice*, está implementado o *login* e registo e gestão de utilizadores assim como o carregamento de formulários, de modo a que estes sejam guardados na base de dados *Firebase*. Também, houve a necessidade de categorizar os formulários carregados e associá-los a uma empresa. Com o objectivo de concretizar esta funcionalidade foi criada uma área de configuração onde se pode adicionar novas empresas e categorias. Existe também um campo que permite fazer o versionamento do formulário carregado.

A aplicação móvel, para o sistema operativo *Android*, também necessitou de validação do utilizador que está a utilizar a aplicação, pois apenas com esta informação

consegue-se saber quais os formulários que se tem que sincronizar. Esta sincronização está dependente da empresa do utilizador, ou seja, a empresa indicada no carregamento do formulário tem que corresponder à empresa associada ao utilizador. Após esta sincronização o utilizador consegue seleccionar o formulário que quer preencher, sendo que os dados após serem preenchidos são automaticamente guardados. Se houver ligação à *internet* os dados recolhidos são guardados automaticamente na base de dados *Firebase* alojada na *cloud*, caso contrário os dados são permanecidos localmente no dispositivo na base de dados *Firebase*.

Na construção dinâmica de formulários foi necessário desenhar cada um dos *widgets* de forma a que fosse possível visualizar na maioria dos dispositivos móveis. Devido à complexidade de desenhar e de posicionar os *widgets Android* no dispositivo móvel consoante a descrição do próprio formulário e mantendo um bom desempenho, o número de *widgets* desenvolvidos não foi o planeado. Considera-se um bom desempenho quando um *widget* é criado num menor tempo possível. É desejável que sejam implementados mais *widgets*, de modo a cobrir o maior número de formulários possível, mantendo sempre o mesmo pressuposto que será um ecrã que contem um *scroll* infinito para o preenchimento dos mesmos.

A comunicação entre a aplicação *web* e a aplicação *Android* tem de ser configurada, esta configuração é feita na aplicação móvel através de uma *Connection Server*. Esta comunicação entre as duas partes é realizada através de pedidos HTTP.

Um dos problemas que se teve que ter em conta foi a existência da ligação à *internet*. O primeiro contacto do utilizador com a aplicação móvel tem como requisito a conexão à *internet*, devido ao facto da base dados *SQLite* ainda não conter a informação que permita o *login* do utilizador, assim como a falta de existência de formulários para o utilizador poder preencher. A partir deste momento, a aplicação não necessita mais desta ligação, pois se o dispositivo estiver no modo *offline* os dados são persistidos na base de dados local, *Firebase*. Sendo que a *Firebase* consegue ter a sensibilidade para saber se os dados podem ou não ser sincronizados com a base de dados remota, o programador não necessita de ter em atenção o estado do dispositivo móvel.

Por fim existe um problema no sistema desenvolvido, caso dois utilizadores partilhem o mesmo *login* e iniciem o preenchimento do formulário ao mesmo tempo, ao milésimo de segundo, existe uma sobreposição de dados quando estes são guardados, teria que se arranjar forma de resolver este problema.

6.1 Trabalho Futuro

Nesta secção, são mencionadas algumas sugestões a trabalho que se pode desenvolver futuramente no âmbito deste projeto.

Este projeto pode conter melhorias continuas, visto que a tecnologia vai avançando e vai sendo possível implementar novas funcionalidades e novos *widgets* em toda a aplicação. Alguns dos *widgets* que são necessários à construção dinâmica dos formulários não estão nem desenhados nem mapeados para tal. Então seria necessário haver um desenvolvimento de *widgets* de modo a suportar qualquer tipo de formulário com qualquer tipo de *widgets*, desde tabelas a leitura de *barcodes*, por exemplo.

Outra melhoria que se pode ponderar será replicar as funcionalidades que se desenvolveram para a aplicação *web* na aplicação móvel, dando assim flexibilidade para em qualquer lugar que contivesse conexão à *internet* fazer a gestão da aplicação e de utilizadores, assim como realizar *uploads* de formulários para o servidor.

Uma das limitações desta aplicação é que só interpreta formulários descritos na linguagem de XML, por exemplo, se fizermos um formulário na linguagem JSON este já não é interpretado de forma dinâmica, então uma outra melhoria seria alterar a aplicação móvel de modo a que diversas linguagens de descrição de formulários fossem facilmente interpretadas.

Finalmente, fica também em aberto o desenvolvimento de uma ferramenta de análise de dados, para que as actividades empresariais consigam criar métricas de modo a tirarem conclusões dos dados obtidos através do preenchimento de formulários.

Referências

[1] Carl Hartung, Adam Lerer, Yaw Anokwa, Clint Tseng, Waylon Brunette & Gaetano Borriello, "Open data kit: Tools to build information services for developing regions", em *Proceedings of the 4th ACM/IEEE International Conference on Information and Communication Technologies and Development*, sér. ICTD '10, London, United Kingdom: Association for Computing Machinery, 2010, ISBN: 9781450307871. DOI: 10.1145/2369220.2369236. URL: https://doi.org/10.1145/2369220.2369236.

[2] Stratos Idreos & Mark Callaghan, "Key-value storage engines", em *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, sér. SIGMOD '20, Portland, OR, USA: Association for Computing Machinery, 2020, 2667–2672, ISBN: 9781450367356. DOI: 10.1145/3318464.3383133. URL: https://doi.org/10.1145/3318464.3383133.

[3] Satya Komatineni, Sayed Hashimi & Dave MacLean, *Pro Android 3*, 1st. USA: Apress, 2011, ISBN: 1430232226.

[4] *O que é um banco de dados de chave-valor?* URL: https://aws.amazon.com/pt/nosql/key-value/.

[5] *Couchbase lite*. URL: https://docs.couchbase.com/home/index.html.

[6] *Firebase db*. URL: https://firebase.google.com/docs/database.

[7] Jennifer Kyrnin, *HTML5 Mobile Application Development*. Sams, 2012.

[8] *Html evolution*. URL: https://www.w3schools.in/html-tutorial/web-forms/.

[9] *Html history*. URL: https://www.hostinger.com/tutorials/what-is-html#The-History-of-HTML.

[10] *Infopath*. URL: https://support.office.com/pt-pt/article/introduÃğÃčo-ao-microsoft-infopath-2010-70c21df0-6f93-4140-99e6-676d489b2818.

[11] *Javarosa*. URL: https://github.com/opendatakit/javarosa.

[12] *Kobotoolbox*. URL: https://www.kobotoolbox.org/#home.

[13] João Ricardo Lourenço, Bruno Cabral, Paulo Carreiro, Marco Vieira & Jorge Bernardino, "Choosing the right nosql database for the job: A quality attribute evaluation", *Journal of Big Data*, vol. 2, n.º 1, pág. 18, 2015, ISSN: 2196-1115. DOI: 10.1186/s40537-015-0025-0. URL: https://doi.org/10.1186/s40537-015-0025-0.

[14] Oracle. URL: https://mariadb.org/.

[15] Microsoft, *Databases*. URL: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15.

[16] Oracle. URL: https://dev.mysql.com/.

[17] *Open data kit: Tools to build information services for developing regions*. URL: https://dl.acm.org/doi/10.1145/2369220.2369236.

[18] *Odk briefcase*. URL: https://docs.getodk.org/briefcase-intro/.

[19] *Odk build*. URL: https://docs.getodk.org/build-intro/.

[20] *Odk central*. URL: https://docs.getodk.org/central-intro/.

[21] *Odk collect*. URL: https://docs.getodk.org/collect-intro/.

[22] *Odk javarosa*. URL: https://docs.getodk.org/javarosa/.

[23] *Odk validate*. URL: https://docs.getodk.org/validate/.

[24] *Odk-x*. URL: https://docs.odk-x.org/#.

[25] *Oracle database documentation*. URL: https://docs.oracle.com/en/database/oracle/oracle-database/index.html.

[26] *Orm db*. URL: http://ormlite.com/.

[27] *Enabling offline capabilities on android*. URL: https://firebase.google.com/docs/database/android/offline-capabilities#section-disk-persistence.

[28] University of California at Berkeley. URL: https://www.postgresql.org/.

[29] *O que é react native?* URL: https://www.organicadigital.com/blog/o-que-e-react-native/.

[30] *Real db*. URL: https://realm.io/.

[31] Abhishek Parihar, *Five of the most popular databases for mobile apps*, 2017. URL: https://blog.trigent.com/five-of-the-most-popular-databases-for-mobile-apps.

[32] Sergio Palazzo, Antonio Puliafito & Marco Scarpa, "Design and evaluation of a replicated database for mobile systems", *Wireless Networks*, vol. 6, páginas 131–144, mai. de 2000. DOI: 10.1023/A:1019125227988.

[33] *Distribuição de dados*. URL: https://1920moodle.isel.pt/pluginfile.php/923008/mod_resource/content/1/ASI_M2_DistribuiÃğÃčoDeDados.pdf.

[34] *Sqllite*. URL: https://www.sqlite.org/index.html.

[35] Microsoft, *Azure sql database documentation*. URL: https://docs.microsoft.com/en-us/azure/sql-database/.

[36] *Surveycto*. URL: https://www.surveycto.com/.

[37] *Surveycto vs odk*. URL: https://www.surveycto.com/product/surveycto-vs-odk/.

[38] Ryan D. Cronk Michael B. Fisher Benjamin H. Mann, Katherine F. Shields, Tori L. Klug & Rohit Ramaswamy, "Evaluating mobile survey tools (msts) for field-level monitoring and data collection: Development of a novel evaluation framework, and application to msts for rural water and sanitation monitoring", *International journal of environmental research and public health*, vol. 13, 2016.

[39] Bruno Rafael, *Sql, nosql, newsql: Qual banco de dados usar?*, 2019. URL: https://blog.geekhunter.com.br/sql-nosql-newsql-qual-banco-de-dados-usar/.

[40] Ed Tittel, *XML For Dummies, 4th Edition*. John Wiley & Sons, 2005. URL: https://cdn.preterhuman.net/texts/manuals/XMLforDummies4thEdition.pdf.

[41] Alberto Silva e Carlos Videira, *UML, Metodologias e Ferramentas CASE, Centro Atlântico, 1ª edição*. Edições Centro Atlântico Portuga, Abril, 2001.

[42] John M. Boyer, *Xforms 2.0 - w3c working draft 7 august 2012*, 2012. URL: https://www.w3.org/TR/xforms20/.

[43] *Xlsform*. URL: https://docs.getodk.org/xlsform/.