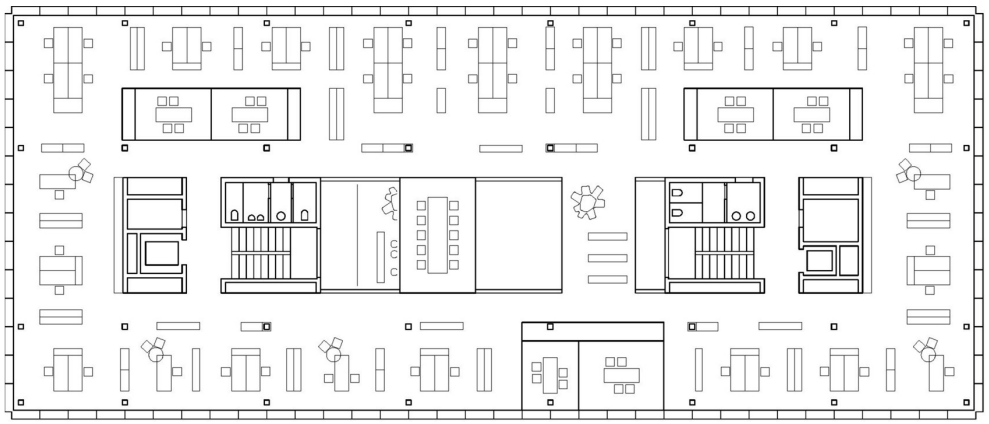
ISEL – Instituto Superior de Engenharia de Lisboa
ADEETC – Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

LEIM
Licenciatura em Engenharia Informática e Multimédia
Unidade Curricular de Projeto

SpaceManager – Módulo App



Ana Rita Venâncio Alves (42360)

Orientador

Professor Doutor Carlos Gonçalves

Julho, 2020

Resumo

Escrever aqui uma perspectiva geral do seu trabalho ...

Motivação, ideias mais relevantes, principais contributos, avaliações e breve conclusão.

Frases breves. Parágrafos concisos. Abordagem "top-down".

Abstract

Write here an overview of your work ...

Motivation, most relevant ideas, main contributions, evaluations and brief conclusions.

Short sentences. Succinct paragraphs. Top-down approach.

Agradecimentos

Escrever aqui eventuais agradecimentos ...

Índice

Lista de Figuras

Lista de Tabelas

Lista de Exemplos

Capítulo 1

Introdução

No contexto de empresas que tem funcionários que passam pouco tempo nas instalações, e onde os locais de trabalho estão organizados em *open space*, não se justifica que cada funcionário tenha um local de trabalho fixo. Neste cenário surgiu a ideia de desenvolver o projeto *SpaceManager* que permite: i) Gerir a reserva dos postos de trabalho; ii) Verificar através de um conjunto de sensores quais os postos de trabalho efetivamente ocupados; iii) Definir a organização dos espaços *open space*.

Na figura 1.1 estão representadas as três partes que constituem o projeto *SpaceManager*. O trabalho que irá ser abordado neste relatório é o desenvolvimento da componente de *software* que permite gerir as reservas dos vários postos de trabalho.

Figura 1.1: Relação entre as três componentes do projeto *SpaceManager*

Como podemos ver na figura 1.1 este projeto é constituido por três partes. A primeira é a componente de *software*, representado pelo telemóvel, que permite gerir as reservas, como foi referido anteriormente. A segunda [1], re-

presentado pelo sensor, é a componente *hardware* cujo objetivo é desenvolver um sistema que consiga detetar se o posto de trabalho está efetivamente ocupado ou não. A terceira parte [2], representado pelo edifício, tem o objetivo de implementar uma componente de *software*, a ser executada num *browser*, que permite definir um edifício a quantidade de pisos que existem num edifício, o formato de cada piso, os postos de trabalho que existem em cada piso e os sensores que existem em cada posto de trabalho.

1.1 Objetivo

O trabalho apresentado neste relatório corresponde à componente do projeto *SpaceManager* que visa o desenvolvimento da componente de gestão de reservas de postos de trabalho. O nome atribuído a este trabalho foi *SpaceManager – App* e disponibiliza uma aplicação para o sistema *Android*, na qual o utilizador tem a possibilidade de reservar um lugar. A aplicação permite reservar lugares de vários modos: i) Reservar um lugar aleatório; ii) Reservar um lugar através do mapa da sala; iii) Reservar um lugar aleatório com uma determinada característica, como por exemplo um lugar que seja perto de uma janela; iv) Reservar vários lugares para uma reunião. A aplicação permite ainda a possibilidade de prolongar uma reserva que esteja a decorrer. Como funcionalidades extra da aplicação é possível efetuar pesquisas que permitem saber se um determinado colaborador da empresa está com uma reserva ativa.

1.2 Organização do Documento

Este documento além deste capítulo contém os seguintes capítulos. O capítulo 2 onde são apresentados trabalhos relacionados e as tecnologias utilizadas para o desenvolvimento do projeto. Os casos de utilização, os requisitos funcionais e não funcionais e a abordagem utilizada são apresentados no capítulo 3. No capítulo 4 é apresentada a implementação do projeto, incluindo o modelo entidade-associação e o modelo relacional. O capítulo 5 apresenta os testes realizados que comprovam o correto funcionamento da aplicação. Este trabalho termina com o capítulo 6 onde são apresentadas as conclusões e o trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo apresentam-se soluções já existentes semelhantes à temática tratada neste trabalho, secção 2.1, e as tecnologias utilizadas no desenvolvimento deste trabalho, secção 2.2.

2.1 Aplicações

Existem várias soluções com funcionalidades semelhantes ao projeto *SpaceManager*. A Steelcase, uma empresa de venda de mobiliário tem uma solução proprietária para integrar com o mobiliário que é vendido por eles. O RoomWizard [3] é um sistema de reserva de salas de reunião. Foi projetado intencionalmente para mostrar informação importante de reuniões à distância, ajudar na localização e agendamento de espaços de reuniões da sua mesa ou do dispositivo. O software esta integrado no mobiliario, nao permite integrar software de outros fabricantes.

A Sony também apresenta um solução para a gestão de postos de trabalho inteligentes. TEOS [4] é um conjunto completo de soluções de gestão de postos de trabalho.

A Cisco

tabela com as ideias o trabalho apresentado neste relatorio...

2.2 Tecnologias

Para a implementação deste projeto foi utilizado como ambiente de desenvolvimento o `Android Studio` [5]. Para o armazenamento dos dados e au-

tenticação foi utilizada a *framework* `Firebase` [6], cuja a arquitetura simplificada se apresenta na figura2.1.

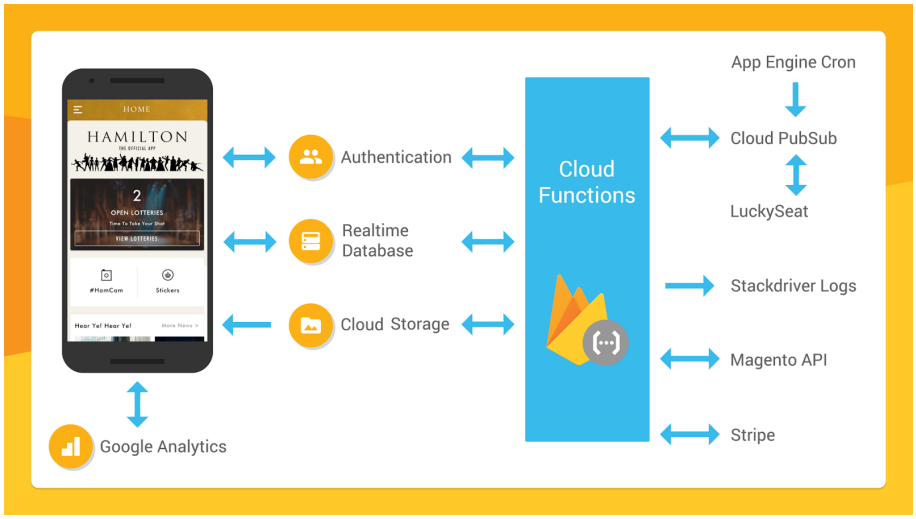


Figura 2.1: Arquitetura do `Firebase`

Esta *framework* foi utilizada pelo facto de ser a componente que auxilia o `Android Studio` na autenticação de utilizadores. Dado que o `Android Studio` disponibiliza uma base de dados, neste projeto, por uma questão de simplificação optou-se por utilizar essa base de dados. No entanto, a base de dados suportada pela *framework* `Firebase` é do tipo `NoSQL`, pelo que apenas permite um modelo de dados não relacional. Uma vez que as bases de dados `NoSQL` não garantem a integridade das relações dos dados, foi necessário implementar a nível da aplicação a verificação e manutenção destas integridades. Na secção 3.4 apresenta-se com mais detalhe como esta verificação foi implementada.

Neste projeto apenas foram utilizados os mecanismos de autenticação e armazenamento de dados. Os mecanismos *Cloud Storage* e *Google Analytics*, apesar de não terem sido utilizados, seriam uma hipótese a considerar se fosse necessário, respetivamente, guardar um conjunto de recursos, tais como ficheiros `PDF` associados a uma reunião ou efetuar uma análise estatística sobre quais os postos de trabalho mais reservados ou as características que os colaboradores procuram mais quando reservam um posto de trabalho.

Capítulo 3

Modelo Proposto

Neste capítulo apresenta-se o modelo proposto para o projeto. Na secção 3.2 são apresentados os casos de utilização. Na secção 3.1 apresentam-se os requisitos funcionais e não funcionais. O capítulo termina com a secção 3.3 onde é apresentada a arquitetura implementada.

3.1 Requisitos

Nesta secção são apresentados os requisitos funcionais e não funcionais. Um requisito é uma descrição das necessidades ou propósitos do produto. Um requisito funcional é a descrição de uma funcionalidade do sistema, enquanto que um requisito não funcional é a descrição de como é realizada uma funcionalidade do sistema. Na subsecção 3.1.1 são apresentados os requisitos funcionais e na subsecção 3.1.2 são apresentados os requisitos não funcionais.

3.1.1 Requisitos Funcionais

Os requisitos funcionais dividem-sem em três categorias: i) Evidentes – o utilizador tem que ter conhecimento da sua realização; ii) Invisíveis – não é possível ao utilizador visualizar-los; iii) Adornos – não afeta significativamente o custo ou outras funções. No contexto deste trabalho apenas foram considerados os requisitos funcionais evidentes e invisíveis. Os requisitos funcionais são apresentados na tabela 3.1.

Da análise da tabela 3.1 verifica-se que os requisitos funcionais que são realizados dentro do sistema são invisíveis, como é o caso do **Registar utilizador** e do **Iniciar sessão**. Os requisitos **Reservar lugar**, **Prolongar**

Tabela 3.1: Requisitos funcionais

Requisito	***Tipo***
1. Registar utilizador	Invisível
2. Iniciar sessão	Invisível
3. Reservar lugar	Evidente
4. Prolongar reserva	Evidente
5. Pesquisar pessoa	Evidente
6. Ver reservas	Evidente
7. Apagar reserva	Evidente

reserva, **Pesquisar pessoa**, **Ver reservas** e **Apagar reserva** como são funcionalidades que o utilizador vê, são classificados como requisitos funcionais evidentes.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais são apresentados na tabela 3.2. Como foi referido anteriormente estes representam a forma como vão ser implementadas as funcionalidades descritas nos requisitos funcionais.

Tabela 3.2: Requisitos não funcionais

Requisito
Mecanismo de autenticação Google
Base de dados
Sensores infravermelhos

Como se apresenta na tabela 3.2 existem três requisitos não funcionais. O primeiro, **Mecanismo de autenticação Google**, é a capacidade que o sistema tem de permitir a autenticação pelo Google através do Gmail. O segundo, **Base de dados**, é utilizar uma base de dados onde é guardada a informação sobre os utilizadores e os lugares. Por último, **Sensores infravermelhos**, são utilizados sensores infravermelhos para determinar se um lugar está ou não ocupado atualmente.

3.2 Casos de Utilização

Na figura 3.1 estão representados os casos de utilização do sistema.

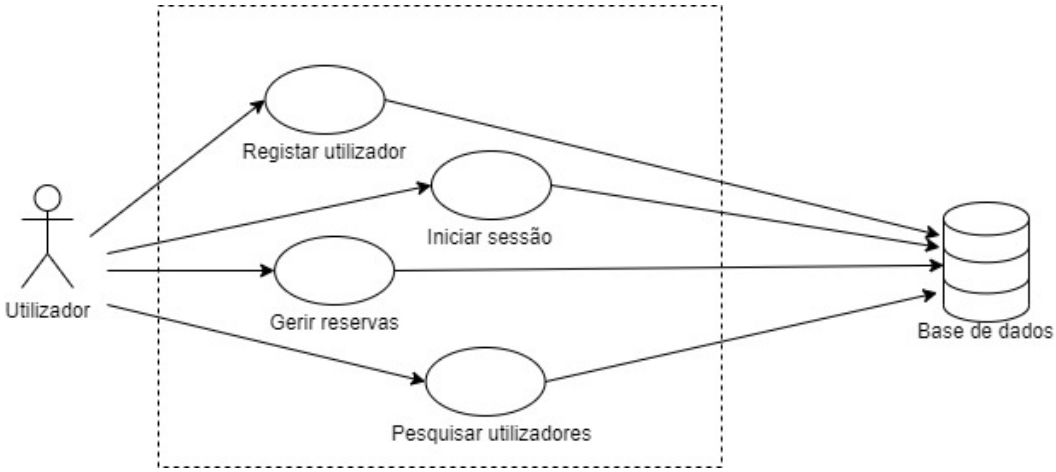


Figura 3.1: Casos de utilização

Como podemos aferir existem 4 casos de utilização. O utilizador tem a possibilidade de se **registar no sistema**, no qual este fica guardado num base de dados. Posteriormente pode **iniciar a sessão**, em que o sistema acede à base de dados para verificar as credenciais do utilizador e ver se as mesmas são válidas. Na **gestão das reservas** existem várias possibilidades, o utilizador pode fazer uma reserva, que fica guardada na base de dados, pode eliminá-la, e esta é removida da base de dados e pode prolongá-la e a reserva é alterada. Em relação à **pesquisa de utilizadores**, é possível consultar as reservas atuais de modo a perceber se existe uma reserva ativa para um colaborador que é especificado como argumento da pesquisa.

3.3 Modelo de Dados

Na figura 3.2 está representado o modelo entidade-associação.

Como podemos ver no modelo na figura 3.2 a entidade `Sala` que é caracterizada por ter um identificador, uma descrição e um piso. Um `Lugar` está sempre associado a uma `Sala` e também tem um identificado. Um `Lugar` também está associado a um `Sensor` que é caraterizado por um identificador e por um `URL`. A `Caracteristica` está associada ao `Lugar` e tem como atri-

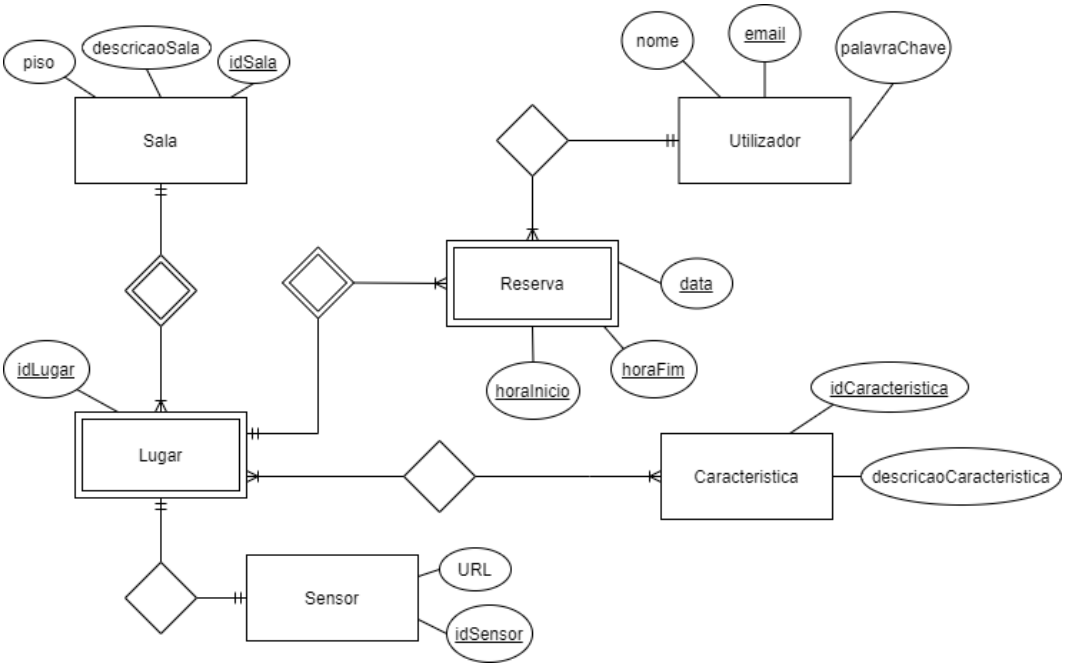


Figura 3.2: Modelo entidade-associação

butos o identificador e a descrição. A entidade `Utilizador` define o utilizador do sistema, é caracterizado pelo *e-mail*, nome e palavra-chave. Por fim, a entidade `Reserva`, depende de um `Lugar` e está associada a um `Utilizador`.

Do modelo entidade-associação apresentado na figura 3.2 obtém-se o modelo relacional apresentado na listagem 3.1.

Listagem 3.1: Modelo Relacional

```
1  SALA( idSala, piso, descricao )
2  CK = { idSala }
3
4  LUGAR( idLugar, idSala )
5  CK = { idLugar }
6  FK = { idSala } em SALA
7
8  CARACTERISTICA( idCaracteristica, descricao )
9  CK = { idCaracteristica }
10
11 LUGAR_CARACTERISTICA( idLugar, idSala,
     ↪idCaracteristica )
12 CK = { idLugar, idSala, idCaracteristica }
```

```
13 FK = { idLugar, idSala } em LUGAR
14
15 UTILIZADOR( email, nome, palavraChave )
16 CK = { email }
17
18 RESERVA( data, horaInicio, horaFim, idLugar, idSala,
      ↪ email )
19 CK = { data, horaInicio, horaFim, idLugar, idSala }
20 FK1 = { idLugar, idSala } em LUGAR
21 FK2 = { email } em UTILIZADOR not null
22
23 LUGAR_SENSOR( idLugar, idSala, idSensor )
24 CK = { idLugar, idSala, idSensor }
25 FK1 = { idLugar, idSala } em LUGAR
26 FK2 = { idSensor} em SENSOR
27
28 SENSOR( idSensor, URL )
29 CK = { idSensor }
```

3.4 Implementação do Modelo Relacional numa Base de Dados NoSQL

Do modelo entidade-associação apresentado na figura 3.2 foi derivado o modelo relacional da listagem 3.1. Como foi referido na secção 2.2 o tipo da base de dados utilizada é NoSQL, pelo que o modelo relacional apresentado no listagem 3.1 teve de ser implementada garantindo a integridade da relações. Foi criada uma `Database Reference` por cada tabela do modelo relacional. Também foi criada uma `Database Reference` para simular o estado de cada sensor, ou seja, determinar se o posto de trabalho está ou não ocupado.

Houve a necessidade de representar cada identificador através de um código único. Esse código é do tipo MD5 e cria um identificador de 128 bits e pode ser derivado de um ou mais valores. No capítulo 4 esta explicação será complementada.

Capítulo 4

Implementação

Neste capítulo apresenta-se a implementação da aplicação que foi desenvolvida para *Android*. Na secção 4.1 são apresentados os métodos de autenticação disponibilizados pela aplicação. Os diversos tipos de reserva são apresentados na secção 4.2. A funcionalidade de prolongar uma reserva é apresentada na secção 4.3. Na secção 4.4 é apresentada a funcionalidade que permite visualizar as reservas futuras.

4.1 Autenticação

O utilizador tem a possibilidade de se autenticar através da aplicação. Primeiro tem que efetuar o registo, na qual insere o nome, endereço de *e-mail* e palavra-chave, de seguida inicia a sessão utilizando o endereço de *e-mail* e palavra-chave. Em alternativa é possível efetuar a autenticação através do endereço de *e-mail* do Google, o Gmail.

Quando o utilizador faz o registo na aplicação, os seus dados são adicionados à base de dados do `Firebase` na referência `utilizadores`, e onde cada utilizador fica associado a um código `MD5`. Este é utilizado para ser mais fácil organizar a informação dos utilizadores na base de dados. O código Java referente à criação do código MD5 para o utilizador encontra-se na listagem 4.1.

Listagem 4.1: Criação do código MD5 para o utilizador

```
1 public void onDataChange(@NonNull DataSnapshot ds) {
2  String md5 = "";
3  try {
```

```
4     MessageDigest md=MessageDigest.getInstance("MD5");
5     md.update(sEmail.getBytes(),0,sEmail.length());
6     md5 = new BigInteger(1, md.digest()).toString(16);
7    }catch(NoSuchAlgorithmException e){
8     System.err.println("Erro ao gerar o ócdigo MD5");
9    }
10
11   DatabaseReference user = users.child(md5);
12   DatabaseReference nome = user.child("nome");
13   nome.setValue(sNome);
14   DatabaseReference email = user.child("email");
15   email.setValue(sEmail);
16   DatabaseReference pass = user.child("pass");
17   pass.setValue(sPass);
18  }
```

Neste caso o código md5 é derivado do e-mail. este codigo é utilizado como identificador único na referencia utilizadores. Este mecanismo foi utilizado em todas as situacao onde foi necesario suportar o modelo sql numa base de dados nosql.

Quando o utilizador inicia a sessão, o sistema verifica se o endereço de *e-mail* introduzido corresponde à palavra-chave fornecida.

Para o inicio da sessão ser realizada através do endereço de *e-mail* do Google o sistema ...

4.2 Reservas

Na subsecção 4.2.1 é apresentada a implementação da reserva de um lugar aleatório. A implementação da reserva de um lugar escolhido do mapa da sala é apresentada na subsecção 4.2.2. A subsecção 4.2.3 apresenta a implementação da reserva de grupo. A implementação da reserva pela característica do lugar é apresentada na subsecção 4.2.4.

A figura ... apresenta o ecra inicial comum a todas as reservas.

Antes do utilizador efetuar qualquer reserva necessita de escolher o horário no formato data/hora. Na aplicação a seleção da data é efetuada com recurso e um `Widget` do tipo calendário (`CalendarView`). A escolha da hora é realizada com recurso a um `Widget` do tipo `Spinner`. Por omissão a duração inicial de uma reserva poderá ser uma ou duas horas.

sempre qe se carrega num botao...

4.2.1 Reserva de um Lugar

ga

4.2.2 Reserva Através do Mapa da Sala

Este tipo de reserva é realizado sobre um mapa, em que o utilizador seleciona o lugar que pretende clicando sobre ele.

4.2.3 Reserva por Grupo

aga

4.2.4 Reserva por Característica

ga

4.3 Prolongar Reserva

ga

4.4 As Minhas Reservas

O utilizador tem a possibilidade de poder ver as suas reservas futuras. O sistema vai à base de dados e percorre todas as reservas e vê quais é que pertencem ao utilizador coma sessão iniciada.

4.5 Integração com o Projeto SpaceManager – Módulo Sensores

ga

Capítulo 5

Validação e Testes

explicar topicos/funcionalidades

Capítulo 6

Conclusões e Trabalho Futuro

hgxhxhxg Notificação quando reserva começa e quando vai acabar.

Bibliografia

[1] P. Marques, *SpaceManager – Sensores*. Intituto Superior de Engenharia de Lisboa, Projeto Final de Curso ed., 2020. Licenciatura em Engenharia Informática e Multimédia.

[2] B. Silva, *SpaceManager – Web Components*. Intituto Superior de Engenharia de Lisboa, Projeto Final de Curso ed., 2020. Licenciatura em Engenharia Informática e Multimédia.

[3] "Steelcase – RoomWizard," July 2020.

[4] "Sony – TEOS," July 2020.

[5] "Android Studio," June 2020.

[6] "Firebase," June 2020.

[7] "Arquitetura Firebase," June 2020.

Apêndice A

Questionário da Usabilidade do Sistema

Neste apêndice são apresentadas as perguntas colocadas aos utilizadores que testaram a aplicação.

1 Usaria o sistema com frequência.

2 O sistema é demasiado complexo.

3 O sistema é fácil de usar

4 Precisei de ajuda de alguém com conhecimentos técnicos para utilizar o sistema.

5 As várias funções do sistema estão muito bem integradas.

6 O sistema apresenta muita inconsistência.

7 O sistema é de rápida aprendizagem.

8 O sistema é confuso.

9 Senti-me confiante a utilizar o sistema.

10 Foi necessário aprender coisas novas para conseguir utilizar o sistema.

resultados das perguntas

Apêndice B

Outro Detalhe Adicional

Listagem B.1: Utilizacao

```
1  public class Opcao extends AppCompatActivity
     ↪implements View.OnClickListener{
2
3    private ConstraintLayout constraintLayout;
4
5    @Override
6    protected void onCreate(Bundle savedInstanceState)
       ↪ {
7      setRequestedOrientation(ActivityInfo.
         ↪SCREEN_ORIENTATION_PORTRAIT);
8      super.onCreate(savedInstanceState);
9      setContentView(R.layout.opcao);
10
11     Button lugar = findViewById(R.id.BObter);
12     lugar.setOnClickListener(this);
13     Button voltar = findViewById(R.id.BVoltar);
14     voltar.setOnClickListener(this);
15   }
16
17   @Override
18   public void onClick(View v) {
19     switch (v.getId()) {
20       case R.id.BObter:
21         Intent intent_lugar = new Intent(Opcao.this,
             ↪ LugarObtido.class);
22         Opcao.this.startActivity(intent_lugar);
```

```
23          break;
24        case R.id.BVoltar:
25          Intent intent_voltar = new Intent(Opcao.this
              ↪, Reserva.class);
26          Opcao.this.startActivity(intent_voltar);
27          break;
28
29      }
30    }
31  }
```