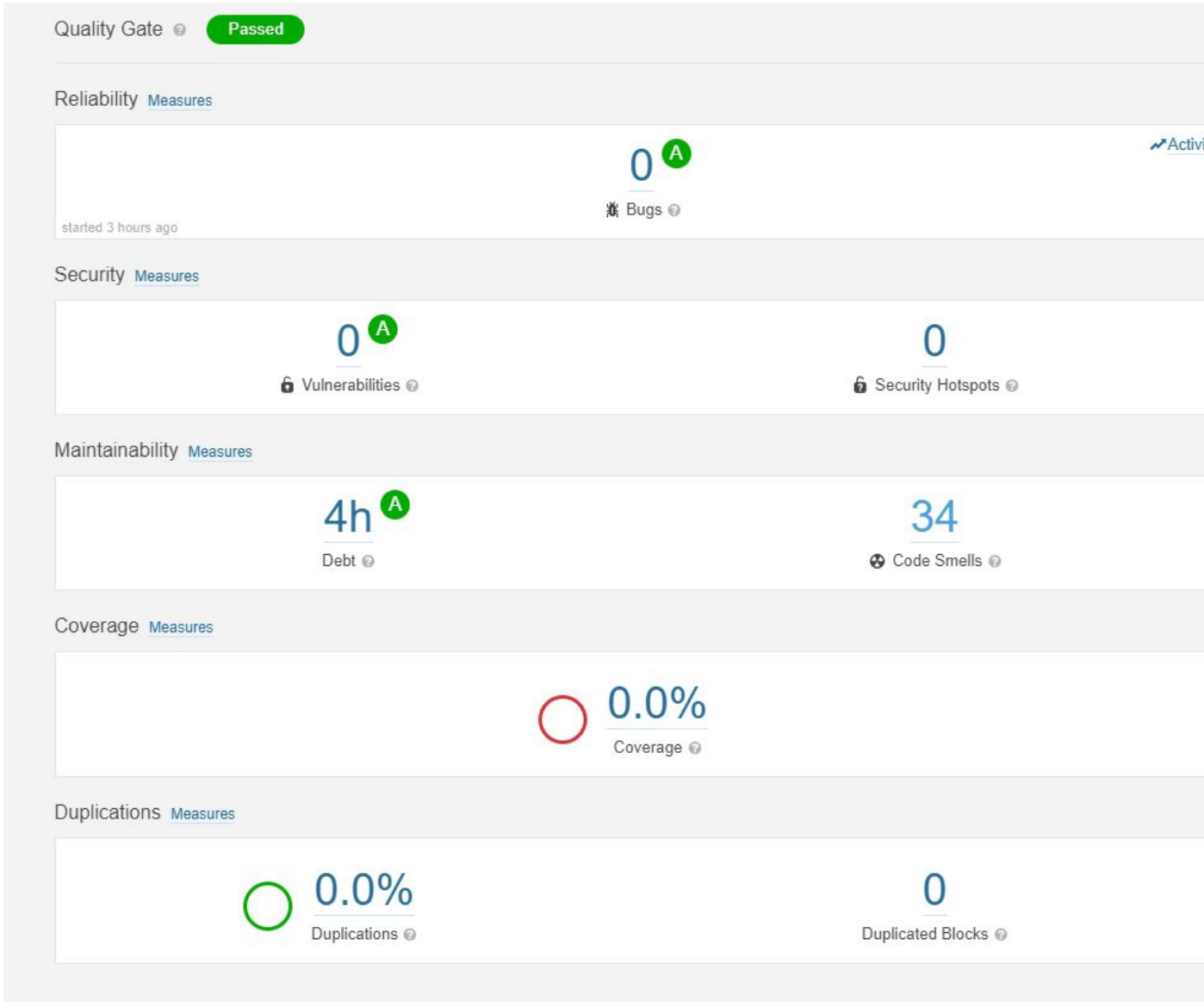


Código analisado com a API Sonarqube em 29/10/2019

Análise geral do código



Code Smells presentes no código

Erros sublinhados de vermelho

A parte de importações de um arquivo deve ser manipulada pelo IDE, não manualmente.

Importações não utilizadas e inúteis não devem ocorrer, se for o caso. Deixá-los reduz a legibilidade do código, pois a presença deles pode ser confusa.

```
1  ... package com.ufrpe.bsi.soresenha.eventos.gui;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.os.Build;
6
7  import android.support.annotation.NonNull;
8  import android.support.v7.widget.RecyclerView;
9  import android.view.ContextMenu;
10
11 import android.view.Gravity;
12
13 import android.view.LayoutInflater;
14 import android.view.MenuInflater;
15
16 import android.view.MenuItem;
17 import android.view.View;
18 import android.view.ViewGroup;
19 import android.widget.ImageButton;
```

```
16 import android.widget.PopupMenu;
17 import android.widget.TextView;
18 import android.widget.Toast;
```

O requisito para um default é a programação defensiva. O método deve tomar as medidas apropriadas ou conter um comentário adequado sobre o motivo pelo qual nenhuma ação foi tomada.

```
private void popupActions(PopupMenu popup, final int position) {
    popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem menuItem) {
            switch (menuItem.getItemId()) {
```

As instruções switch são úteis quando existem muitos casos diferentes, dependendo do valor da mesma expressão.

Porém, apenas para um ou dois casos, o código ficará mais legível com as instruções if.

```
        case R.id.editEvent:
            moveToEdit(position);
            break;
        case R.id.deleteEvent:
            deleteEvent(position);
            break;
    }
    return false;
}
});
}
```

Strings literais duplicadas tornam o processo de refatoração propenso a erros, pois se deve atualizar todas as ocorrências.

Por outro lado, as constantes podem ser referenciadas de muitos lugares, mas precisam ser atualizadas apenas em um único local.

```
+ DBHelper.COLUNA_IDFESTA + " INTEGER PRIMARY KEY, "  
+ DBHelper.COLUNA_NOMEFESTA + " TEXT, "
```

```
+ DBHelper.COLUNA_PRECOFESTA + " TEXT, "  
+ DBHelper.COLUNA_CRIADORFESTA + " INTEGER, "  
+ DBHelper.COLUNA_DESCRICAOFESTA + " TEXT)";  
db.execSQL(QUERY_COLUNAFESTA);  
}
```

```
String QUERY_COLUNAFESTA = "CREATE TABLE " + DBHelper.TABELA_FESTA + "("  
+ DBHelper.COLUNA_IDFESTA + " INTEGER PRIMARY KEY, "  
+ DBHelper.COLUNA_NOMEFESTA + " 1 TEXT, "
```

```
+ DBHelper.COLUNA_PRECOFESTA + " 2 TEXT, "  
+ DBHelper.COLUNA_CRIADORFESTA + " INTEGER, "  
+ DBHelper.COLUNA_DESCRICAOFESTA + " TEXT)";  
db.execSQL(QUERY_COLUNAFESTA);  
}
```

```
private void criarTabelaUsuario(SQLiteDatabase db) {
    String QUERY_COLUNAUSUARIO = "CREATE TABLE " + DBHelper.TABELA_USUARIO + "("
        + DBHelper.COLUNA_ID + " INTEGER PRIMARY KEY, " + DBHelper.COLUNA_NOME
        + " 3 " TEXT, " + DBHelper.COLUNA_EMAIL + " TEXT," + DBHelper.COLUNA_SENHA
        + " TEXT)";
    db.execSQL(QUERY_COLUNAUSUARIO);
}
```

Se um campo privado for declarado, mas não usado no programa, ele poderá ser considerado código morto e, portanto, deverá ser removido. Isso melhora a capacidade de manutenção porque os desenvolvedores não se perguntarão para que serve a variável.

```
public class LoginActivity extends AppCompatActivity {
    private EditText editEmail;
    private EditText editSenha;
    private UsuarioServices usuarioServices = new UsuarioServices(this);
    private SessaoUser sessaoUser ;
}
```

Os programadores não devem comentar o código, pois ele incha os programas e reduz a legibilidade. O código não utilizado deve ser excluído e pode ser recuperado do histórico de controle de origem, se necessário.

```
public void onClick(View v) {
    String email = editEmail.getText().toString();
    String senha = editSenha.getText().toString();
    Usuario res = usuarioServices.getUsuario(email, senha);
    if (res != null){
        //sessaoUser.setEmail(email);
    }
}
```

Se um campo privado for declarado, mas não usado no programa, ele poderá ser considerado código morto e, portanto, deverá ser removido. Isso melhora a capacidade de manutenção porque os desenvolvedores não se perguntarão para que serve a variável.

Quando o valor de um campo privado é sempre atribuído aos métodos de uma classe antes de ser lido, ele não está sendo usado para armazenar informações da classe. Portanto, ela deve se tornar uma variável local nos métodos relevantes para evitar qualquer mal-entendido.

```
public class RegisterActivity extends AppCompatActivity {  
    private boolean task = false;
```

```
private void showExceptionToast(Exception e) {
```

```
private EditText editNome;  
private EditText editEmail;  
private EditText editSenha;  
private EditText editConfSenha;  
private Button registrarButton;
```

Se uma variável local for declarada mas não usada, será um código morto e deve ser removida. Isso aumentará a capacidade de manutenção, pois os desenvolvedores não se perguntarão para que serve a variável.

```
private boolean validarCampos() {  
    String nome = editNome.getText().toString();  
    String email = editEmail.getText().toString();  
    String senha = editSenha.getText().toString();  
    String confSenha = editConfSenha.getText().toString();  
    limparErros();  
    View focusView = null;
```

```
private boolean validarSenha(String senha, String confSenha) {  
    View focusView;
```

```
private boolean validarEmailExiste(String email) {  
    View focusView;
```



```
private boolean validarNomeExiste(String nome) {  
    View focusView;
```

```
private void cadastrar() {  
    String nome = editNome.getText().toString();  
    String email = editEmail.getText().toString();  
    String senha = editSenha.getText().toString();  
    String confSenha = editConfSenha.getText().toString();
```

O retorno de instruções literais booleanas agrupadas em instruções if-then-else deve ser simplificado.

Da mesma forma, as invocações de métodos agrupadas em if-then-else diferentes apenas de literais booleanos devem ser simplificadas em uma única invocação.

```
if (!resultadoValidacoes(nome, email, senha, confSenha)) return false;
```

```
private boolean resultadoValidacoes(String nome, String email, String senha, String  
    if (!validarNomeExiste(nome)) return false;  
    if (!validarEmailExiste(email)) return false;  
    if (!validarSenha(senha, confSenha)) return false;
```

Um armazenamento morto ocorre quando uma variável local recebe um valor que não é lido por nenhuma instrução subsequente. Calcular ou recuperar um valor apenas para substituí-lo ou jogá-lo fora pode indicar um erro grave no código. Mesmo que não seja um erro, é na melhor das hipóteses um desperdício de recursos. Portanto, todos os valores calculados devem ser utilizados.

```
if (senha.isEmpty()) {  
    editSenha.setError("O Campo esta vazio");  
    focusView = editSenha;
```

```
        return false;
    } else if (!validarSenhaIguais(senha, confSenha)) {
        editSenha.setError("Senhas devem ser iguais");
        focusView = editSenha;
```

```
    if (email.isEmpty()) {
        editEmail.setError("O Campo esta vazio");
        focusView = editEmail;
```

```
        return false;
    } else if (!validarEmail(email)) {
        editEmail.setError("Email inválido");
        focusView = editEmail;
```

```
    if (nome.isEmpty()) {
        editNome.setError("O campo esta vazio!");
        focusView = editNome;
```

```
        return false;
    } else if (!validarNome(nome)) {
        editNome.setError("Nome inválido, não aceito caracteres especiais");
        focusView = editNome;
```

Complexidade Ciclomática

Complexidade	Avaliação
--------------	-----------

1-10	Método simples. Baixo risco.
11-20	Método razoavelmente complexo. Moderado risco.
21-50	Método muito complexo. Elevado risco.
51-N	Método de altíssimo risco e bastante instável.



Eventos

Cyclomatic Complexity 58

dominio	14
gui	29
negocio	6
persistencia	9

Gui de eventos

Cyclomatic Complexity 29

 ConsultarEventoActivity.java	2
 CriarEventoActivity.java	3
 EditarEventoActivity.java	3
 ListaEventoActivity.java	5
 RecyclingAdapterFesta.java	16

dominio de eventos

Cyclomatic Complexity 14

 Evento.java	14
---	----

negócio de eventos

Cyclomatic Complexity 6

 EventoServices.java	6
---	---

persistência de eventos

Cyclomatic Complexity 9

 EventoDAO.java	9
--	---

Infra

Cyclomatic Complexity 33

app	2
gui	11
negocio	6
persistencia	14

Infra app

Cyclomatic Complexity 2

SoResenhaApp.java	2
-------------------	---

Infra gui

Cyclomatic Complexity 11

ConfigurationActivity.java	5
MenuActivity.java	4
SplashScreenActivity.java	2



Infra negócio

Cyclomatic Complexity 6

SessaoUsuario.java	4
SoresenhaAppException.java	2

Infra persistência

Cyclomatic Complexity 14

 DBHelper.java	7
 SessaoUser.java	7

Usuário

Cyclomatic Complexity 56

 dominio	11
 gui	33
 negocio	5
 persistencia	7



Usuário domínio

Cyclomatic Complexity 11

 Usuario.java	11
--	----

Usuário gui

Cyclomatic Complexity 33

 LoginActivity.java	5
 RegisterActivity.java	28

Usuário negócio

Cyclomatic Complexity 5

 UsuarioServices.java	5
--	---

Usuário persistência

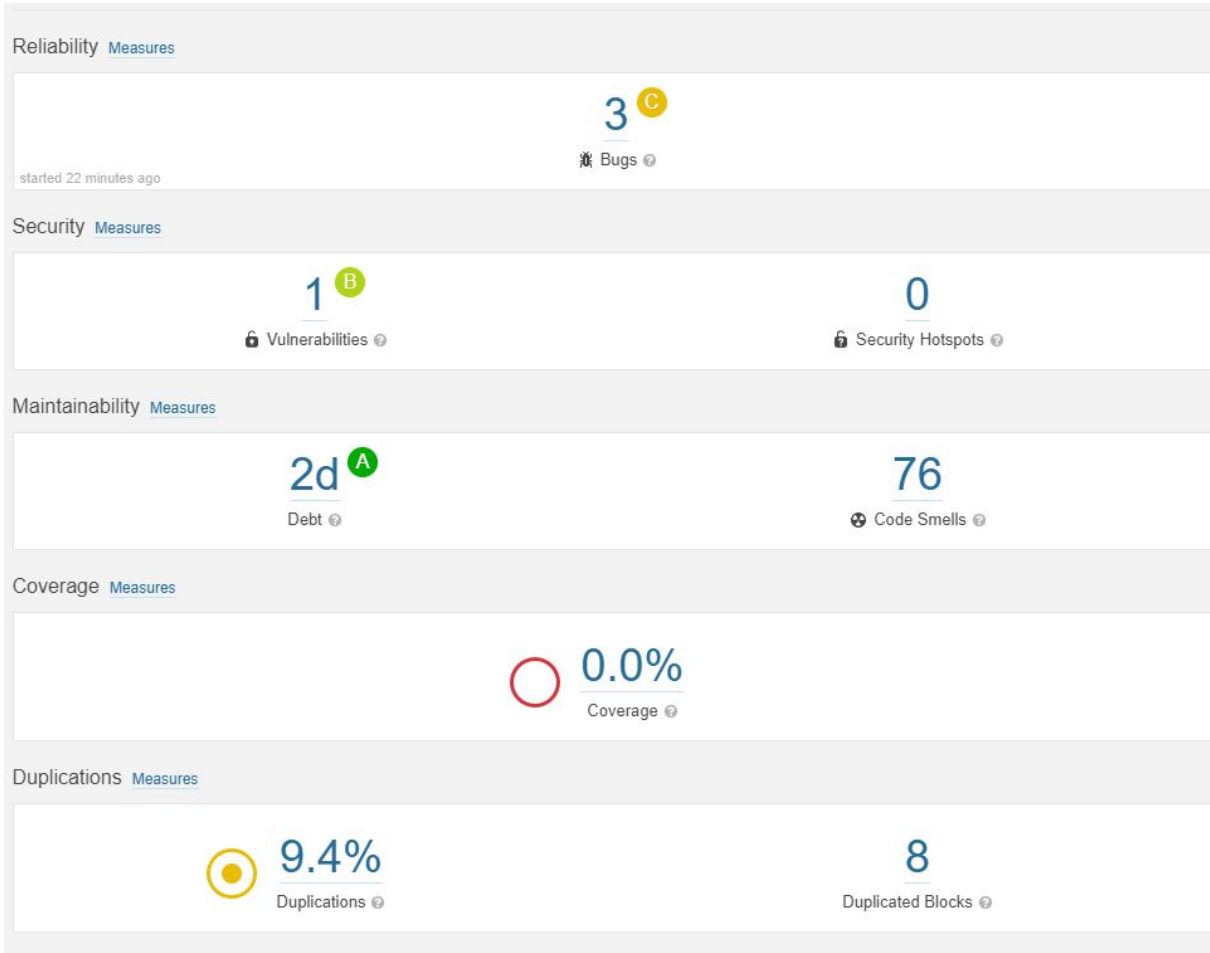
Cyclomatic Complexity 7

 UsuarioDAO.java

7

Código analisado com a API Sonarqube em 04/11/2019

Análise geral do código 2.0



BUGS


```
String cleanString = preco.replaceAll("[R$,.]", "");
new BigDecimal(cleanString)
    .setScale(2, BigDecimal.ROUND_FLOOR)
    .divide(new BigDecimal(100), BigDecimal.ROUND_FLOOR);
```

Quando a chamada para uma função não tem efeitos, qual é o sentido de fazer a chamada se os resultados são ignorados? Nesse caso, a chamada da função é inútil e deve ser descartada ou o código-fonte não se comporta conforme o esperado.

Esse mesmo erro foi notificado duas vezes

Code Smells presentes no código

Erros sublinhados de vermelho

Throwable.printStackTrace (...) imprime um Throwable e seu rastreamento de pilha em algum fluxo. Por padrão, esse fluxo System.Erro, que inadvertidamente pode expor informações confidenciais.

```
try {
    evento.setDate(DBHelper.dateTimeFormat.parse(cursor.getString(cursor.getColumnIndex(DBHelper.COLUNA_DATAFESTA))));
} catch (ParseException e) {
    evento.setDate(new Date());
    e.printStackTrace();
```

O requisito para uma cláusula padrão é a programação defensiva. A cláusula deve tomar as medidas apropriadas ou conter um comentário adequado sobre o motivo pelo qual nenhuma ação foi tomada.

Mesmo erro notificado três vezes

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
```

Strings literais duplicadas tornam o processo de refatoração propenso a erros, pois você deve atualizar todas as ocorrências.

```

        eventoDAO.update(evento);
    } else {
        throw new SoresenhaAppException( 1 "Usuario tentou alterar banco sem ser parceiro");
    }

```

```

        return eventoDAO.cadastrar(evento);
    } else {
        throw new SoresenhaAppException( 2 "Usuario tentou alterar banco sem ser parceiro");
    }
}

```

```

    } else {
        throw new SoresenhaAppException( 3 "Usuario tentou alterar banco sem ser parceiro");
    }
}

```

A atualização correta de um campo estático a partir de um método não estático é difícil de corrigir e pode facilmente levar a erros se houver várias instâncias de classe e / ou vários threads em execução. Idealmente, os campos estáticos são atualizados apenas a partir de métodos estáticos sincronizados.

Esta regra levanta um problema sempre que um campo estático é atualizado a partir de um método não estático.

```

super.onCreate();
context = this;

```

```

@Override
public void beforeTextChanged(CharSequence s, int start, int count, int after) {

```

```

@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {

```

Defina uma constante em vez de duplicar esse "TEXT", 4 vezes

```

String QUERY_COLUNAFESTA = "CREATE TABLE " + TABELA_FESTA + "("
    + COLUNA_IDFESTA + " INTEGER PRIMARY KEY, "
    + COLUNA_NOMEFESTA + 1 " TEXT, "

```

```

+ COLUNA_PRECOFESTA + " 2 " TEXT, "
+ COLUNA_DATAFESTA + " 3 " TEXT, "
+ COLUNA_CRIADORFESTA + " INTEGER, "
+ COLUNA_DESCRICAOFESTA + " TEXT)";

```

```
db.execSQL(QUERY_COLUNAFFESTA);
```

```

private void criarTabelaUsuario(SQLiteDatabase db) {
    String QUERY_COLUNAUSUARIO = "CREATE TABLE " + DBHelper.TABELA_USUARIO + "("
        + DBHelper.COLUNA_ID + " INTEGER PRIMARY KEY, " + DBHelper.COLUNA_SERPARCEIRO
        + " INTEGER DEFAULT 0, " + DBHelper.COLUNA_NOME + " 4 " TEXT, " + DBHelper.COLUNA_EMAIL
        + " TEXT," + DBHelper.COLUNA_SENHA
        + " TEXT)";
}

```

Blocos de código duplicados 2 classes iguais que poderia facilmente ser uma

```

private EditText editNome;
private EditText editDesc;
private EditText editPreco;
private EditText editDate;
private EditText editHora;
private EventoServices eventoServices = new EventoServices(this);
private Calendar eventoDate = Calendar.getInstance();
private SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
private SimpleDateFormat horaFormat = new SimpleDateFormat("kk:mm");

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_criar_evento);
    editNome = findViewById(R.id.nomeFestaedit);
    editPreco = findViewById(R.id.precoFestaedit);
    editDesc = findViewById(R.id.descFestaEdit);
    editDate = findViewById(R.id.dataFestaEdit);
    editHora = findViewById(R.id.horaFestaEdit);
    Button criarBtn = findViewById(R.id.criarFestabutton);
    criarListeners(criarBtn);
}

```

```

private void criarListeners(Button criarBtn) {
    criarBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (!validate()) return;
            String nome = editNome.getText().toString();
            String descricao = editDesc.getText().toString();
            String preco = editPreco.getText().toString();
            String cleanString = preco.replaceAll("[R$,.]", "");
            BigDecimal parsed = new BigDecimal(cleanString)
                .setScale(2, BigDecimal.ROUND_FLOOR)
                .divide(new BigDecimal(100), BigDecimal.ROUND_FLOOR);
            Evento evento = new Evento(nome, descricao, parsed, eventoDate.getTime());
            try {
                eventoServices.criar(evento);
            } catch (SoresenhaAppException e) {
                Toast.makeText(v.getContext(), "Você não tem permissão para alterar eventos", Toast.LENGTH_LONG).show();
                startActivity(new Intent(v.getContext(), ListaEventoActivity.class));
                e.printStackTrace();
            }
            Intent backMenu = new Intent(CriarEventoActivity.this, ListaEventoActivity.class);
            backMenu.setFlags(backMenu.getFlags() | Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_SINGLE_TOP);
            startActivity(backMenu);
        }
    });
}
listenersData();
editPreco.addTextChangedListener(new MoneyTextMask(editPreco));
}

```

```

private void listenersData() {
    final DatePickerDialog.OnDateSetListener date = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
            editDate.setText(dateFormat.format(eventoDate.getTime()));
            eventoDate.set(year, monthOfYear, dayOfMonth);
            editDate.setError(null);
        }
    };

    final TimePickerDialog.OnTimeSetListener time = new TimePickerDialog.OnTimeSetListener() {
        @Override
        public void onTimeSet(TimePicker timePicker, int i, int i1) {
            eventoDate.set(
                eventoDate.get(Calendar.YEAR),
                eventoDate.get(Calendar.MONTH),
                eventoDate.get(Calendar.DAY_OF_MONTH),
                i,
                i1
            );
            String hora = horaFormat.format(eventoDate.getTime());
            editHora.setText(hora);
            editHora.setError(null);
        }
    };

    editDate.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            new DatePickerDialog(v.getContext(), date, eventoDate
                .get(Calendar.YEAR), eventoDate.get(Calendar.MONTH),
                eventoDate.get(Calendar.DAY_OF_MONTH)).show();
        }
    });

    editHora.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            new TimePickerDialog(
                v.getContext(),
                time,
                eventoDate.get(Calendar.HOUR_OF_DAY),
                eventoDate.get(Calendar.MINUTE),
                true
            ).show();
        }
    });
}

```

```

private boolean validate() {
    limparCampos();
    if (!validarCampos()) return false;
    return true;
}

private boolean validarCampos() {
    boolean res = true;
    if (!validarPreco()) res = false;
    if (!validarNome()) res = false;
    if (!validarDescricao()) res = false;
    if (!validarData()) res = false;
    if (!validarHora()) res = false;
    return res;
}

private boolean validarNome() {
    String nome = editNome.getText().toString();
    if (nome.isEmpty()) {
        editNome.setError("Nome não pode ser vazio");
        return false;
    }
    return true;
}

private boolean validarDescricao() {
    String desc = editDesc.getText().toString();
    if (desc.isEmpty()) {
        editDesc.setError("Descrição não pode ser vazia");
        return false;
    }
    return true;
}

private boolean validarPreco() {
    String preco = editPreco.getText().toString();
    if (preco.isEmpty()) {
        editPreco.setError("Preço não pode ser vazio");
        return false;
    }
}

```



```

        return true;
    }

    private boolean validarDescricao() {
        String desc = editDesc.getText().toString();
        if (desc.isEmpty()) {
            editDesc.setError("Descrição não pode ser vazia");
            return false;
        }
        return true;
    }

    private boolean validarPreco() {
        String preco = editPreco.getText().toString();
        if (preco.isEmpty()) {
            editPreco.setError("Preço não pode ser vazio");
            return false;
        }
        try {
            String cleanString = preco.replaceAll("[R$,.]", "");
            new BigDecimal(cleanString)
                .setScale(2, BigDecimal.ROUND_FLOOR)
                .divide(new BigDecimal(100), BigDecimal.ROUND_FLOOR);
        } catch (Exception e) {
            editPreco.setError("Preço não é conversível em número");
            return false;
        }
        return true;
    }

    private boolean validarData() {
        String data = editDate.getText().toString();
        Calendar now = Calendar.getInstance();
        now.set(Calendar.HOUR_OF_DAY, 0);
        now.set(Calendar.MINUTE, 0);
        if (data.isEmpty()) {
            editDate.setError("Data não pode ser vazia");
            return false;
        } else if (eventoDate.before(now)) {
            editDate.setError("Esta data já passou");
            return false;
        }
    }

```

```

        return true;
    }

    private boolean validarData() {
        String data = editDate.getText().toString();
        Calendar now = Calendar.getInstance();
        now.set(Calendar.HOUR_OF_DAY, 0);
        now.set(Calendar.MINUTE, 0);
        if (data.isEmpty()) {
            editDate.setError("Data não pode ser vazia");
            return false;
        } else if (eventoDate.before(now)) {
            editDate.setError("Esta data já passou");
            return false;
        }
        return true;
    }

    private boolean validarHora() {
        String hora = editHora.getText().toString();
        Calendar now = Calendar.getInstance();
        if (hora.isEmpty()) {
            editHora.setError("Hora não pode ser vazia");
            return false;
        } else if (eventoDate.before(now)) {
            editHora.setError("Esta hora já passou");
            return false;
        }
        return true;
    }

    private void limparCampos() {
        editPreco.setError(null);
        editDesc.setError(null);
        editNome.setError(null);
        editDate.setError(null);
        editHora.setError(null);
    }
}

```

Concatenação desnecessária e que pode impactar no desempenho

```

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    logger.log(Level.INFO, "Upgrading DB from "+oldVersion+" to " +newVersion);
}

```

Se um campo privado for declarado, mas não usado no programa, ele poderá ser considerado código morto e, portanto, deverá ser removido. Isso melhora a manutenção, pois os desenvolvedores não se perguntarão para que serve a variável.

```
public class ParceiroServices {  
    private ParceiroDAO parceiroDAO;
```

Parâmetros não utilizados são enganosos. Quaisquer que sejam os valores passados para esses parâmetros, o comportamento será o mesmo.

```
public ParceiroServices(Context context) {this.parceiroDAO = new ParceiroDAO();}
```

Os programadores não devem comentar o código, pois ele incha os programas e reduz a legibilidade.

O código não utilizado deve ser excluído e pode ser recuperado do histórico de controle de origem, se necessário.

```
//public Parceiro getParceiro(String empresa){return parceiroDAO.get(empresa);}
```

```
//
```

```
// public boolean CheckIfisParceiro(Usuario usuario){return parceiroDAO.check(usuario);}
```

Uma vez obsoleto, classes e interfaces e os seus membros devem ser evitados, ao invés de usado, herdada ou estendida. A descontinuação é um aviso de que a classe ou interface foi substituída e, eventualmente, será removida. O período de suspensão permite que você faça uma transição suave para longe do envelhecimento, logo-a-ser-aposentado tecnologia.

```

public void onClick(View v) {
    if (!validate()) return;
    String nome = editNome.getText().toString();
    String descricao = editDesc.getText().toString();
    String preco = editPreco.getText().toString();
    String cleanString = preco.replaceAll("[R$,.]", "");
    BigDecimal parsed = new BigDecimal(cleanString)
        .setScale(2, BigDecimal.ROUND_FLOOR)

```

```

        .divide(new BigDecimal(100), BigDecimal.ROUND_FLOOR);

```

Imports não utilizados

```

import com.ufrpe.bsi.soresenha.R;
import com.ufrpe.bsi.soresenha.infra.app.MecanismoPersistencia;

```

```

import com.ufrpe.bsi.soresenha.infra.gui.MenuActivity;
import com.ufrpe.bsi.soresenha.infra.negocio.SessaoUsuario;
import com.ufrpe.bsi.soresenha.infra.persistencia.SessaoUser;

```

```

import com.ufrpe.bsi.soresenha.infra.negocio.SessaoUsuario;

```

Classe vazia

```

package com.ufrpe.bsi.soresenha.parceiro.persistencia;

public class ParceiroDAO {

```

Complexidade Ciclomática 2.0






Complexidade	Avaliação
1-10	Método simples. Baixo risco.
11-20	Método razoavelmente complexo. Moderado risco.
21-50	Método muito complexo. Elevado risco.
51-N	Método de altíssimo risco e bastante instável.

Eventos

Cyclomatic Complexity 125

dominio	18
gui	89
negocio	9
persistencia	9

Gui de eventos

 ConsultarEventoActivity.java	2
 CriarEventoActivity.java	30
 EditarEventoActivity.java	32
 ListaEventoActivity.java	8
 RecyclingAdapterFesta.java	17

domínio de eventos

Cyclomatic Complexity 18

 Evento.java	18
---	----

negócio de eventos

Cyclomatic Complexity 9

 EventoServices.java	9
---	---

persistência de eventos

Cyclomatic Complexity 9

 EventoDAO.java	9
--	---

Infra

Cyclomatic Complexity **44**

📁 app	5
📁 gui	11
📁 helper	6
📁 negocio	8
📁 persistencia	14

Infra app

Cyclomatic Complexity **5**

📄 MecanismoPersistencia.java	3
📄 SoResenhaApp.java	2



Infra gui

Cyclomatic Complexity **11**

📄 ConfigurationActivity.java	5
📄 MenuActivity.java	4
📄 SplashScreenActivity.java	2

Infra negócio

Cyclomatic Complexity 8

 SessaoUsuario.java	6
 SoresenhaAppException.java	2

Infra Helper

Cyclomatic Complexity 6

 MoneyTextMask.java	6
--	---





Infra persistência

Cyclomatic Complexity 14

 DBHelper.java	7
 SessaoUser.java	7

Usuário

Cyclomatic Complexity 96

 dominio	13
 gui	57
 negocio	15
 persistencia	11




Usuário domínio

Cyclomatic Complexity 13

 Usuario.java	13
--	----

Usuário gui

Cyclomatic Complexity 57

 EditUserActivity.java	22
 LoginActivity.java	5
 RegisterActivity.java	30

Usuário negócio

Cyclomatic Complexity 15

 UsuarioServices.java	15
--	----

Usuário persistência

Cyclomatic Complexity 11

 UsuarioDAO.java	11
---	----

Parceiro

Cyclomatic Complexity 5	
 dominio	4
 negocio	1
 persistencia	0

Parceiro domínio

Cyclomatic Complexity 4	
 Parceiro.java	4

Parceiro Negócio

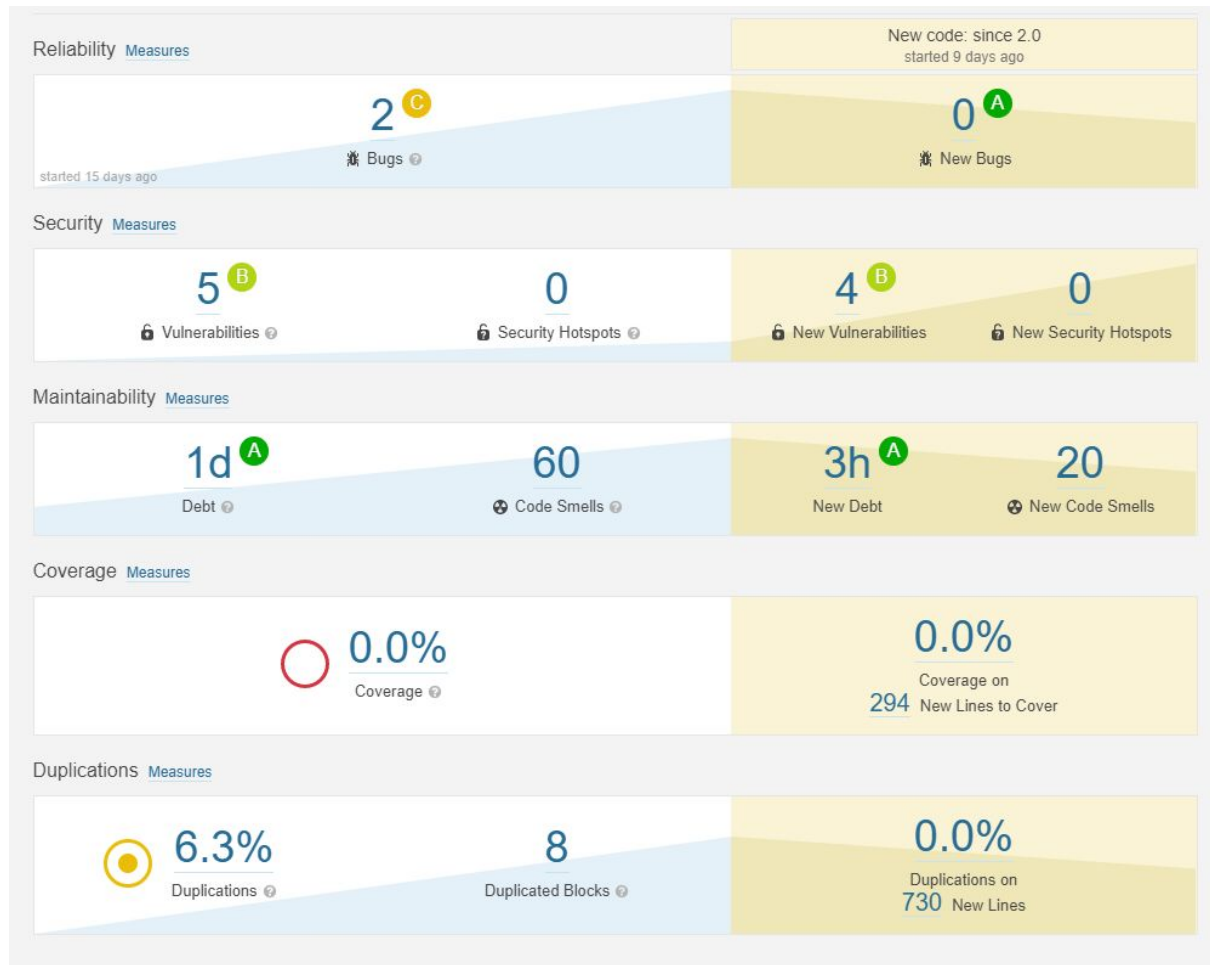
Cyclomatic Complexity 1	
 ParceiroServices.java	1

Parceiro persistência

Cyclomatic Complexity 0	
 ParceiroDAO.java	0

Código analisado com a API Sonarqube em 11/11/2019

Análise geral do código 3.0



Code Smells presentes no código

Erros sublinhados de vermelho

Os valores estão sujeitos a alterações em qualquer parte do código

```
public class Avaliacao {  
    public long id;
```

eventos/dominio/Avaliacao.java os 3 ids e o tipo Avaliação

Campo privado não usado

```
public class EventoServices {  
    private EventoDAO eventoDAO;  
    private UsuarioServices usuarioServices;
```

eventos/negocio/EventoServices.java

Parâmetro não usado

```
}  
  
public List<Usuario> getListParticipantes(Evento 1 evento){
```

/eventos/persistencia/EventoDAO.java

Complexidade Ciclomática 3.0

Complexidade	Avaliação
1-10	Método simples. Baixo risco.
11-20	Método razoavelmente complexo. Moderado risco.
21-50	Método muito complexo. Elevado risco.
51-N	Método de altíssimo risco e bastante instável.

Eventos

Cyclomatic Complexity **174**

dominio	27
gui	105
negocio	20
persistencia	22




Gui de eventos

Cyclomatic Complexity 105

 ConsultarEventoActivity.java	14
 CriarEventoActivity.java	28
 EditarEventoActivity.java	30
 ListaEventoActivity.java	8
 RecyclingAdapterFesta.java	18
 RecyclingAdapterParticipante.java	7



domínio de eventos

Cyclomatic Complexity 27

 Avaliacao.java	10
 Evento.java	17
 TipoAvaliacao.java	0



negócio de eventos

Cyclomatic Complexity 20

 AvaliacaoServices.java	8
 EventoServices.java	12

persistência de eventos

Cyclomatic Complexity 22

 AvaliacaoDAO.java	11
 EventoDAO.java	11



Infra

Cyclomatic Complexity 50

 app	5
 gui	12
 helper	7
 negocio	11
 persistencia	15




Infra app

Cyclomatic Complexity 5

 MecanismoPersistencia.java	3
 SoResenhaApp.java	2


Infra gui

Cyclomatic Complexity **12**

 ConfigurationActivity.java	6
 MenuActivity.java	4
 SplashScreenActivity.java	2



Infra negócio

Cyclomatic Complexity **11**

 SessaoUsuario.java	9
 SoresenhaAppException.java	2



Infra Helper

Cyclomatic Complexity **7**

 BigDecimalUtil.java	1
 MoneyTextMask.java	6

Infra persistência

Cyclomatic Complexity **15**

 DBHelper.java	8
 SessaoUser.java	7

Usuário

Cyclomatic Complexity 95

 dominio	14
 gui	63
 negocio	8
 persistencia	10




Usuário domínio

Cyclomatic Complexity 14

 TipoUsuario.java	0
 Usuario.java	14

Usuário gui

Cyclomatic Complexity 63

 EditUserActivity.java	27
 LoginActivity.java	6
 RegisterActivity.java	30

Usuário negócio

Cyclomatic Complexity 8

 UsuarioServices.java	8
--	---



Usuário persistência

Cyclomatic Complexity 10

 UsuarioDAO.java	10
---	----

Parceiro

Cyclomatic Complexity 5

 dominio	4
 negocio	1
 persistencia	0

Parceiro domínio

Cyclomatic Complexity 4

 Parceiro.java	4
---	---

Parceiro Negócio

Cyclomatic Complexity 1

 ParceiroServices.java

1

Parceiro persistência

Cyclomatic Complexity 0

 ParceiroDAO.java

0