Eu, Agente!

Sumário

[**Introdução**](#_x6j9pq9qfgqc) **4**

[**Visão do Sistema**](#_4nkb6zcdnbrk) **4**

[Instrução do Problema](#_9pegdtwicik6) 4

[Oportunidades de Negócio](#_d602xh2jzkhb) 4

[Objetivos de Negócio](#_tax82hdwdf9d) 4

[Usuários](#_axj1eerqiz5u) 5

[Risco do Negócio](#_1ue0iktl2mgi) 5

[**Escopo do Produto**](#_pp2fnl34d0b1) **5**

[Backlog do Produto](#_hyhyzcih4bpa) 5

[Principais funcionalidades](#_k1pdd3gg525o) 7

[**Telas**](#_x8gt5vrzksyf) **8**

[User Stories](#_3dtqb47jp1ea) 8

[Limitação](#_h7f1apdcb993) 8

[Sugestão de melhoria](#_4uonfrc777sy) 8

[**Gerenciamento de Configuração**](#_41z2c47aonyx) **8**

[Definição de tags de uma versão](#_1ze2ox2s5ygl) 9

[**Integração Contínua**](#_46kru0maixpn) **9**

[**Tecnologias**](#_3dnke5h33zpu) **9**

[Tecnologias gerais](#_jrfwm0vatuyo) 9

[Aplicativo](#_pm0y4iyfk1yf) 10

[Backend](#_87j19yguixes) 10

[**Arquitetura**](#_x8gt5vrzksyf) **11**

[Aplicativo](#_asbf2tojhwy2) 11

[Backend](#_q9513bxlhmw) 11

[**Instalação e execução**](#_x8gt5vrzksyf) **13**

[Aplicativo](#_5dyjae3vxtd) 13

[Backend](#_7s6s8cdca910) 14

[**Testes**](#_flvuf0nhz6wx) **15**

[Testes automatizados utilizando o Espresso](#_6fxn0kbikp69) 15

[Testes Unitários](#_djtutsljv4nm) 16

[Cobertura de Teste](#_ha5xlnmwx8wd) 17

[**Análise Estática**](#_v32ji1178eep) **19**

[Visual Studio](#_iaow9ojbmpm1) 19

[SonarQube](#_8ikh3pod90sn) 20

[Painel Aplicativo](#_ej8q7g7k7ni8) 21

[Painel Backend](#_ebmm4aypfeh6) 22

# **Introdução**

Esse documento fornece uma visão abrangente do produto, desde a definição do problema até informações mais detalhadas do desenvolvimento da solução como tecnologias utilizadas, frameworks e padrões de projetos.

# **Visão do Sistema**

## **Instrução do Problema**

Os moradores de Campo Grande, Mato Grosso do Sul, enfrentam diversas complicações quando o assunto é Dengue ou outras doenças similares a ela. O controle da endemia é dada pelos agentes de saúde que vão em casa para análise e coletam as informações que irão gerar dados usados para montar o mapa da saúde da Dengue que a SESAU disponibiliza. As dificuldades que os agentes possuem para vistoriar as residências

## **Oportunidades de Negócio**

Essa aplicação ajudaria os moradores a vistoriar a própria residência e facilitar o acesso ao mapa da área que o mesmo mora, para análise do nível de dengue informado. Além disso, facilita o acesso a informações sobre a endemia e ações preventivas a proliferação do mosquito.

Para a Secretária de Saúde, seria uma maneira de engajar a população na parte de coleta de possíveis focos e possíveis pessoas doentes. Podendo assim, usar a aplicação para comparar os dados que são gerados através dos agentes de saúde da cidade. Chegando a conclusão das áreas mais afetadas pela dengue e facilitando a distribuição de recursos para prevenir a Dengue nas regiões.

## **Objetivos de Negócio**

Com relação ao projeto, foram depreendidos os objetivos de negócio a seguir:

* Facilitar a coleta de dados sobre a dengue em Campo Grande;
* Facilitar a distribuição de informações e prevenções contra a dengue;
* Engajar os moradores a vistoriar a residência própria;

## **Usuários**

Os usuários do sistema são classificados em dois papéis: Administrador e Morador.

O morador possui acesso a todas as funções disponíveis no aplicativo Android, como relacionadas a módulo de usuário, cadastro de pessoa doente, denúncia de foco, checklist de foco em residência, informações e mapas.

O administrador além das funções do morador, também pode cadastrar outros administradores e gerar relatórios do sistema a partir do módulo web.

## **Risco do Negócio**

Os riscos de negócio observados foram:

* Atraso na entrega dos releases (P=25%, I=8);
* Baixa adesão do sistema por parte dos usuários (P=40%, I=10).

# **Escopo do Produto**

## **Backlog do Produto**

Estado Atual dos Itens no Backlog do Produto

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Dentro de escopo concluídos |  | Fora do escopo |  | Fora do escopo, porém concluídas |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EPIC: Estrutura do sistema | | | | | |
|  | - | | Repositório | | |
|  | Configuração de entidades | | | |
|  | Definição de contexto de banco de dados | | | |
|  | Configuração Swagger | | | |
|  | Configuração pipeline | | | |
|  | Configuração repositório | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | - | | API para acesso | | |
|  | Controller de acesso usuário | | | |
|  | Controller de repositories | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EPIC: Lançamento de ocorrências | | | | | |
|  | Feature 1 | | Morador | | |
|  | CRUD | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Feature 2 | | Endereço | | |
|  | CRUD | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Feature 3 | | Usuário | | |
|  | CRUD | | | |
|  | Login | | | |
|  | Perfil | | | |
|  | Recuperar Senha | | | |
|  | Confirmação de email | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Feature 4 | | Realizar lançamento de pessoas doentes | | |
|  | CRUD | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Feature 4 | | Assinalar lista de possíveis focos na residência | | |
|  | CRUD | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Feature 3 | | Exibir mapa da cidade | | |
|  | Comunicação API Microsoft | | | |
|  | Mapa de calor | | | |
|  | Mapa de agrupamento | | | |
|  | FIltro por período | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Feature 9 | | Endemias | | |
|  | CRUD | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EPIC: Lançamento de Focos | | | | | |
|  | Feature 6 | | Registrar denúncias de locais com possíveis focos | | |
|  | CRUD | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Feature 7 | | Assinalar lista de possíveis focos na residência | | |
|  | CRUD | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Feature 8 | | Gerar histórico de focos | | |
|  | Implementação biblioteca para exportação | | | |
|  | Gerar relatórios | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EPIC: Informações | | | | | |
|  |  | Lista de informações importantes | | | |
|  | Lista de prevenções | | | |
|  | Tutorial da aplicação | | | |

## **Principais funcionalidades**

A descrição das principais funcionalidades podem ser lidas no documento de Visão e Escopo.

## **Telas**

A descrição de telas podem ser lidas no documento de Especificação de Requisitos.

## **User Stories**

As estórias de usuário podem ser lidas no documento de Especificação de Requisitos.

## **Limitação**

O Eu, Agente por enquanto irá registrar dados apenas através do aplicativo Android.

## **Sugestão de melhoria**

Recomenda-se como próximos passos a implementação de denúncias através da plataforma web e de dashboard com dados dos relatórios para acesso da população.

# **Gerenciamento de Configuração**

Para auxiliar o gerenciamento de configuração foi utilizado uma ferramenta muito completa que é o Azure DevOps. Com ele foi possível controlar a versão dos artefatos e a rastreabilidade entre eles.

Para o controle de versão foi estabelecido o padrão de branch a seguir:

* **Master**
  + A branch master contém todo código que está atualmente em produção, ou seja, todo o que código que está nessa branch já foi validado pelos proponentes.
* **Dev**
  + A branch dev possui todo o código que está atualmente na master mais alterações que ainda estão em teste e ainda não foram validadas pelos proponentes. Todo código aqui precisar ter sido validado por um desenvolvedor que não seja o mesmo que desenvolveu a task.
  + Essa branch é derivada da branch Master.
  + Essa branch recebe alterações da branch Task\_n.
* **Task\_n**
  + Para cada requisito do sistema haverá mapeado uma task\_n onde n é o número específico da task em questão. Todo código aqui está em pleno desenvolvimento onde poderá ocorrer diversas alterações até ser aberto um Pull Request para ser merjada na branch DEV.
  + Essa branch é derivada da branch Dev.

## **Definição de tags de uma versão**

A definição de tags de versão de uma release será definido com base no Semantic Versioning 2.0.0 (disponível em <https://semver.org/>).

De maneira resumida, o número dado a versão do sistema será descrito por MAJOR.MINOR.PATCH de maneira incremental sendo:

* MAJOR quando houver alterações onde exista a inclusão de um grande módulo de funcionalidades ou API’s de conexão se tornam incompatíveis;
* MINOR quando houver alterações com adição de novas funcionalidades;
* PATCH quando houver alterações para correções em funcionalidades já existentes.

# **Integração Contínua**

Foi utilizada integração contínua para verificação, compilação e publicação de containers dos módulos de backend no Docker Hub após aprovação de pull request através da função Pipeline do Azure DevOps (arquivo de configuração está contido junto a solução).

# **Tecnologias**

O Eu, Agente foi desenvolvido utilizando as seguintes tecnologias.

## **Tecnologias gerais**

* Azure DevOps para controle de versão, controle do processo de desenvolvimento, rastreabilidade entre produto e estória de usuário e integração contínua;
* PostgreSQL 12.2 como banco de dados;
* PGAdmin 4.11 como sistema gerenciador de banco de dados;
* SonarQube 8.2.0.32929 para análise do código;

## **Aplicativo**

* Android SDK 21 (Lollipop);
* Android Studio 3.6.1 como IDE de desenvolvimento do aplicativo Android;
* Espresso 3.2.0 para testes automatizado da UI do aplicativo;
* Gson 2.8.5 para manipulação de Json;
* Jacoco 0.8.5 para relatório de cobertura de teste;
* JUnit 4.12 para testes automatizado no aplicativo;
* Lottie 3.4.0 para manipulação de animações do Adobe After Effects;
* Mask-Edittext 1.1.1 para inclusão de máscaras em campos de entrada;
* Material design 1.1.0 como base de layout;
* Viacep 1.2.0 para busca de endereços;
* Volley 1.1.1 para requisições web;

## **Backend**

* .NET Core Framework 3.1 para o desenvolvimento da API e projeto Web;
* AutoMapper 7.0.0 para mapeamento de objetos;
* Docker 2.3.0.3 para virtualizar e encapsular a solução;
* Entity Framework Core 3.1.2 como ORM (Object-Relational Mapping);
* EPPlus 4.5.3.2 para manipulação de planilhas Excel;
* Identity Framework Core 3.1.2 para autenticação com utilização de token Bearer Token no formato JWT;
* Newtonsoft.Json 12.0.3 para manipulação de Json;
* NToastNotify 6.1.3 para manipulação de notificações toast no ambiente Web
* Swagger 5.1.0 para documentação da API;
* Visual Studio Community 2019 16.6.1 como IDE de desenvolvimento da API.

# **Arquitetura**

## **Aplicativo**

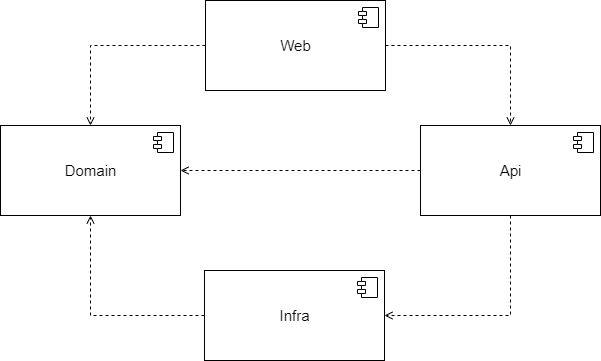
O aplicativo foi implementado com uma customização da arquitetura MVVM, onde o aplicativo tem a responsabilidade interação com o usuário e comunicação com a API e a lógica de negócio fica na API.

No diretório principal do projeto **(euagenteandroid)** existe a seguinte organização:

* **Model** *(Diretório)*
  + **dao:** *(Diretório)* Responsável com a comunicação com a API.
  + **entity:** *(Diretório)* Possui todas as classes serializadas para comunicação com a API.
  + **util:** *(Diretório)* Possui algumas configurações e arquivos que são reutilizados no sistema.
  + **DaoFacade:** *(Arquivo)* Responsável pela comunicação do View com o Dao.
* **View** *(Diretório)*
  + **bases:** *(Diretório)* Possui arquivos com código base para todas as activities. Onde cada activity estende o código base específico para a sua função.
  + **util:** *(Diretório)* Possui arquivos que são reutilizados nas Views.
  + **Activities:** *(Arquivos)* todas as activities que exercem papel de ViewModel.
* **assets** (Diretório): Possui arquivos de animações do Adobe After Effects.
* **res** (Diretório): Possui arquivos de recursos como layout, cores, imagens, menus e textos relacionados às telas.

## **Backend**

O backend da solução foi implementada com base nos princípios do DDD (Domain Driven Design). Nesta modelagem, a solução foi dividida em nas camadas web, api, domain e infra.



A camada **Web** foi desenvolvida em arquitetura MVC e tem a função de receber requisições do usuário e exibi-las.

A camada **Domain** é responsável pela implementação de classes, interfaces, DTOs e enums.

A camada de **Api** foi desenvolvida seguindo a recomendação da Microsoft em forma semelhante ao MVC, porém sem uma camada para View. Ela possui regras de negócio e é responsável por se comunicar com a infra para acesso ao banco de dados.

A camada de **Infra** é subdividida em duas camadas: **Data** e **CrossCutting**.

A subcamada **Data** é responsável por lidar com a persistência no banco de dados, enquanto a subcamada **CrossCutting** possui implementações que são utilizadas em mais de uma dessas camadas, como códigos para validação de CPF por exemplo, e acesso a serviços externos.

A solução faz uso de managers transientes definidos em inicialização do servidor e com utilização através de injeção de dependências quando necessário. Dessa forma o servidor gerenciará o número de instâncias de cada manager, mantendo singleton sempre que possível e escalando quando o servidor identifica gargalos.

O acesso a dados dentro da aplicação se deu utilizando do Entity Framework Core, como ferramenta de ORM (*Object-Relational Mapping*). Dessa forma, a solução permite a troca de um banco de dados alterando-se apenas a “Connection String”, que contém dados de conexão ao banco de dados, e alterando o driver de conexão no startup do servidor, não sendo necessário alteração de nenhuma lógica de dados.

Com relação a criação e alteração da estrutura de dados do banco de dados, a solução foi criada fazendo uso do padrão Migrations do .NetCore que possui a premissa code-first. As migrations são geradas após utilização de comando específico (vide documentação da Microsoft) e fornecem controle de versão da estrutura do banco de dados. Durante a inicialização do docker-compose, o servidor back-end atualiza o banco de dados automaticamente através de execução de atualização das migrations.

Para envio emails ao usuário, a solução realiza conexão SMTP com serviço Outlook. Para troca do serviço de e-mail, basta trocar os dados de conexão, como endereço do servidor, login e senha, no arquivo docker-compose.yml. Neste arquivo consta o email e senha atual do email criado para envio dessas informações.

O controle de acesso foi criado através do Identity Framework Core aliado com emissão e autenticação de Bearer Token utilizando-se JWT. A utilização de framework amplamente utilizado e validado garante maior segurança aos dados do usuário, sendo que dados sensíveis são salvos de forma criptografada. Já a utilização de token fornece forma de manutenção de login de forma comum as boas práticas do mercado atual. As permissões de acesso a um usuário foram criadas através de um sistema de Roles (papéis). A solução está configurada para realizar a validação do token através do servidor antes mesmo dela chegar a controller.

Por fim, vale ressaltar que a solução foi desenvolvida e configurada para o uso de DTO’s através do uso de padrão de InputModel’s. Dessa forma, sempre que é realizado uma requisição a WebAPI, o próprio servidor realiza a conferência dos dados recebidos com as configurações do input model e faz a aprovação ou reprovação da entrada com retorno de mensagem ao usuário antes mesmo da requisição chegar a controller.

# **Instalação e execução**

## **Aplicativo**

Para que o aplicativo possa ser disponibilizado a comunidade em geral, basta que sejam configurados o endereço da api, do servidor web e chave de acesso ao Bing Maps (pode ser obtida em <https://www.bingmapsportal.com/>) em model/util/Constants.java. Após isso, a apk poderá ser gerada e publicada na Play Store.

Para ambiente de desenvolvimento, os endereços de api e web podem ser deixados em branco. Nesta situação, a primeira tela do aplicativo será uma tela onde deverá ser inserido o IP da máquina que está rodando o backend via Docker.

## **Backend**

O backend está configurado para executar em containers Docker, sendo assim é necessário a instalação do Docker Engine compatível com o sistema operacional onde será executado.

Após clonar o projeto é necessário configurar as variáveis de ambiente do arquivo *docker-compose.yml*. Neste arquivo, deverão ser configurados o endereço do servidor web, do servidor da api, a chave de acesso ao Bing Maps Api, dados de acesso ao banco de dados e dados de acesso ao serviço de email. Além disso, para ambiente de produção, é necessário trocar a variável de ambiente ASPNETCORE\_ENVIRONMENT para modo “Release”.

Também neste arquivo é possível alterar a porta de execução da API e da plataforma Web. A API está configurada para executar na porta 60633, enquanto a plataforma Web está configurada para executar na porta 32772. Lembre-se que, caso seja alterado as portas de execução do backend, também deverá ser alterado o endereço no aplicativo.

Embora seja possível utilizar o banco de dados também em container (existe arquivo docker-compose-db.yml com banco de dados e SGBD para uso), recomenda-se que para implantação em produção, seja instalado o banco de dados diretamente no sistema, devido a maior facilidade para implantar tarefas de backup. Recomenda-se neste caso, o uso do banco de dados em container apenas para o ambiente de desenvolvimento.

Considerando que tanto a api, quanto o web estarão no mesmo servidor, após a instalação do Docker e a configuração do docker compose basta executar o comando a seguir no console dentro do diretório da aplicação.

|  |
| --- |
| docker-compose up --build |

O Docker irá baixar dependências e configurar tudo aquilo que foi previamente estabelecido para a execução do projeto.

O docker-compose.yml pode ser separado para que possa ser instalada a api em um servidor, enquanto o web poderá ficar em outro.

Por fim, as imagens do container podem ser disponibilizadas em um sistema de registro (como Docker Hub) para facilitar a implantação.

# **Testes**

## **Testes automatizados utilizando o Espresso**

O framework Espresso auxilia na criação de testes analisando o possível comportamento do usuário e analisando as interações entre as telas do aplicativo. Os casos de teste gerados pelo Espresso visam tanto o conceito de teste de caixa-preta, quando não se tem conhecimento do código em si, quanto o conceito de teste de caixa branca, já que ele é recomendado quando conhece-se as rotas entre telas e demais interações da aplicação.

Para o uso do Espresso é necessário adicionar nas dependências do *gradle* do aplicativo e sincronizar na opção que aparece logo após adição da

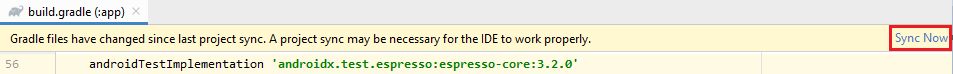


Figura 8.1

O modo de escrever casos de teste utilizando o Espresso foi dada de duas maneiras: escrevendo o código analisando suas chamadas ou gravando um teste utilizando a ferramenta Espresso Test Recorder. Esta ferramenta é acessada pela opção Run do Menu principal, como ilustra a imagem abaixo.

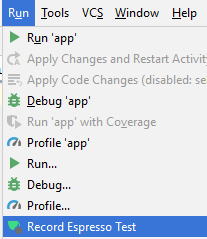


Figura 8.2

Após selecionar a opção Record Espresso Test, abre-se uma tela que irá listar o passo a passo feito no aplicativo através do dispositivo, podendo ele ser um dispositivo virtual ou o celular Android conectado ao cabo.Após a gravação dos possíveis passos feitos pelo usuário, adiciona-se uma assertiva para conferir o fechamento do caso de teste.

O detalhamento dos casos de teste está na documentação de teste (Especificação de Casos de Teste). Dentro da pasta abaixo, estão os 8 casos de testes utilizando ao Espresso:

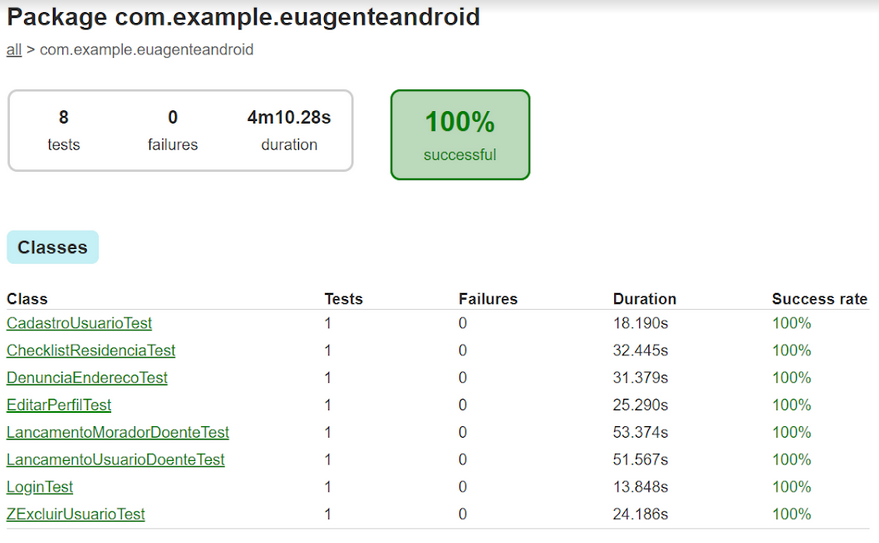


Figura 8.3

A ordem dos casos de teste é importante pois é seguido estritamente a ordem proposta acima e a manipulação de dados em determinado caso de teste pode afetar um caso de teste diferente posteriormente.E ao executar os casos de teste, é necessário a existência de algum dispositivo para reproduzir as interações entre as telas citadas no caso de teste.

Ao finalizar os testes, é possível ver quais testes passaram e os testes que não passaram, é exibido em qual linha do código que ele quebrou.

## **Testes Unitários**

Para a criação dos casos de testes unitários, é preciso adicionar as dependências abaixo (Figura 8.4) e sincronizar.



Figura 8.4

E adicionar no *defaultConfig* também do aplicativo, o código abaixo e sincronizar (Figura 8.5).



Figura 8.5

Os testes unitários foram realizados para testar algumas possíveis entradas do *validates*, encontrados em *model.util* no aplicativo.

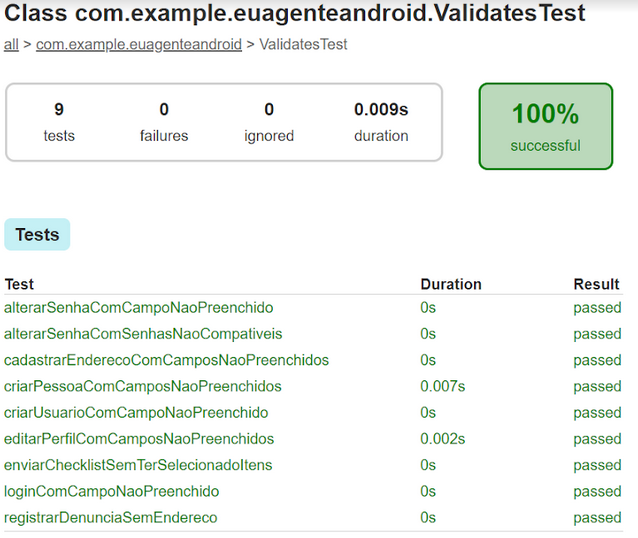


Figura 8.6

## **Cobertura de Teste**

Para analisar a cobertura dos casos de testes citados nos tópicos acima, foi usada a ferramenta JaCoCo (*Java Code Coverage*). Para usá-la, é necessário colocar o código (Figura 8.7) no *build* do projeto e sincronizar.



Figura 8.7

No *build* do aplicativo, coloca-se no buildTypes o código (Figura 8.8) para habilitar a cobertura e sincronizar.



Figura 8.8

A geração do relatório é dada através de duas tasks: jacocoTesteDebugUnitTestReport (teste unitário) e inserirNomeTak (teste com o Espresso).

No relatório do teste unitário, 100% dos casos de teste (Figura 8.6) passaram logo o relatório mostrou o mesmo (Figura 8.9)

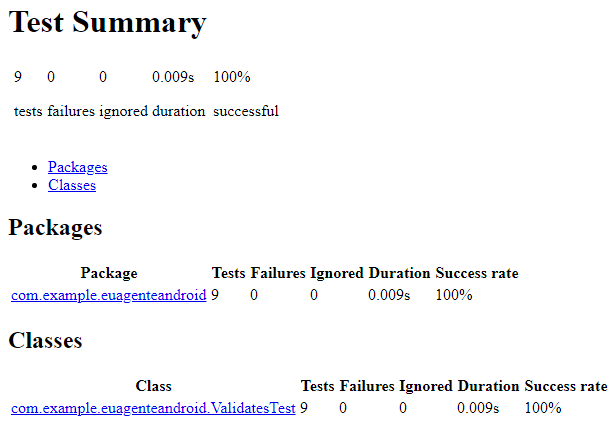


Figura 8.9

No relatório dos testes utilizando o Espresso, 100% do casos de teste passaram (Figura 8.3) e no relatório tivemos o resultado abaixo em relação à cobertura:

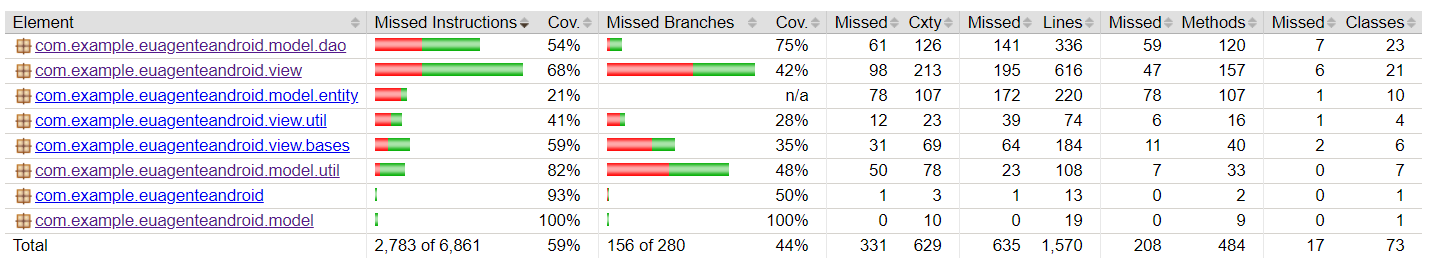


Figura 8.10

# **Análise Estática**

Para análise da estática do código foram utilizadas estratégias diferentes que serão detalhadas a seguir.

## **Visual Studio**

A IDE Visual Studio foi utilizada para análise estática de código. Através dela foi possível identificar classes e métodos que possuíam um alto índice de complexidade. Para isso, foi considerado o índice de manutenibilidade.

O Índice de Manutenibilidade foi introduzido em 1992 por Paul Oman e Jack Hagemeister. Apresentado originalmente na Conferência Internacional sobre Manutenção de Software ICSM 1992 e posteriormente refinado em um artigo publicado no IEEE Computer. É uma mistura de várias métricas, incluindo o volume de Halstead (HV), a complexidade ciclomática de McCabe (CC), as linhas de código (LOC) e a porcentagem de comentários (COM). Para essas métricas, a média por módulo é obtida e combinada em uma única fórmula:

O resultado é um valor entre 0 e 100, onde quanto maior melhor.

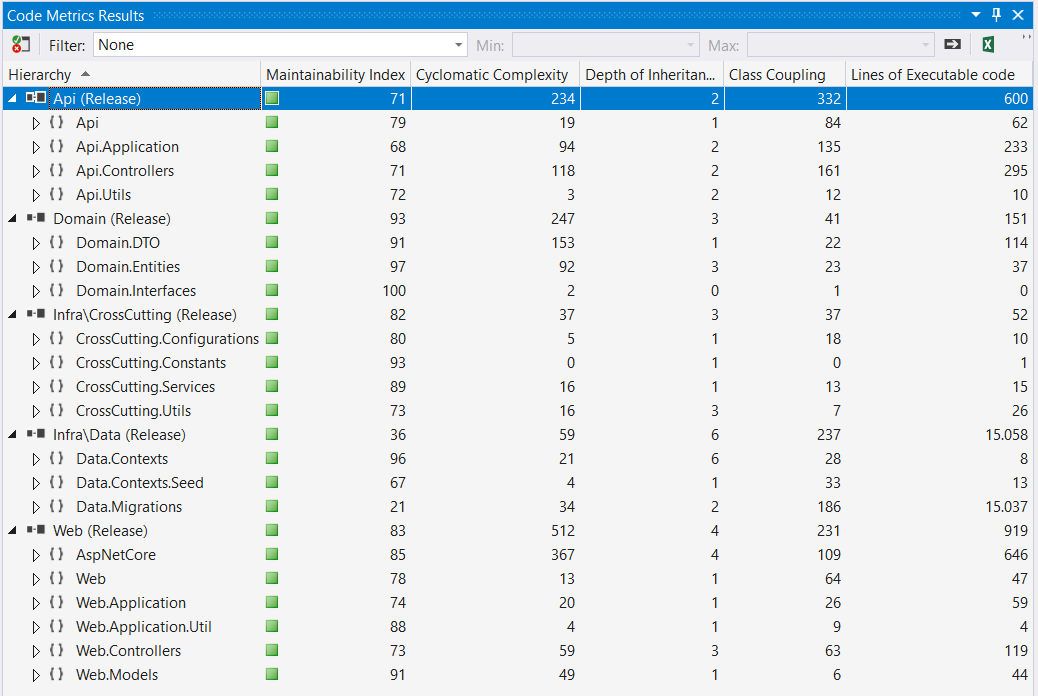
A Microsoft considerou que o Visual Studio deve fornecer a seguinte interpretação:

|  |  |  |
| --- | --- | --- |
|  | MI >= 20 | Manutenibilidade alta - indica que o código deve manutenível |
|  | 10 <= MI < 20 | Manutenibilidade moderada - indica que o código deve ser “semi-díficil” de manter |
|  | MI < 10 | Manutenibilidade baixa - indica que o código deve ser muito díficil de ser mantido |

Tabela 10.1 - Interpretação do Índice de Manutenibilidade

O Visual Studio gera um relatório interativo onde é possível verificar as métricas para cada pacote, cada classe e cada método.

O estado atual do sistema possui as seguintes métricas em nível de pacote.



## **SonarQube**

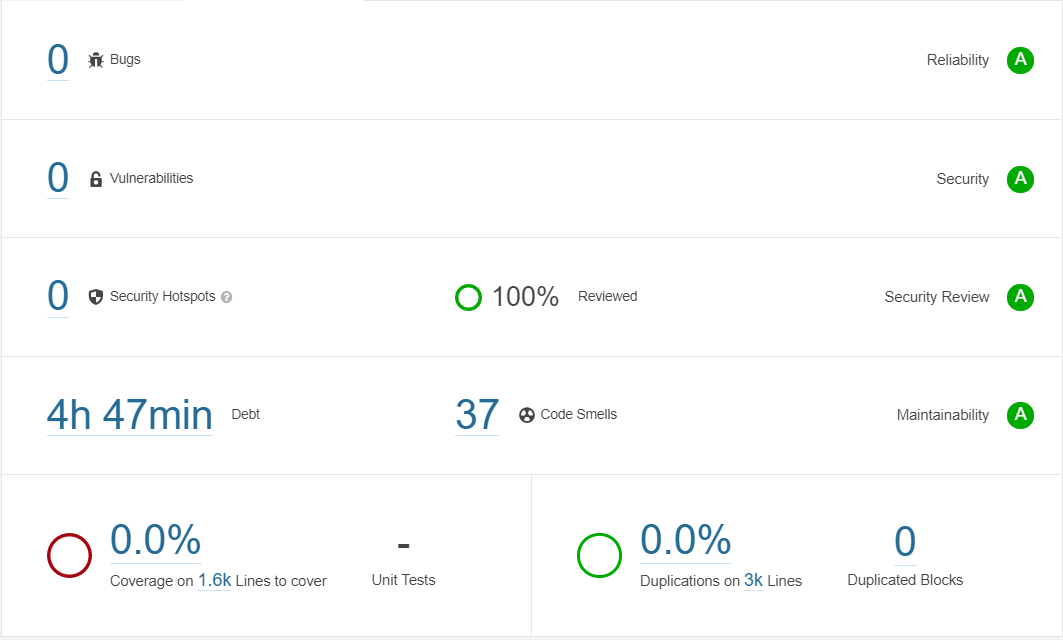
Foi utilizado o SonarQube para identificação de bugs, falhas de segurança, code smells e códigos duplicados.

O estado atual do sistema possui um débito de cerca de 5 horas, contendo apenas code smells pendentes, sendo que possuem criticidade de nível baixo. Os débitos técnicos que ficaram pendentes de correção referem-se apenas a algumas otimizações, como por exemplo, criar constante para strings que se repetem em alguns locais do código.

Já quanto os códigos apontados pelo SonarQube como duplicados, referem-se em sua totalidade a trechos de códigos em html de form’s de campos que estão tanto na parte de criação, quanto edição da entidade.

Por fim, não foi possível importar o relatório de cobertura de testes, já que o Sonarqube retirou a opção de carga de relatórios do tipo jacoco.

### **Painel Aplicativo**



### **Painel Backend**

