UNIVERSIDADE FEDERAL DE SANTA CATARINA

Ambiente Web integrado com App Inventor para execução de aplicações Android.

Gustavo Borges França

Florianópolis - Santa Catarina

2019/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Curso de Ciências da Computação

Ambiente Web integrado com App Inventor para execução de aplicações Android

Gustavo Borges França

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Florianópolis - Santa Catarina

2019/2

Gustavo Borges França

Ambiente web integrado com APP Inventor para execução de aplicações Android

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador(a) : Prof. Dr. Jean Carlo Rossa Hauck

Banca examinadora

_____

Prof. Leandro José Komosinski

_____

Fernando da Cruz Pinheiro, Ms.C

Resumo

Nos dias de hoje, com a intensa presença da computação no cotidiano de todos, o ensino de programação nos ambientes escolares se torna assunto em alta, pois para se manter competitivo no mercado, competências atreladas à computação se fazem necessárias. No intuito de formar nos alunos essas competências, várias iniciativas já foram criadas para integrar o ensino de computação ao ensino “tradicional”. Dessa forma, além de acabar atraindo jovens ao mercado da computação, esse ensino também tem papel no desenvolvimento de habilidades como resolução de problemas, lógica e programação em si. O *App Inventor* é uma ferramenta criada com o propósito de ser didática mas ao mesmo tempo poderosa o suficiente para criar aplicações para dispositivos Android. No entanto, existem dificuldades na hora de executar essas aplicações criadas com a ferramenta devido à complexidade de execução dos aplicativos durante o desenvolvimento, devido a problemas muito comuns no contexto de escolas brasileiras, a limitação de acesso dos alunos a celulares compatíveis e às restrições de rede são dois exemplos clássicos. Assim, o objetivo deste trabalho é então desenvolver um ambiente web que ofereça acesso remoto a uma máquina virtual com Android para que as aplicações criadas com a ferramenta *App Inventor* sejam executadas e testadas. Para a realização deste trabalho, são analisados conceitos relacionados com o tema abordado. Também é feita uma pesquisa sobre as ferramentas existentes hoje em dia que permitem o acesso por meio de um navegador à um emulador Android, comparando os resultados encontrados com os requisitos estabelecidos. Posteriormente, é desenvolvido o ambiente web proposto, e são feitos testes com o propósito de avaliar e validar as funcionalidades do sistema. Os resultados dos testes apontam que o ambiente desenvolvido consegue desempenhar o papel proposto, além de evidenciar que suas funcionalidades servem ao seu propósito.

Palavras-Chave: Android, *App Inventor,* Máquina Virtual, Ensino de programação.

Abstract

Nowadays, with the intense presence of technology in the everyday life, the teaching of programming in the school environments becomes a hot topic, because, to keep yourself competitive in the market, some competences related to Computer science and technology are necessary.There are a lot of initiatives that have the intent to integrate the teaching of computation to the “traditional” school. In this way, besides bringing new attracting new people to the Computer science field, those initiatives also have a role in helping the students develop skills like, problem solving, logic and programming. The MIT *App Inventor* is a tool created with the purpose of being both didactic, and powerful enough to create applications for Android devices. However, there are some difficulties when it comes to testing those applications, because there is a complexity in the execution of those applications during development, given the context that the brazilian schools live in (poor wireless connection for an example), and the limitations to the student’s access to phones that are compatible to the application. Therefore, the objective of this work is to develop a web environment that gives remote access to a virtual machine, running Android, that way the applications develop with *App Inventor* can be tested and executed. Through research in the concepts related to the approached subject, and comparison of the state of the art to the established, the proposed web environment is developed, and tested. In those tests, it is shown by the results that the developed software perform the proposed role, and its functionalities serves their purposes.

Keywords: Android, *App Inventor*, Virtual machine, Programming teaching.

Lista de imagens

Lista de Tabelas

1. Introdução

O ensino de programação favorece o desenvolvimento do raciocínio lógico, da capacidade de abstração, além de noções de causa e efeito (quando faço “isto”, acontece “aquilo”) e também capacidade de concentração e resolução de problemas (GOMES, 2015). Jovens atualmente estão cada vez mais conectados, cerca de 82% dos jovens de 9 a 17 anos de idade acessam a internet por dispositivos móveis (BARBOSA, 2015). Não somente quem deve aprender programação deve ser quem deseja se tornar um profissional de TI, mas sim qualquer pessoa que deseja melhorar as habilidades de lógica e abstração.

Na Educação Básica, a Computação, quando presente no ensino, tem sido apenas de forma instrumental, envolvendo assuntos como edição de textos, planilhas, e não o desenvolvimento do pensamento computacional, que é o que realmente desenvolve e possivelmente atrai jovens para a área, ou muitas vezes trata a disciplina como um diferencial mercadológico (BARBOSA, 2015). O Pensamento Computacional trata da habilidade de solucionar problemas por meio de conceitos da Ciência da Computação (SEBRAE, 2019).

Algumas iniciativas como Hour of Code[1] e Computação na Escola[2] levam o ensino de computação para crianças e jovens da Educação Básica. O ensino de programação nessa faixa etária tem sido tipicamente realizado utilizando ferramentas como *App Inventor*[3] e Scratch[4].

O *App Inventor* é uma aplicação de código aberto criada inicialmente pela Google e atualmente mantida pelo MIT (MIT, 2019). É uma aplicação que permite a criação de aplicações para o sistema operacional Android por meio de uma interface visual, pela qual é possível programar utilizando blocos e objetos visuais que formam a lógica de uma aplicação. É uma ferramenta voltada para pessoas que estão começando na programação como crianças e adolescentes, ou seja, serve muito bem como uma plataforma para aprendizado.

Uma das problemáticas que envolvem o uso do *App Inventor* como uma ferramenta de ensino em escolas brasileiras, é a execução e o teste dos aplicativos desenvolvidos na ferramenta, uma vez que um aluno deve ter um celular Android, e a escola ou instituição que está oferecendo as aulas deverá ter uma rede wi-fi disponível. Segundo a revista Época, em 2015 o uso de wi-fi em escolas públicas cresceu de 84% a 91%, o que pode parecer um

[1] Hour of Code. https://hourofcode.com/pt/pt.
[2] Computação na Escola. http://www.computacaonaescola.ufsc.br/.
[3] *App Inventor*. http://appinventor.mit.edu/explore
[4] Scratch. https://scratch.mit.edu/

excelente número a primeira vista perde força pelo fato de 45% das conexões dessas escolas não passam de 4 Mbps (Época, 2015), o que gera muitos problemas quando a escola inteira está usando a rede. Além disso, para utilizar o próprio celular é necessário a instalação e configuração do aplicativo MIT AI2 Companion[5], o que gera um aumento na complexidade, que pode levar ao desinteresse.

Uma solução possível seria o uso de emuladores de Android instalados nos computadores onde se ensina programação com *App inventor*, para permitir a execução dos aplicativos. No entanto, emuladores são muito exigentes em relação ao hardware que uma máquina deve ter, o emulador BlueStacks[6] por exemplo, nos seus requisitos recomendados pede 6GB de memória RAM, e um processador Intel Core i5-680, o que é uma exigência razoavelmente alta de hardware, além de existir uma problemática sobre configurar o emulador corretamente, o que pode limitar sua aplicação em ambiente escolar.

Assim, o presente trabalho apresenta uma solução que consiste no desenvolvimento de um ambiente *web* que permite o acesso remoto a uma instância de uma máquina virtual Android para execução das aplicações criadas na ferramenta *App Inventor*. Através dessa ferramenta, será possível a execução e teste dos aplicativos desenvolvidos com o *App Inventor*, sem a necessidade de realizar algum tipo de configuração, ou o uso de qualquer dispositivo externo ou rede sem fio.

1.1 Objetivo

1.1.1 Objetivo Geral

Desenvolver um ambiente *web* que permita a execução remota de aplicações desenvolvidas com o *App Inventor*. Esse ambiente deve ter acesso remoto a uma máquina virtual com uma imagem Android rodando em um servidor, e por meio desse servidor permitir a execução remota das aplicações desenvolvidas com o auxílio da ferramenta *App Inventor*.

1.1.2 Objetivos Específicos

- Levantar fundamentação teórica sobre acesso remoto a dispositivos Android, máquinas virtuais em servidores com requisitos de hardware e acesso remoto a tais servidores, além de desenvolvimento de aplicativos com *App Inventor* e compatibilidade com versões de Android.

---

[5] Conectar celular ou tablet via Wi-Fi ao MIT App Inventor. http://appinventor.mit.edu/explore/ai2/setup-device-wifi.html.

[6] BlueStacks. https://www.bluestacks.com/

- Análise do estado da arte de ambientes que permitem acesso remoto a Android ou outros sistemas operacionais de dispositivos móveis.
- Modelar solução computacional para desenvolvimento do ambiente remoto de execução.
- Desenvolver aplicação web que permita a execução de aplicações Android criadas pelo App Inventor
- Testar o ambiente desenvolvido, executando aplicações desenvolvidas com o *App Inventor*
- Avaliar o ambiente com base nos testes realizados com as aplicações criadas e executadas no mesmo.

1.2 Metodologia de Pesquisa

Etapa 1 - Fundamentação teórica: Analisar referências de assuntos que são relacionados ao desenvolvimento da solução apresentada no trabalho.

A1 - Análise teórica sobre acesso remoto a dispositivos Android, servidores e VMs

A2 - Análise sobre desenvolvimento de aplicações no *App Inventor*.

A3 - Análise teórica sobre aplicações que acessam Android na nuvem.

Etapa 2 - Levantamento do estado da arte: Estudar e analisar aplicações que já tenham sido desenvolvidas que estão na mesma linha da aplicação proposta por este presente trabalho.

A1 - Definir protocolo de Busca.

A2 - Busca por ambientes web que controlam dispositivos Android na nuvem.

A3 - Analisar resultados e extrair o máximo de informações possível para utilizar no desenvolvimento do trabalho

Etapa 3 - Desenvolvimento da Aplicação: Desenvolver a aplicação web que ofereça um ambiente Android para execução de aplicações desenvolvidas com auxílio da ferramenta *App Inventor*.

A1 - Descobrir a melhor forma de se ter uma Imagem Android em um servidor.

A2 - Criar um ambiente web que tenha acesso remoto a esse servidor.

A3 - Permitir a execução de aplicações criadas com *App Inventor*.

A4 - Testar.

Etapa 4 - Avaliação e Prova de Conceito: Testar a corretude do ambiente desenvolvido criando algumas aplicações e executando-as.

A2 - Executar uma aplicação real, desenvolvida com *App Inventor* no ambiente desenvolvido.

1.3 Estrutura do trabalho

No capítulo 2 é apresentada a fundamentação teórica acerca dos conceitos necessários para o desenvolvimento da solução proposta. Durante capítulo 3 é feito um estudo do estado da arte de emuladores Android pela web (*cloud*). No capítulo 4 é feita uma proposta para a solução, apresentando os casos de uso, requisitos e protótipos de tela. O capítulo 5 são feitos, o desenvolvimento, baseando-se na modelagem e nos requisitos levantados no capítulo 4, e os testes, para validar a solução desenvolvida. Por fim, no capítulo 6 é apresentada a conclusão do trabalho, e propostas para trabalhos futuros.

2. Fundamentação Teórica

Neste capítulo são apresentados conceitos sobre o ensino de computação/programação na educação básica. São abordadas algumas questões sobre o *App Inventor.* E por fim são apresentados também tópicos sobre emuladores em geral e emuladores específicos para Android.

2.1 Ensino de programação na educação básica.

Atualmente, no Brasil, o ensino de computação, de forma geral, tem sido abordado somente em cursos de ensino superior. Para o Ensino Fundamental II (6º ao 9º ano), nos Parâmetros Curriculares Nacionais, não consta conteúdo explícito sobre computação, mas somente diluído na área de matemática (PCN, 2019). Além disso, ainda não existe um currículo nacional oficial no Brasil para o ensino de programação na educação básica (BNCC, 2019).

Apesar disso, já existem diversas outras bases e padrões curriculares de referência ao redor do mundo, que incorporam o ensino da computação à educação básica, um dos mais reconhecidos é o CSTA K-12 (CSTA, 2017). Esse *Framework* apresenta padrões e currículos para a implementação do ensino de ciências da computação nos ensinos básico e médio (CSTA, 2017). Através desses padrões esse *framework* promove uma visão na qual todos os estudantes se empenham em problemas de ciências da computação, como tratar de problemas de uma maneira inovadora, e além disso, criar artefatos computacionais com intuito prático, pessoal e social (CSTA, 2017).

Esse *framework* representa uma visão na qual todos os estudantes se engajam nos conceitos e práticas da ciência da computação. Iniciando no começo do ensino fundamental e continuando até o ensino médio, os estudantes desenvolver a base do conhecimento de ciências da computação, e aprendem novas estratégias para resolução de problemas que utilizam o pensamento computacional, tornando-se tanto usuários quanto criadores de tecnologia.

As diretrizes CSTA K-12 se baseiam em um modelo de 3 níveis para o ensino de computação nos ensino fundamental e médio. O nível 1 define padrões de aprendizagem para estudantes do ensino básico até o quinto ano do ensino fundamental. O nível 2 define padrões de aprendizagem para alunos do sexto até o nono ano do ensino fundamental. O nível 3 define padrões de aprendizagem para alunos do primeiro ao terceiro ano do ensino médio (CSTA, 2017). A estrutura desses níveis pode ser visualizada na figura 1.

Figura 1. Níveis do currículo CSTA K-12 (CSTA, 2017)

O foco do presente trabalho é nos níveis 1 e 2, pois a aplicação a ser desenvolvida será aplicada principalmente no ensino fundamental.

O CSTA K-12 define alguns conceitos fundamentais que podem ser vistos como objetivos de aprendizagem, pois são áreas-chave da ciência da computação, e também define práticas fundamentais, que são as “estratégias” utilizadas para atrair e incentivar o aprendizado desses conceitos fundamentais pelos alunos. A tabela 1 mostra os conceitos e as práticas fundamentais.

Table 1:

Conceitos Fundamentais
Sistemas de computação; Redes e internet; Dados e análise; Algoritmos e programação; Impactos da computação;
Práticas fundamentais
Fomentar uma cultura de computação inclusiva; Criação de artefatos computacional; Colaboração acerca da computação; Testar e refinar artefatos computacionais; Reconhecendo e definindo problemas computacionais; Comunicando sobre computação; Desenvolver e usar abstrações;

Tabela 1. Conceitos e práticas fundamentais do *framework* (CSTA, 2017)

Alguns temas significantes estão incluídos implicitamente no *framework* como:

- **Equidade:** Problemas de equidade, inclusão, e diversidade são tratados nos conceitos e práticas do *framework,* em recomendações para padrões e

currículo, e em exemplos de esforços para ampliar a participação na educação em ciências da computação.

- **Ideias poderosas:** Os conceitos e práticas despertam ideias autênticas e poderosas, estas podem ser utilizadas para resolver problemas do mundo real, e conectar o conhecimento de múltiplas disciplinas.
- **Pensamento computacional:** Pensamento computacional estimula a prática habilidades como abstração, modelagem e decomposição, realizando uma intersecção com conceitos da computação como algoritmos, visualização de dados e automação.
- **Variedade de aplicação:** Ciência da computação é mais que programar. Envolve sistemas físicos e redes. A coleta, armazenamento e análise de dados, e o impacto da computação na sociedade. Essa visão ampla da ciência da computação dá ênfase à variedade de aplicações que a computação tem em outros campos.

O presente trabalho trata justamente da criação de uma ferramenta que irá auxiliar no ensino da programação nas salas de aula. Com o uso do *App Inventor* os estudantes da Educação Básica irão desenvolver aplicações para o sistema operacional Android, e utilizando do software desenvolvido neste projeto, será possível testar essas aplicações desenvolvidas de forma ágil, possivelmente acelerando e facilitando o processo de aprendizagem.

2.2 *App Inventor*

O MIT *App inventor* é uma ferramenta de programação visual, de código aberto, desenvolvido com a linguagem de programação Java, criada pela Google, e hoje em dia é mantido pelo *Massachusetts Institute of Technology* (MIT). Essa ferramenta serve para desenvolver aplicações para o sistema operacional Android através de um ambiente de programação em blocos. Seu diferencial consiste em possibilitar aos usuários criar aplicações que incorporem serviços baseados na web, interação com redes sociais, leitura de códigos de barra, interação com sensores de orientação e geolocalização, e de funcionalidades como text-to-speech e reconhecimento de fala (MIT, 2019).

Ela é muito útil para ensinar recém-chegados ao mundo da programação a criar aplicações para Android, e aprender sobre programação em geral. Seu ambiente possibilita o ensino de conceitos de lógica de programação de uma forma atraente e motivadora para estudantes dos ensinos básico e médio (GOMES e MELO *et al.,* 2013).

O *App Inventor* utiliza um formato específico de arquivo .aia que serve para importar ou exportar projetos, e este arquivo guarda todas as informações do projeto, componentes, eventos e blocos. Além disso, no final do desenvolvimento da aplicação é possível gerar um .apk (extensão de aplicações para Android), juntamente com um QR Code para escanear a aplicação e executar no celular, ou baixá-la diretamente no computador. Existe também a possibilidade de conectar o celular com a aplicação através da rede, utilizando a aplicação chamada MIT *Companion*. Desta forma, o teste da aplicação desenvolvida se dá em tempo real.

O ambiente de desenvolvimento de apps do *App inventor* é baseado em componentes, que são formados por métodos, propriedades e eventos. A programação é orientada a eventos, ou seja, o que define o comportamento dos componentes, na maioria, são os eventos gerados por interação do usuário com a aplicação (alguns paralelos com programação mais convencional são bem notáveis aqui, ex: componentes com classes, métodos com métodos das classes/funções, etc.).

Existem duas janelas para o desenvolvimento de uma aplicação no ambiente de desenvolvimento do *App Inventor*, a janela *design* e a janela *blocks.* A janela *design* apresenta o *palette* que disponibiliza componentes como, botões, *labels,* e efeito sonoros, para serem inseridos no *front-end* da aplicação, e esses componentes são mostrados visualmente na parte *viewer*, e também são mostrados de forma hierárquica na parte *components*, que mostra todos os componentes que estão na tela atual. O outro bloco na parte do *design* é o *properties*, que serve para alterar propriedades como, altura, largura, cor, de um componente.

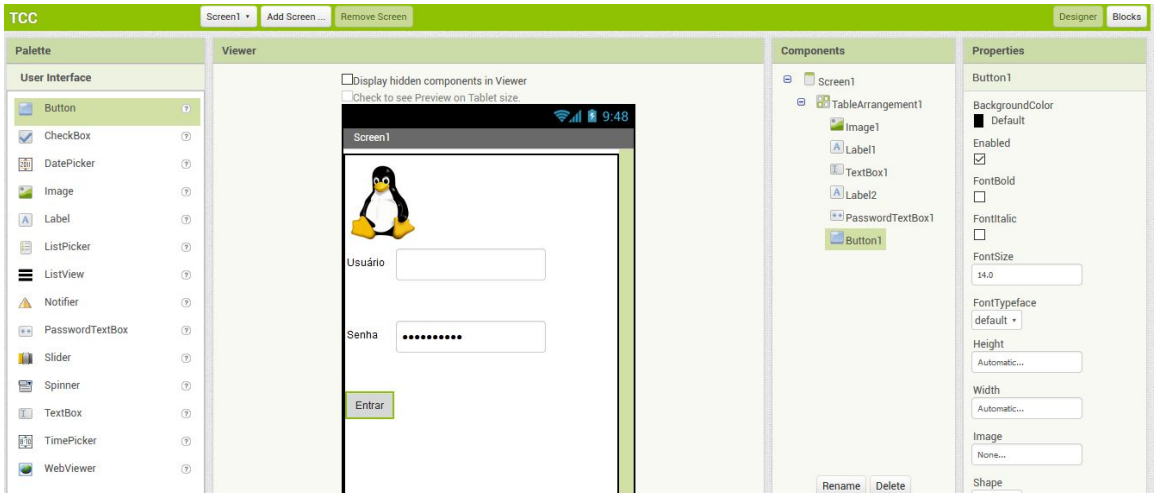

Figura. 2 Tela *design* do *App Inventor*.(AUTOR, 2019)

A tela *blocks* do *App Inventor* permite realizar o controle dos componentes inseridos na aplicação através da janela de *design.* Nessa tela existem vários blocos que podem ser arrastados e conectados para criar a lógica do programa, esses blocos são eventos ou métodos. Esses blocos podem realizar operações de controle, como nas linguagens de programação convencionais (*if, else, while,* etc.), além de operações aritméticas e outras funcionalidades.

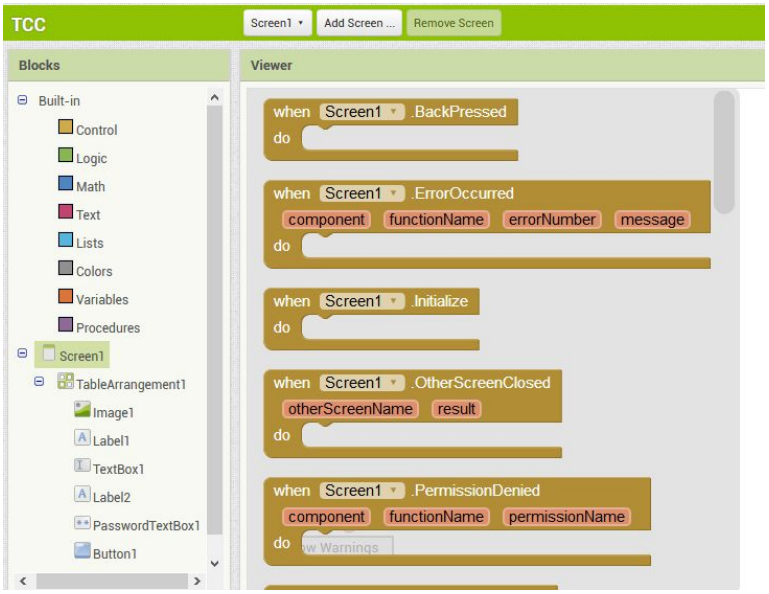

Figura. 3 Tela *blocks* do *App Inventor*.(AUTOR, 2019)

No site oficial do *App Inventor* é possível encontrar diversos materiais de ensino e tutoriais, eles mostram desde como criar seu primeiro aplicativo até como criar aplicações mais elaboradas, com o "código fonte" e etc.

2.2.1 MIT AI2 Companion

Dentro do ambiente do *App Inventor* existem algumas formas de realizar o *live testing* de uma aplicação, uma delas é por meio do aplicativo MIT AI2 *Companion*[7] que, em tempo real, através de um aparelho Android permite o *debugging* da aplicação desenvolvida.

Esse retorno sobre o estado de uma aplicação é importante, pois ajuda os estudantes a avaliar o trabalho que estão realizando e refletir sobre os resultados que estão obtendo, seus objetivos, e como alcançá-los (CHATZINIKOLAKIS; PAPADAKIS, 2014). Ainda com o uso do *live testing* é possível verificar as alterações feitas em interfaces por exemplo, sem a necessidade de reinstalação da aplicação criada no dispositivo (WOLBER, 2011).

[7] http://appinventor.mit.edu/explore/ai2/setup-device-wifi

Para realizar o *debugging* utilizando o MIT *Companion*, o usuário deve ter um aparelho Android com o MIT AI2 *Companion* instalado, esse aparelho deve estar na mesma rede wi-fi que o computador onde a aplicação está sendo desenvolvida. No ambiente do *App Inventor*, dentro do menu *connect* é possível ver a opção “AI Companion”, ao clicar nessa opção, um pequeno *pop-up* se abre mostrando um QR Code e um código de caracteres, tal como a figura 4 mostra.

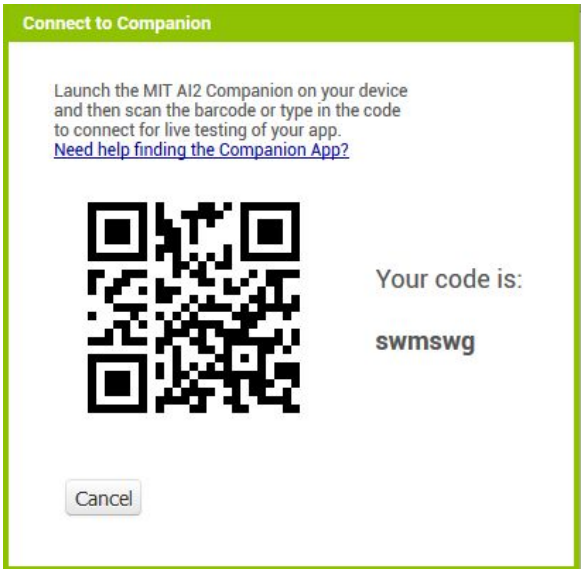

Figura. 4 Conectar *App Inventor* com MIT AI2 Companion. (AUTOR*,* 2019*)*

Como é possível ver na figura 4 existe um *QR Code* e um código que é apenas uma sequência de caracteres. O usuário com o dispositivo móvel Android (celular, tablet) pode utilizar qualquer dos dois códigos para se conectar ao *App Inventor* utilizando o MIT *Companion* previamente instalado.

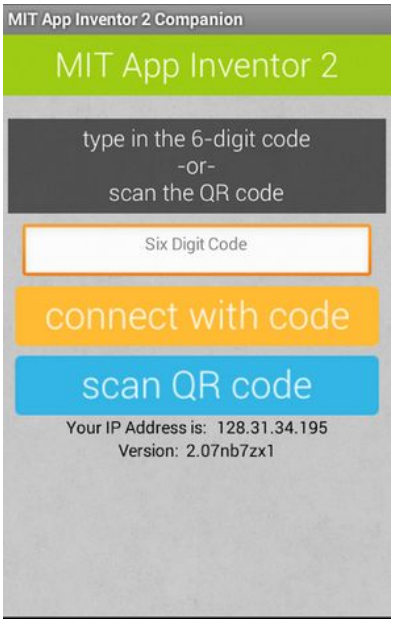

Figura. 5 Tela inicial MIT AI2 Companion (AUTOR, 2019).

Desde a *release* nb172 o MIT *Companion* utiliza o protocolo WebRTC para realizar a comunicação entre o navegador (*App Inventor*) e o dispositivo (MIT *Companion*).

2.2.2 WebRTC

WebRTC[8] é uma API de código aberto que tem como objetivo levar comunicação em tempo real para navegadores e dispositivos móveis. Ela permite a execução de aplicações de chamada telefônica, vídeo-chamada, e compartilhamento p2p sem necessidade de *plugins.*

Essa API consiste em diversas APIs interligadas e protocolos, estes trabalham conjuntamente para realizar todas as funcionalidades do WebRTC sem necessidade de quaisquer softwares de terceiros. A figura 6 traz uma visão geral de como é a arquitetura dessa API.

[8] https://webrtc.org/architecture/

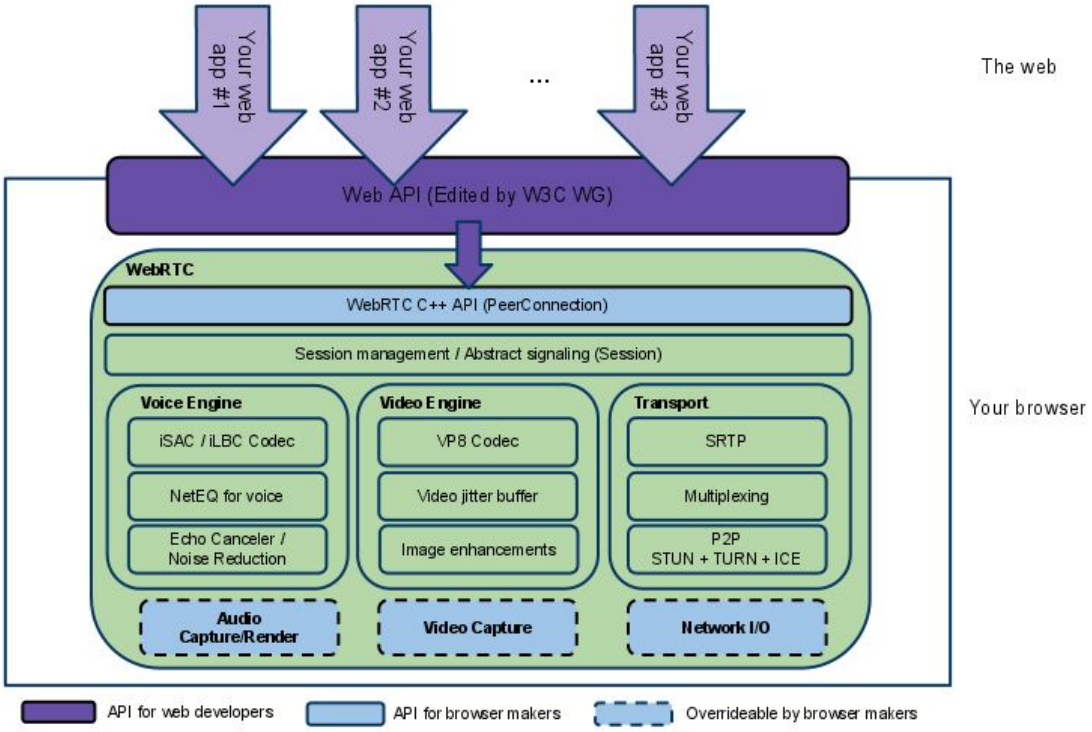

Figura. 6 Arquitetura WebRTC. (WebRTC[9], 2019)

A Web API é a parte utilizada pelos desenvolvedores web para criação de aplicações que contenham funcionalidades como troca de arquivos, conferências em áudio e vídeo, aplicações com chat de vídeo.

Já a outra camada, que seria a API C++, é voltada para os desenvolvedores de navegadores, pois com essa API será definida a implementação da API web para o navegador. As outras partes são componentes que fazem parte da API como um todo, como codecs de áudio e vídeo, implementação de protocolos de rede, entre outras coisas.

2.2.3 Outras formas de *live testing* com o *App Inventor*

Além do *debugging* via *wi-fi*, o *App Inventor* disponibiliza outras duas formas de realizar o *live testing* das aplicações desenvolvidas:

1) Por meio de um dispositivo Android conectado ao computador através de um cabo USB
2) Por meio de um emulador

O *live testing* rodando por via do cabo USB pode ser um pouco problemático, pois além de ainda ser necessário um dispositivo Android com o MIT AI2 Companion instalado, se o sistema operacional do computador for windows, esse método envolve instalação de

[9] https://webrtc.org/architecture/

drivers, e se for linux, envolve a inserção de comandos no terminal. E depois de tudo isso, ainda é necessário colocar o dispositivo Android em modo depuração via USB (*USB Debugging*) (MIT AI2, 2019).

Já o *debugging* por meio do emulador disponibilizado pelo *App Inventor,* é um software que deve ser instalado na máquina, que simula a execução dos aplicativos como se fosse um dispositivo Android. O problema é que o emulador possui algumas limitações, como não ser possível testar funções da categoria “sensores” ou algumas das categorias “social” e “multimídia”. Além disso o emulador também não permite simular alguns comportamentos que existem em um dispositivo real, como, determinar localização via GPS, realizar chamadas telefônicas, ou o envio de SMS.

Alternativa	Requisitos
Android com rede sem fio.	● Dispositivo Android com MIT AI2 Companion instalado; ● Rede local, com comunicação sem fio, e portas 8001/9987 ● abertas; ● Computador;
Android com USB.	● Computador; ● Dispositivo Android com MIT AI2 Companion instalado; ● Cabo USB; ● Conexão à rede sem fio;
Emulador.	● Computador com o aiStarter instalado (emulador disponibilizado pelo *App Inventor*); ● Conexão à rede sem fio;

Tabela 2. Comparativos das opções de *live testing* do *App Inventor*.

A tabela 2 mostra todas as opções de *live debugging* que o *App Inventor* disponibiliza, e também os requisitos para cada um desses métodos.

2.3 Emuladores

Um emulador é um software que possibilita que um computador, chamado de *host* ou hospedeiro, se comporte como outro sistema, chamado de *guest* ou hóspede (BARRIO, 2001). Em outras palavras um emulador tipicamente possibilita que o sistema hospedeiro rode programas que foram feitos para o sistema hóspede. Além da possibilidade de adicionar mais funcionalidades que o hardware original não possuía, em emuladores de video-games,

por exemplo, existe também a possibilidade de melhorar os gráficos e tudo isso mantendo sempre a aparência, e o comportamento original do sistema emulado.

A ideia básica por trás dos emuladores é de tratar o comportamento do processador e dos componentes individuais, criando cada uma dessas partes individuais do sistema e depois juntá-las e formar o sistema emulado como um todo. A figura 9 mostra uma visão de alto nível das funções básicas de um emulador.

```
while (emulando) {
        emulaCPU(Ciclos_CPU);
        geraInterrupções();
        emulaGraficos();
        emulaSom();
        emulaOutrosHardwares();
        emulaSoftwares();
        sincronizaTempo();
}
```

Figura. 7 Código em linhas gerais para um emulador (BARRIO, 2001)

Na sequência, as funções de um emulador mais relevantes para o contexto deste trabalho são apresentadas em mais detalhes.

2.3.1 Emulando o processador.

Na emulação do processador existem três formas de realizar esta tarefa:

- Interpretação
- Recompilação dinâmica
- Recompilação estática

Essas três formas levam a um resultado/objetivo em comum, executar um pedaço de código para modificar o estado do processador e interagir com o hardware (O estado do processador se refere aos valores dos registradores, tratadores de interrupção, etc.).

Na interpretação, é dado início no registrador IP*(Instruction pointer,* ou PC, *program counter* em alguns casos). Esse registrador está presente em toda arquitetura de processador, ele é o registrador que aponta para o endereço da instrução que o processador está executando. A instrução é lida então da memória e o código faz um *parsing* desta

instrução (decodifica), e a utiliza para alterar o estado do seu processador. O problema dessa abordagem é que é muito lenta, toda vez que uma instrução é tratada, é necessário decodificá-la e realizar a operação requisitada.

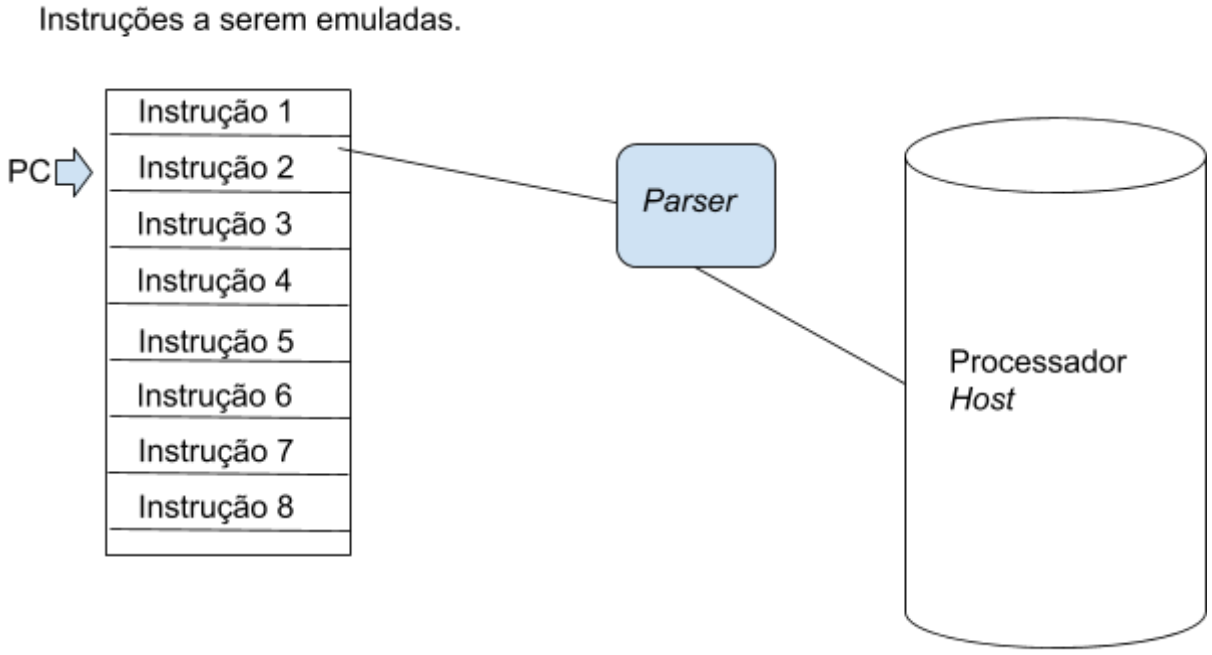

Figura 8. Emulação do processador por interpretação (BARRIO, 2001).

Na recompilação dinâmica, você itera sobre as instruções, como na emulação por interpretação, só que invés de ficar executando a cada instrução lida, constrói-se uma lista de operações. Quando uma instrução do tipo *branch* (instrução de desvio) é alcançada, essa lista de operações é então compilada na máquina *host*, colocada na *cache* e executada, dessa forma, caso esse grupo de instruções volte a ser executado, ele já está na *cache,* o que torna a execução mais rápida.

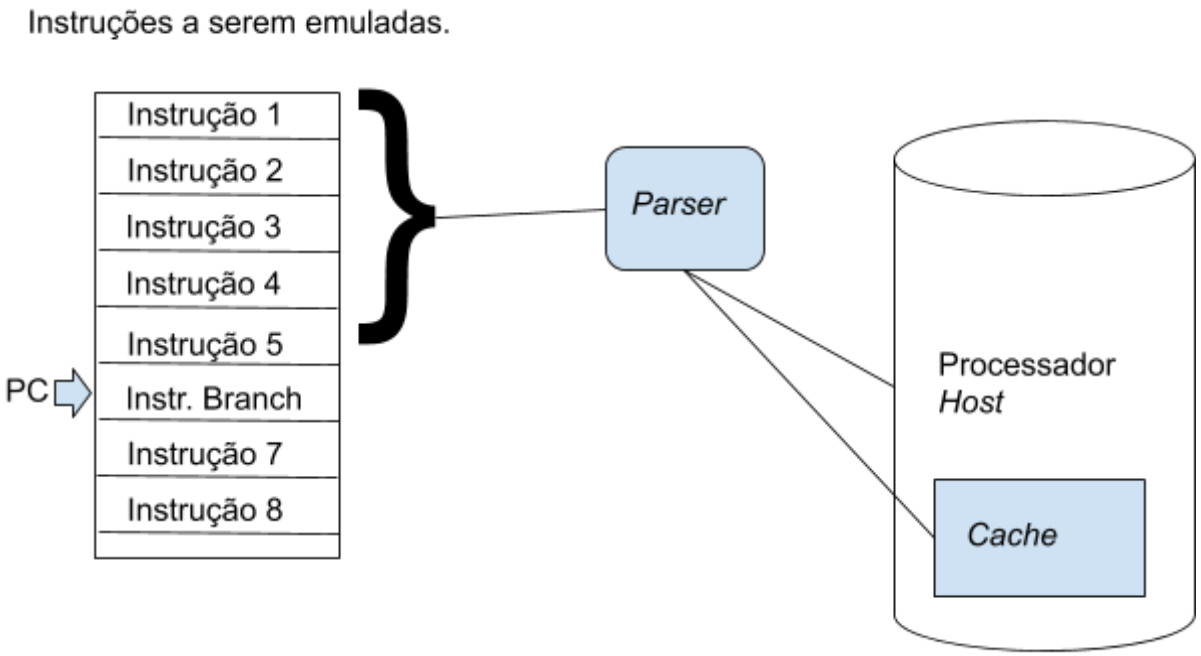

Figura. 9 Emulação do processador por recompilação dinâmica (BARRIO, 2001).

Na recompilação estática, ao invés de compilar conjuntos de instruções de cada vez, o código inteiro é recompilado, dessa forma as instruções serão executadas sem nenhuma interferência. Esse método seria o mecanismo perfeito se não fossem os seguintes problemas:

- Códigos que não estão no programa original (cifrados, gerado em tempo de execução, etc.) não serão executados de forma alguma.
- Já foi provado que achar todo o código de um programa dado o seu binário é equivalente ao problema da parada (*halting problem)*.

Esses dois problemas fazem com que a recompilação estática seja impraticável na maior parte dos casos.

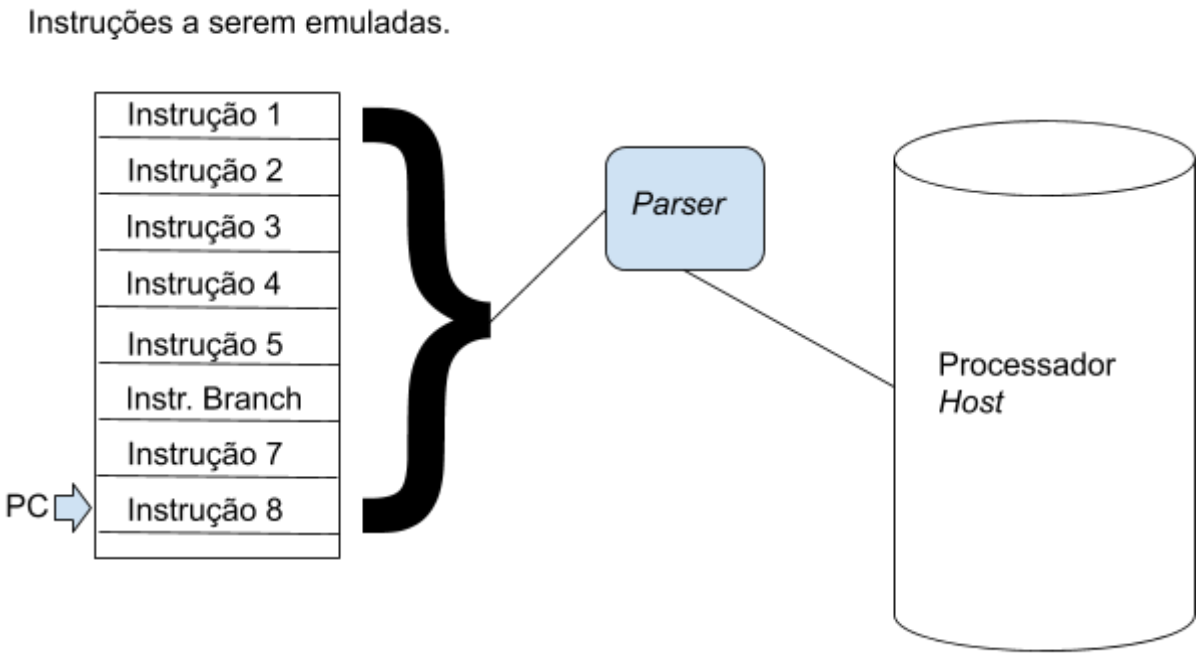

Figura. 10 Emulação do processador via recompilação estática (BARRIO,2001).

2.3.2 Emulando *Hardware*.

Para completar o emulador, seus outros componentes devem ser emulados juntamente com a emulação do processador (Este que precisa de todo um código a parte). Os outros componentes de hardware que podem ser emulados variam de acordo com o que se deseja emular, mas aqui podem ser incluídos, memória, *hardware* de gráficos, *hardware* de som, e assim vai.

Dado um dispositivo de *hardware* que se deseja emular, existem duas tarefas a serem cumpridas:

- Emular o dispositivo em si (suas funcionalidades).
- Emular suas interfaces.

Tomando o caso de emulação de um *hardware* de gráficos, deve-se emular desse *hardware* a forma em que ele realiza cálculos de bits, pixels, essas seriam suas funcionalidades. Um *hardware* desse tipo geralmente deve se comunicar com a CPU, que irá passar informações do que deve ser calculado, e com alguma tela, que irá passar o resultado final desses cálculos para o usuário, essas interfaces então, devem ser emuladas da exata forma em qual elas ocorrem, e devem ser feitas de uma maneira com rápida execução, uma

vez que esse tipo de *hardware* é utilizado com uma frequência muito alta, principalmente em emuladores de consoles de videogame antigos (BARRIO, 2001).

3. Estado da Arte

Nessa seção são apresentadas soluções que existem atualmente que disponibilizam ambientes web que por sua vez permitem acessos a emuladores Android, ou seja, emuladores Android acessados através do navegador, ou emuladores Android na nuvem e é feita uma análise dessas soluções conforme requisitos definidos para o problema levantado neste trabalho. Cada solução encontrada é apresentada, com o objetivo de encontrar a solução que mais se adequaria para a solução do problema apresentado. Neste trabalho, adotou-se o método de mapeamento sistemático da literatura proposto por Kitchenham (2007), adaptado para o contexto de um trabalho de conclusão de curso de graduação.

3.1. Definição do Mapeamento

Nesta etapa é estabelecida a pergunta de pesquisa a qual vai ser respondida como parte do mapeamento da literatura, além disso também, serão definidos os repositórios de dados no qual a buscas vai ser executada, os critérios de inclusão e exclusão além da qualidade, e as *strings* de busca executadas.

Todo esse processo de mapeamento sistemático visa responder a pergunta:

- Quais emuladores Android com acesso web existem atualmente?.

Para este mapeamento sistemático foi definido o Google[10] como ferramenta para realizar a busca de soluções que respondam a pergunta estabelecida, pois esta é uma ferramenta de busca que possibilita a busca abrangente por sites de empresas que implementam esse tipo de solução, além de soluções criadas por pessoas independentes de empresas.

Partindo da pergunta definida, agora são estabelecidos os termos de busca com sinônimos em inglês, estes irão ser utilizados na composição das *strings* de busca. A seguinte tabela mostra os termos de busca com seus sinônimos em inglês e as traduções.

TERMOS	**SINÔNIMOS**	**INGLÊS**
Emulador	-	*Emulator*

[10] http://www.google.com.br

Android	-	-
Web	Navegador	*Browser based, Web based*
Nuvem	-	*Cloud*

Tabela 3. Termos utilizados na busca.

Critérios de Inclusão/Exclusão. Os critérios de inclusão/exclusão são os seguintes:

- As soluções devem ser emuladores Android.
- Para poder executar o emulador web, não deve ser necessário downloads de softwares auxiliares ou de terceiros.
- Emuladores que necessitem de configurações extras que possam adicionar complexidade são descartados.
- Deve ser possível executar/realizar upload de aplicações no emulador.

Para responder a pergunta estabelecida anteriormente foi então elaborada uma *string* de busca a partir dos termos de busca supracitados. A tabela a seguir mostra a *string* de busca.

***String* de busca**	“Android” **AND** (“Browser based” **OR** “Web based” **OR “**Cloud**”**) **AND** “Emulator”

Tabela 4. *String* de busca utilizada.

3.2 Execução da Busca

A busca foi executada em março de 2019. A tabela seguinte mostra os resultados obtidos pela busca, os resultados potencialmente relevantes e os realmente relevantes. A busca por resultados potencialmente relevantes se limitou às vinte primeiras páginas do Google. Cada página apresenta dez links, ou seja, um total de 200 resultados.

Total resultados	Soluções potencialmente relevantes	Soluções realmente relevantes
200	7	4

Tabela 5. Relação dos resultados obtidos.

Desses sete resultados, apenas quatro atendem aos critérios de inclusão, e são eles: APK Online[11], RunThatApp[12], Appetize IO[13], Genymotion[14].

Os três resultados que foram considerados não relevantes são o MobilePhoneEmulator[15], ARchon[16] e ManyMo[17]. O problema do MobilePhoneEmulator é que ele permite apenas o teste de páginas web em um ambiente mobile simulado, ou seja não é bem um emulador de Android, mas sim um tipo de visualizador de páginas em dispositivos mobile, logo ele não se encaixa nos critérios de inclusão. Já o ARchon o problema é que ele na verdade é uma extensão de navegador, ele permite executar aplicações Android no Chrome, porém é necessário instalá-lo por se tratar de uma extensão. E o ManyMo o problema é um pouco mais simples, aparentemente o projeto foi descontinuado, porém pelas informações encontradas, caso ele ainda estivesse *on-line* ele se encaixaria nos critérios de inclusão e exclusão muito bem.

3.3 Extração e análise dos dados

Essa seção apresentará então as quatro soluções que foram encontradas com a execução da busca da seção anterior, extraindo e analisando dados sobre diversas características desses programas.

3.3.1 APK Online

É uma solução que disponibiliza uma máquina virtual e dentro dessa máquina virtual existe um *Android Virtual Device* rodando (AVD) este então permite a execução de aplicações Android pois é um ambiente simulado/emulador de Androids. É um software gratuito, e além

[11] https://www.apkonline.net/
[12] http://runthatapp.com/
[13] http://appetize.io/
[14] https://www.genymotion.com/
[15] http://www.mobilephoneemulator.com/
[16] https://archon-runtime.github.io/
[17] http://www.manymo.com/

disso ele provê um sistema de armazenamento para guardar suas aplicações Android (.apk), ele também contém um repositório de várias aplicações que podem ser utilizadas com o emulador. Roda em qualquer navegador, porém é mais eficiente no Google Chrome.

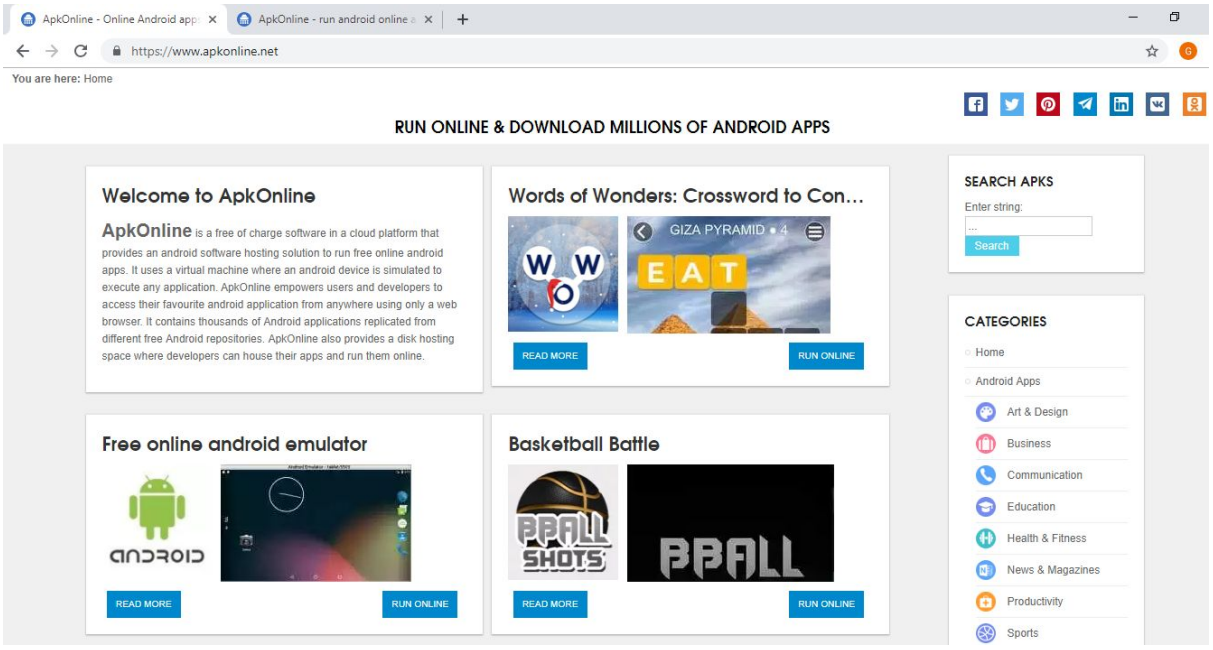

Figura 11. Página inicial APK Online. (AUTOR, 2019)

Nessa tela inicial o usuário pode escolher rodar um emulador vazio ou pegar um apk(aplicação) já disponível no repositório desse software para rodar direto.

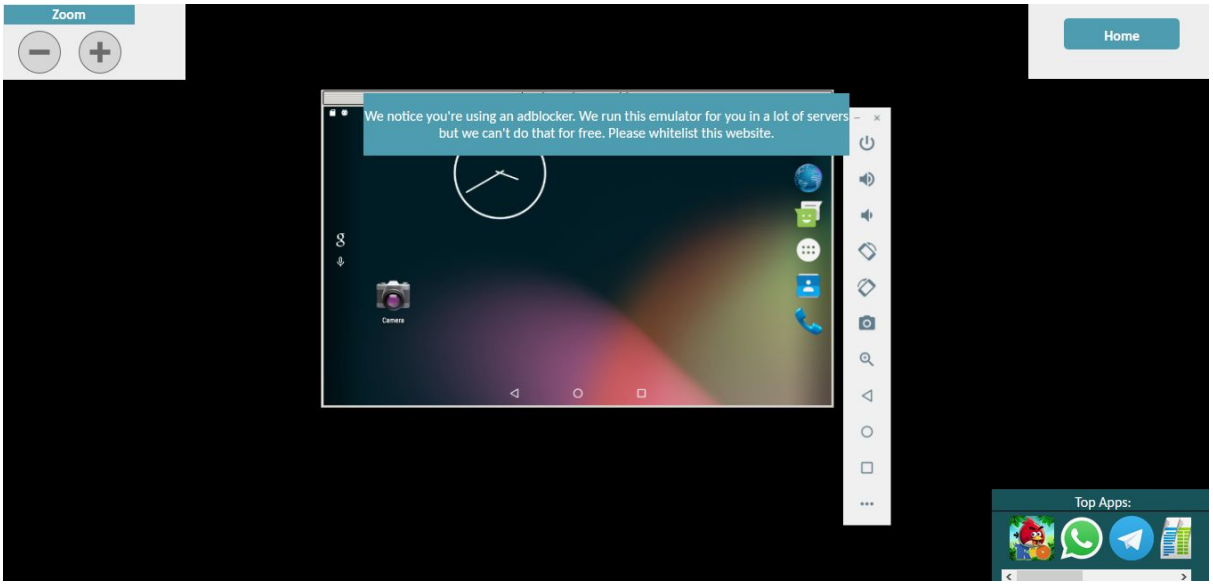

Figura 12. Emulador do APK Online rodando. (AUTOR, 2019)

Após iniciar o emulador basta utilizá-lo como um aparelho Android normalmente.

3.3.2 Appetize IO

O Appetize IO é um software licenciado, e ele é um emulador que tem propósitos gerais também. Além do emulador de Android ele tem um emulador de iOS, e a forma que ele funciona da seguinte forma:

- O server faz uma transmissão de uma sequência de *screenshots* tiradas de uma instância de um emulador e envia para o navegador usando websocket
- No navegador, código em javascript realizará atualizações em um canvas que seria a tela do dispositivo.
- Para receber a interação do cliente, a interação do mouse com o canvas é enviada para o servidor e no servidor é traduzido para comandos no emulador.
- Repetem-se as etapas anteriores

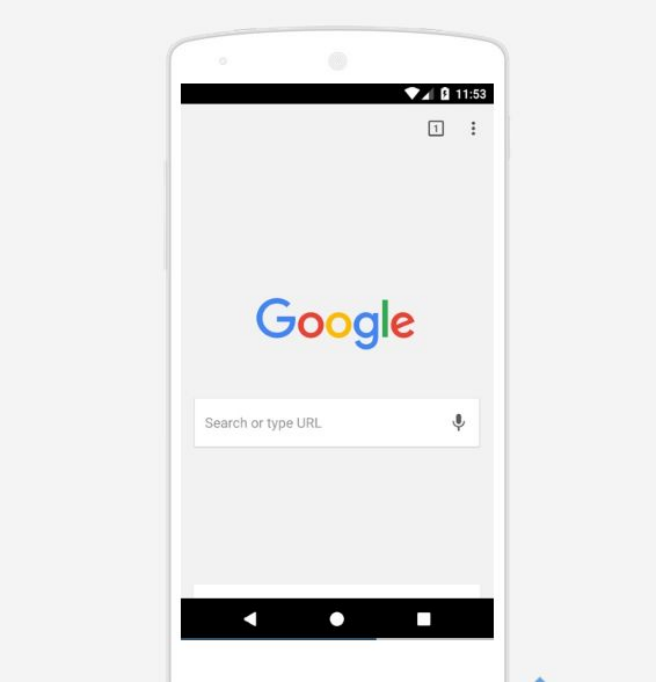

Figura. 13 Emulador Appetize IO. (AUTOR, 2019)

Para o websocket essa solução utiliza a API Socket.IO e para fazer a tradução de canvas para tela do emulador é utilizado a tecnologia ADB (Android debug bridge) que está disponível no Android SDK. Ele também dá suporte a upload de arquivos para execução no próprio emulador, e por ser um canvas simples, ele dá suporte a todos os navegadores.

3.3.3 Run That APP

Software licenciado, utiliza da mesma ideia do Appetize IO, ou seja, um emulador que roda em um servidor e os screenshots desse emulador são enviadas para o navegador, o navegador cria um canvas e atualiza a tela com esses *screenshots* e os cliques do usuário no canvas são enviados para o servidor, que fará a tradução do canvas para o emulador, executando ações de touch no emulador. Ele também permite o upload de arquivos, porém é um pouco diferente, pois após enviar o arquivo, o usuário deve solicitar do serviço um link via email para acessar uma instância do emulador com seu aplicativo rodando. Além disso, como o Appetize IO, o fato deste software utilizar apenas html e javascript para mostrar o emulador para o usuário faz com que ele rode em qualquer navegador.

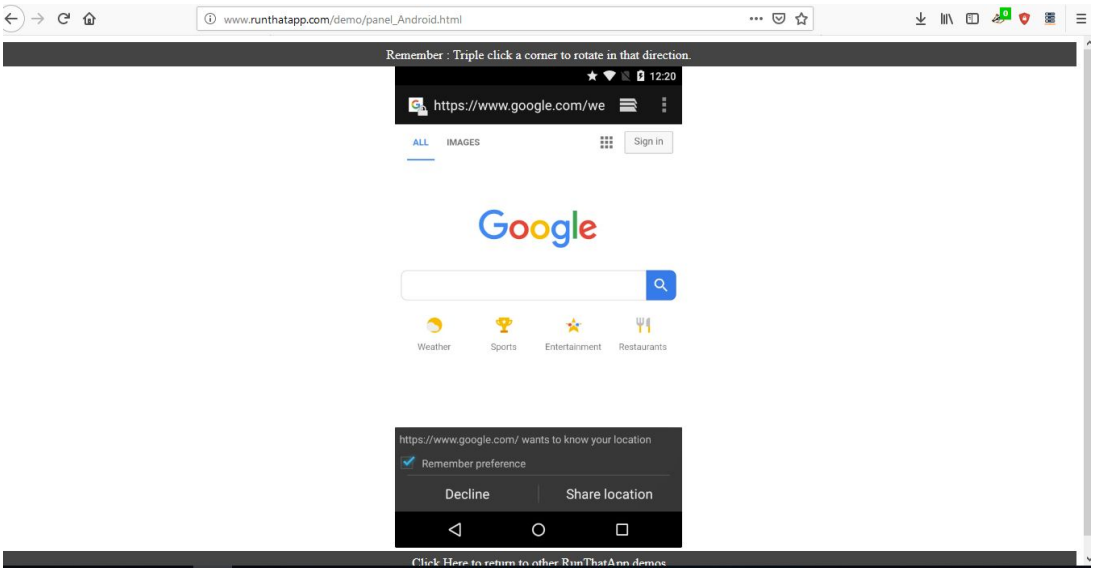

Figura. 14 Emulador RunThatApp. (AUTOR, 2019)

O Run That APP também disponibiliza emulador para iOS.

3.3.4 Genymotion

Genymotion é mais um software licenciado, ele oferece dois tipos de serviço, genymotion *cloud* e genymotion *desktop.* O Genymotion *desktop* é nada menos que um

emulador de Android normal, uma aplicação. Já o Genymotion *cloud* utiliza do *Android virtual device*, da mesma forma que a aplicação citada anteriormente, o APK Online.

O Genymotion *cloud* disponibiliza muitas funcionalidades para executar no navegador aplicações criadas pelo o usuário, também permite que o usuário rode múltiplas instâncias do emulador para automação de testes por exemplo, além de ser completamente compatível com vários *frameworks de* automação de testes.além de disponibilizar algumas APIs para realizar o controle dessas AVDs criadas pelo usuário, essas APIs incluem controle das AVDs que estão rodando, executar algumas macros de cliques ou alterar níveis de bateria, ou até mesmo rodar código Java para fazer alguns ajustes no dispositivo durante execução de testes.

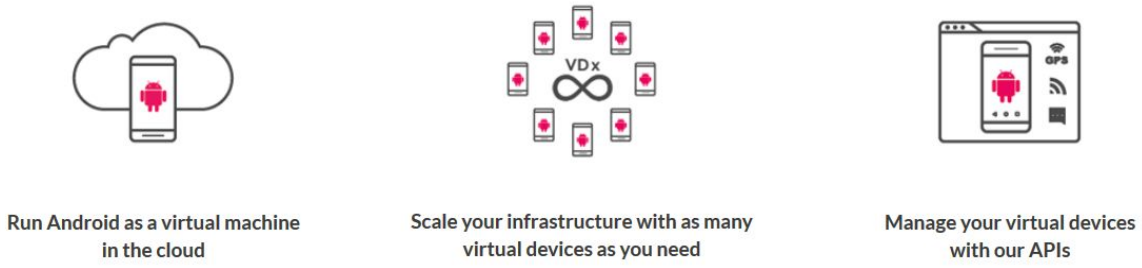

Figura 15. Funcionalidades (GENYMOTION, 2019)

No lado do servidor ele roda uma máquina virtual com as *virtual devices*

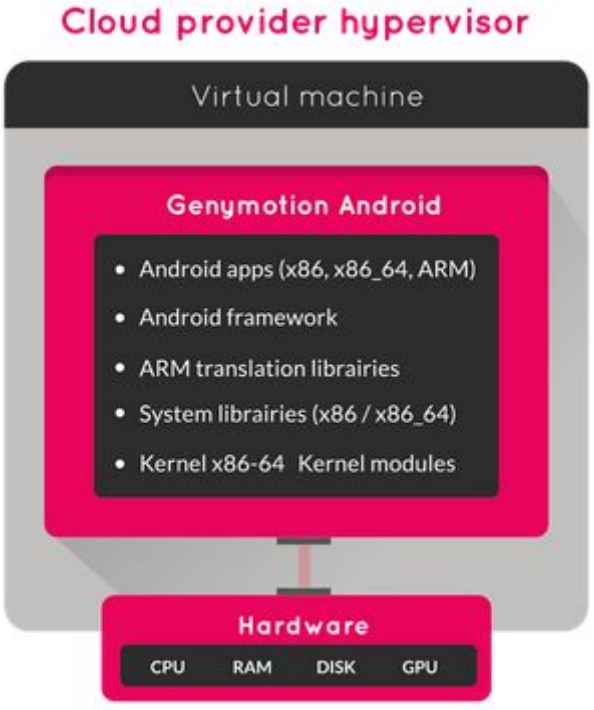

Figura. 16 Estrutura do emulador genymotion (GENYMOTION, 2019)

O Genymotion também disponibiliza a possibilidade de fazer upload de APKs, e funciona simplesmente arrastando o APK do computador do usuário para a aba do navegador em que se encontra o emulador.

3.4 Discussão

É possível observar que esse tipo de solução de software é bem escasso, já que foram encontradas apenas quatro aplicações relevantes, e dessas quatro soluções, três delas são software licenciados. Além disso, são utilizadas apenas duas formas de solucionar o problema, uma é pela utilização de websocket, e a outra é por utilização de *Android virtual devices.* Foram encontradas algumas outras soluções, mas essas outras ou necessitavam de algum tipo de download de software externo, ou não permitia executar aplicações do usuário no emulador.

É notável então a carência de softwares desse tipo, e a gama de formas de resolver o problema é bem limitada. A tabela 6 faz um comparativo de alguns aspectos importantes para a solução do problema proposto neste trabalho. Os aspectos considerados são:

- A aplicação é gratuita? Para servir como uma solução para o problema, o software deve ser gratuito em primeiro lugar, pois será utilizado em ambientes escolares.
- Tipo de solução. A forma em que o problema é resolvido. Esse aspecto permite analisar como os softwares que já existem realizam o trabalho de emular Android no navegador. Serve como um parâmetro para decisões que estão relacionadas ao uso de recursos de hardware (memória, processamento, etc.)
- Upload de arquivos. Se o software permite ou não realizar o upload de um arquivo APK externo e executá-lo de alguma forma no emulador. Isso é importante, pois o problema apresentado gira em torno de executar aplicações desenvolvidas no computador, e a necessidade de testar essas aplicações de forma simples e ágil.
- Ferramentas para controlar múltiplas instâncias. Como o objetivo desse projeto é o uso em ambientes escolares, múltiplas instâncias do emulador serão postas em execução simultaneamente, portanto uma ferramenta/interface para controlar essas instâncias se faz necessária.
- Integração *App inventor*. Novamente, o objetivo desse projeto é o uso em ambientes de escola em que o *App Inventor* é utilizado para ensinar programação, portanto é importante a existência dessa integração.

- Gestão de uso de memória. Esse aspecto se faz necessário pelo fato de vários emuladores serem executados ao mesmo tempo, para monitorar o uso de memória e realizar um gerenciamento mais inteligente das instâncias que estão rodando.

Essas características então, além de servirem para comparar as soluções de emuladores Android *web* que existem hoje, servem também para levantar quais desses softwares atendem aos requisitos que o problema apresentado demanda.

	APK Online	**Appetize IO**	**Run That App**	**Genymotion**
Gratuito	Sim	Não	Não	Não
Tipo de Solução	*Android Virtual Device*	WebSocket com ADB HTML5 e JS	WebSocket com ADB HTML5 e JS	*Android Virtual Device*
Upload de arquivos	Upload para servidor do host e Upload para AVD persistente	Upload para servidor do host(roda no emulador)	Upload para servidor do host(roda no emulador)	Upload para AVD persistente (*Drag and drop*)
Ferramentas para controlar múltiplas instâncias	não.	não.	não.	Sim.
Compatibilidade com Navegadores	Total, porém roda melhor em Google Chrome	Total	Total	Total
Integração *App Inventor*	Não	Não	Não	Não
Gestão de uso de memória	Não.	Não.	Não.	Não.

Tabela 6. Comparação de resultados

Analisando a tabela 6, observa-se que as soluções são muito similares, o Genymotion porém destaca-se um pouco mais, pelo fato de ele disponibilizar diversas ferramentas que auxiliam na hora de testar e desenvolver uma aplicação Android. Sua maior desvantagem, além de também ser um dos motivos dele não se adequar como solução para o problema apresentado, é o fato de ser um software licenciado. Já o APK Online é gratuito e faz o

trabalho muito bem, seu maior problema é que muitas vezes o emulador demonstra lentidão na hora de executar aplicações e também o fato de não ser integrado ao *App Inventor*. As outras duas soluções também dão conta do trabalho muito bem, são praticamente iguais no quesito de arquitetura, salvo algumas particularidade do Appetize IO, e ainda sim ambas têm a limitação de serem pagas, o que inviabiliza seu uso no presente projeto.

Então, analisando todas essas soluções é possível perceber um padrão, as ferramentas pagas não servem como solução e o Apk Online, que não é pago, não se adequa como solução para o problema apresentado pois apesar de ser uma ferramenta gratuita, ele apresenta instabilidade e alguns problemas de travamento. Além disso, nenhuma solução apresenta nenhum tipo de integração com o *App Inventor*.

Existe também o fato de que exceto o GenyMotion, todas as outras soluções não apresentam uma forma de gerenciar múltiplas instâncias do emulador, e isso é será um instrumento necessário pois para a solução do problema apresentado a execução simultânea de diversas instâncias do emulador será quase a regra devido a natureza do ambiente onde se deseja aplicá-lo, então uma interface para gerenciar as instâncias que estão executando em um dado momento é algo indispensável.

3.4.1 Ameaça da validade do mapeamento sistemático.

Alguns dos motivos pelos quais esse estudo poderia ter sua validade prejudicada seria por exemplo a inclusão de emuladores com acesso web que não fossem de Android, ou então que não permitissem executar aplicações que o usuário criou. Além disso apenas vinte páginas da ferramenta de busca foram consideradas, isso é uma ameaça pois ao excluir outras páginas possivelmente pode-se excluir outras soluções.

Para mitigar essas ameaças foram incluídos nos critérios de inclusão/exclusão a necessidade do software disponibilizar emulador Android e que esse software deve possibilitar a execução de aplicações criadas pelo usuário. Além disso, os softwares que foram encontrados na execução da busca foram todos encontrados nas primeiras duas páginas, então pela natureza da ferramenta de busca, que ordena os resultados pela relevância, considera-se baixa a probabilidade de que outras soluções viessem a aparecer em páginas subsequentes.

4. Desenvolvimento

Neste capítulo serão apresentados como foram feitos o projeto e o desenvolvimento da aplicação. Primeiramente é apresentada toda a modelagem da solução, partindo dos requisitos, passando pelos casos de uso até diagramas de sequências, mostrando também alguns protótipos de telas. Após demonstrada como foi feita a modelagem, é apresentado como o software foi implementado.

4.1 Análise e Projeto

Nesta seção é apresentada a proposta de solução desenvolvida pelo presente trabalho. Primeiramente serão levantados requisitos funcionais e não funcionais baseando-se no problema apresentado no capítulo de introdução. Também se leva em conta a análise do estado da arte, na qual é possível identificar que não existe nenhuma solução, já implementada, que seja adequada para o problema apresentado. Também serão considerados aspectos e necessidades do MIT AI2 Companion para desenvolvimento dessa solução (conexão com wi-fi, portas necessárias liberadas).

4.1.1 Requisitos

Os requisitos foram levantados com base em aspectos técnicos necessários para atender as especificações do MIT AI2 Companion, além de aspectos funcionais para a execução dos aplicativos desenvolvidos. Para tanto, foram consideradas as funcionalidades relevantes para o contexto dos softwares encontrados durante o levantamento do estado da arte. A tabela 7 mostra então os requisitos funcionais e não funcionais.

Requisitos funcionais		
ID	**Requisito**	Descrição
RF1	Ambiente que disponibiliza emuladores	Prover um ambiente web que permita ao usuário acessar um emulador Android.
RF2	Containers com Dispositivos Virtuais	Prover *containers* , cada um desses *containers* terá um Dispositivo Virtual, que será o emulador disponível para cada usuário.

RF3	Roteamento de portas dos containers no servidor	O roteamento das portas do servidor e dos containers deve permitir que vários usuários possam utilizar containers diferentes, ao mesmo tempo.
RF4	Conexão *App Inventor* ↔ MIT Companion	Permitir a conexão entre o MIT Companion em um dispositivo virtual(que está em um container) e o *App Inventor* em um navegador no pc.
RF5	Módulo de administração de containers	O módulo de administração de containers deve permitir: criar, remover ou reiniciar um container dentro do servidor.
RF6	Gerenciamento de recursos	Possibilitar gerenciamento de recursos, como memória, armazenamento.
Requisitos não funcionais		
ID	**Requisito**	**Descrição**
RNF1	Possibilidade de executar em navegadores	Pelo fato do emulador ser web, não existe a restrição de sistema operacional. Focar em Chrome e Firefox versões 70 e 64 respectivamente.
RNF2	Múltiplos usuários simultâneos em emuladores diferentes	Devem ser permitidos ao menos 20 usuários utilizando a aplicação simultaneamente, e de forma que a execução de um usuário não interfira na execução de outro.
RNF3	Emuladores de Android	As aplicações desenvolvidas pelo *App Inventor* são para a plataforma Android, então o emulador deve emular Android versão?.
RNF4	AVDs padronizadas. aplicações e etc.	As imagens do dispositivo virtual devem ser padronizadas para não haver nenhum tipo de diferença de desempenho, ou problema de execução das aplicações.
RNF5	Limitar o uso de memória dos dispositivos virtuais	Memória é um recurso limitado, múltiplos emuladores rodando são problema para desempenho, limitar a quantidade de memória que os dispositivos virtuais utilizam é uma solução.128MB por dispositivo
RNF6	Interface Web	Disponibilizar o módulo através de uma interface web para o administrador.
RN7	AI2 Companion	Permitir a instalação da versão mais atual do AI2 Companion.

Tabela 7. Requisitos funcionais e não funcionais do projeto.

4.1.2 Desenho da solução.

Nesta seção é apresentada uma proposta geral de arquitetura para a solução técnica. A solução consiste em um ambiente web que disponibilize uma instância de um emulador de Android para executar aplicativos que são desenvolvidas com *App Inventor* do MIT. Na introdução deste presente trabalho foi apresentado a necessidade de um emulador com acesso pelo navegador para o uso no projeto Computação na Escola, pois algumas dificuldades atrapalham o processo de ensino de programação, essas dificuldades estão bastante relacionadas com a parte de testar a aplicação desenvolvida, pois existem fatores que adicionam complexidade, ou as vezes impossibilitam, o uso dos ambientes de *live-testing* que o *App Inventor* disponibiliza (MIT AI2 Companion, emulador desktop, USB).

Também durante o estudo do estado da arte, não foi encontrada nenhuma solução que atenda completamente o problema apresentado. Por essa razão, faz-se necessário o desenvolvimento de um ambiente que atenda às demandas. A figura 17 mostra uma visão geral para a arquitetura da solução.

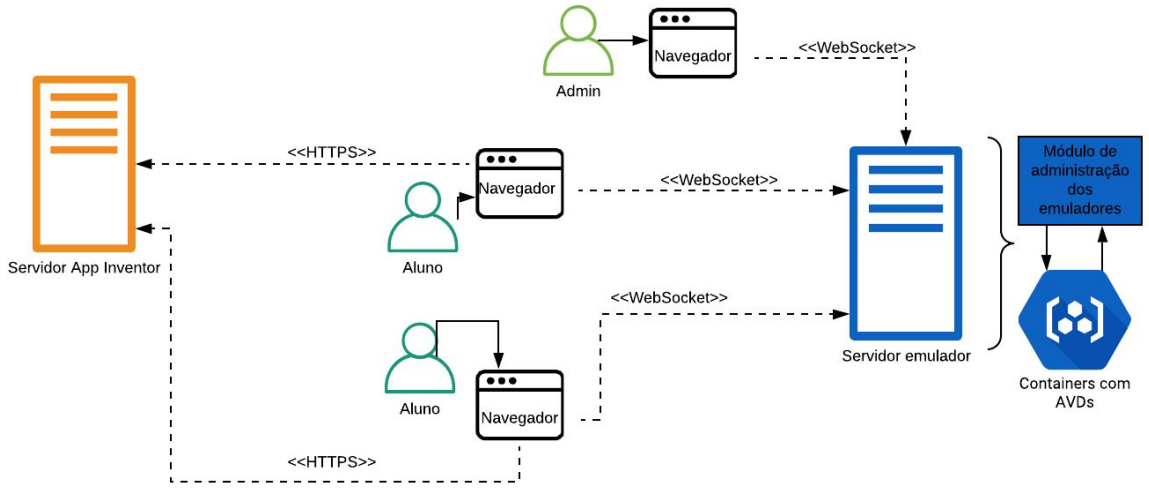

Figura 17. Visão geral da arquitetura da solução. (AUTOR, 2019)

Dentro do servidor do emulador existem, o módulo de administração dos emuladores e os containers que vai conter os emuladores em si. O módulo de administração controla o *deploy*, remoção e através dele, é possível realizar a gerência cada um dos containers, um usuário, o administrador, tem acesso a esse módulo através de um ambiente web.

Cada aluno, pode acessar um ambiente web que lhe dá a opção de “pegar” uma instância do emulador automaticamente, se já existir uma instância livre o sistema irá

automaticamente atribuir ao usuário essa instância, caso não exista, o próprio sistema cria uma instância, ou o administrador pode criar uma manualmente através do seu ambiente de administração. Através do protocolo WebSocket o usuário então se comunica com um container específico no servidor, por uma porta específica, que será atribuída a cada container para permitir acesso através do navegador, dessa forma ninguém pode entrar no meio dessa execução, e da mesma forma essa execução não atrapalha outras execuções.

O protocolo WebSocket permite a comunicação bidirecional, *full-duplex,* sobre um soquete TCP(WebSocket, 2019). Colocando em palavras mais simples, o cliente e o servidor estabelecem uma conexão persistente no qual ambos os lados enviam e recebem dados a qualquer momento. Essa característica o torna perfeito para a aplicação, pois enquanto o servidor manda dados sobre o emulador para tanto o aluno quanto para o administrador, os alunos estarão enviando para servidor, comandos como cliques e execução de aplicações. Todas estas ações não têm um momento para serem enviadas, portanto a conexão deve ser mantida aberta sempre pois não se sabe quando será necessário o envio ou recebimento dos dados.

Do outro lado os alunos estão conectados com o *App Inventor*, desenvolvendo suas aplicações, e a partir dali eles geram o código alfanumérico e o QR *Code* para conectar com o MIT AI2 Companion, que estará sendo executado no emulador, e assim a conexão entre o emulador e o *App Inventor*, através do MIT Companion, permite um ambiente de *live-testing* no navegador.

4.1.3 Casos de Uso

Tomando em conta os requisitos levantados na seção 4.1.1, e o desenho da solução apresentado na solução 4.1.2, foram identificados 3 atores principais do sistema:

- O Aluno, que irá utilizar os emuladores para executar as aplicações que ele tenha desenvolvido no *App Inventor*,
- O Administrador, que irá utilizar o ambiente de administração, que é uma interface com o módulo de administração no servidor, para gerenciar tanto os emuladores rodando, quanto criar novas instâncias manualmente, e
- O *App Inventor*, que vai permitir que o aluno acesse uma instância do emulador.

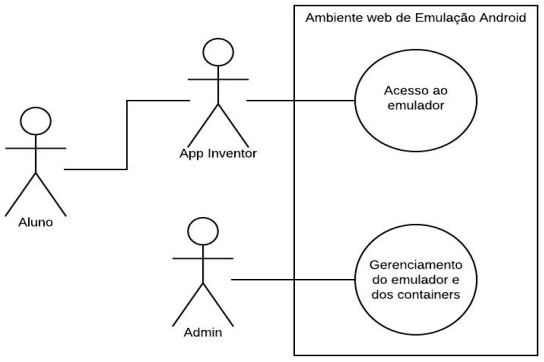

Figura 18. Diagrama de casos de uso. (AUTOR, 2019)

A figura 18 mostra o diagrama de casos de uso para os dois casos de uso que estão descritos a seguir.

[UC01 - Acesso ao emulador]:

Descrição do caso de uso: O aluno, através de um navegador, acessa o *App Inventor*, e pelo *App Inventor*, têm acesso ao emulador para executar sua aplicação.

Requisitos: **[RF1, RF2, RF3, RF4, RNF7]**

Ator(es): Aluno, *App Inventor*

Pré-Condições:

- Aluno deve estar utilizando um navegador suportado [**RNF1]**;
- Aluno deve estar logado no *App Inventor*

Pós-Condições:

- Aluno consegue obter uma instância do emulador e executar sua aplicação

Fluxo Principal

[FP01 - Acessar emulador]

1. O aluno abre um navegador e o *App Inventor*;
2. O aluno clica no menu para iniciar um emulador no *App Inventor*;
3. O *App Inventor* cria um *token* único e manda uma requisição para o servidor com esse *token*
4. O servidor verifica se há uma instância disponível;
5. O servidor atribui ao aluno uma instância;

Fluxo Alternativo

[FA01 - Criar instâncias automaticamente]

3. O servidor verifica que não há instâncias disponíveis**;**
    - Cria uma instância automaticamente utilizando chamadas de sistema;

Fluxo Exceção

[FE01 - Número máximo de instâncias atingido]

3. O servidor tenta criar uma instância, porém já atingiu seu limite máximo (uso de memória máximo estipulado foi ultrapassado);
    - Retorna uma mensagem de erro informando o ocorrido;

[UC02 - Gerenciamento do emulador e dos containers]

Descrição do caso de uso: O administrador, através do ambiente web de administração, consegue gerenciar os emuladores, criar instâncias, remover e observar recursos utilizados.

Requisitos: **[RF5, RF6]**

Ator(es): Administrador.

Pré-condições:

- Administrador deve estar logado;

Pós-condições:

- Depende da opção escolhida pelo Administrador;

Fluxo Principal

[FP01 - Gerenciamento de instâncias]

1. O administrador abre um navegador e acessa o ambiente de administração;
2. O administrador seleciona uma opção na aba de gerenciamento;
3. O servidor executa a ação do administrador;

Fluxo Alternativo

[FA01 - Criar Instâncias]

2. O Administrador escolheu criar um instância
    - O servidor dá a opção do administrador configurar manualmente atributos como: portas, memória, ou deixar o servidor atribuir automaticamente.

Fluxo Alternativo

[FA02 - Remover Instâncias]

2. O administrador escolhe remover uma instância;
    - O servidor mostra as instâncias que estão rodando;
    - Administrador escolhe uma para remover;

[FA03 - Gerenciar recursos do emulador]

2. O administrador escolhe gerenciar recursos de uma instância;
    - O servidor mostra as instâncias rodando;
    - O administrador escolhe uma, e o recurso que ele deseja alterar;
    - Administrador altera recurso;

4.1.4 Protótipos de tela

Essa seção apresenta alguns protótipos de tela para cada caso de uso definido anteriormente, e detalha um pouco melhor as funcionalidades existentes em cada um. A figura 19 mostra a tela inicial geral do ambiente, ela não depende de nenhum dos casos de uso em particular.

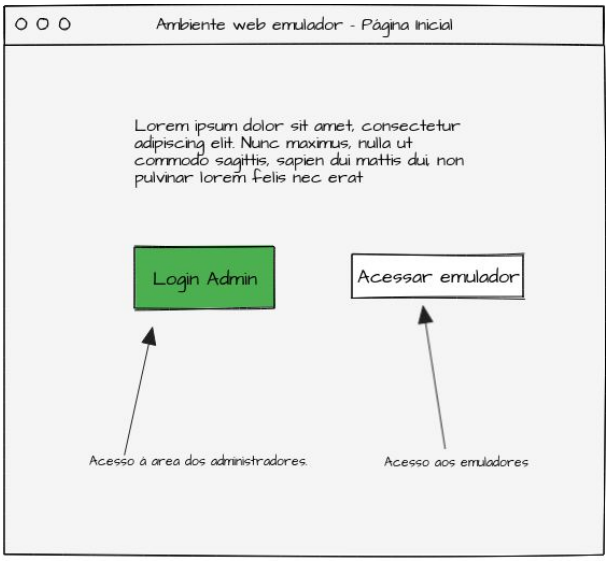

Figura 19. Página inicial da aplicação. (AUTOR, 2019)

Protótipo de tela - UC01:

No primeiro caso de uso, é definida então a obtenção do emulador pelo aluno, na tela inicial (figura 19) o aluno clica no botão “Acessar Emulador” para pegar uma instância do navegador e poder iniciar o uso. O aluno então é redirecionado para um emulador, o qual somente ele terá acesso enquanto a janela do emulador estiver aberta.

Figura 20. Tela do emulador. (AUTOR, 2019)

A “entrega” do emulador para os alunos será feita de uma das duas formas:

1. Ao clicar no botão “Acessar emulador” a aplicação irá verificar se já não existe nenhum emulador existente o qual não esteja sendo acessado por ninguém. Caso exista essa instância não utilizada, ele simplesmente redireciona o aluno para ela.
2. Caso não exista esse emulador, então por meio de chamadas do sistema, a aplicação irá instanciar um novo emulador e redirecionar o aluno.

Protótipos de tela - UC02:

O segundo caso de uso, define a parte administrativa do sistema, onde um usuário (Administrador) pode realizar operações nas instâncias dos emuladores, ou então criar instâncias manualmente. As operações que o admin pode executar em um emulador já existente são: remover uma instância, caso ninguém esteja utilizando o processo é terminado,

e também gerenciar recursos dessa instância como memória e armazenamento. A figura 20 mostra a tela de login do admin.

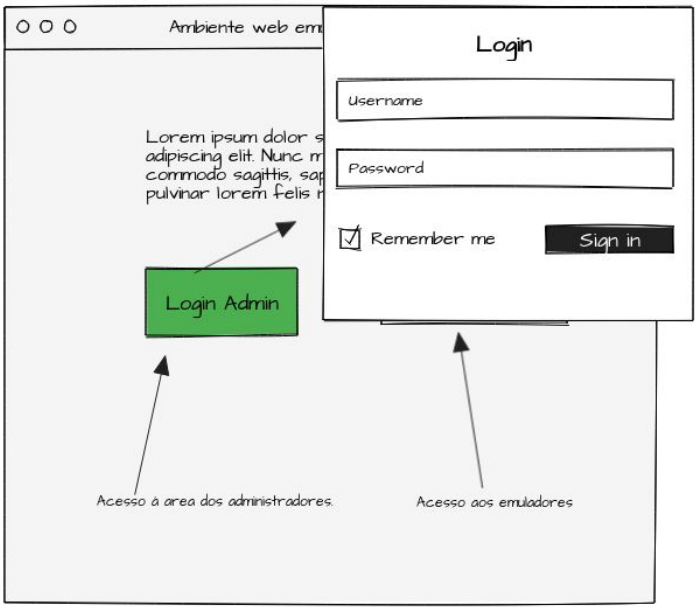

Figura 21. Página de login do admin. (AUTOR, 2019)

A figura 22 mostra a página de administração dos emuladores, onde o admin pode ver os emuladores que estão rodando, os que estão efetivamente em uso. Essa página também possibilita instanciar, editar ou remover um emulador, além de permitir a visualização das quantidades de recursos, como memória e disco, para cada emulador, e o IP de quem está utilizando, para os emuladores que tiverem alguém utilizando. No final da página também é possível visualizar o total de recursos que estão sendo utilizados.

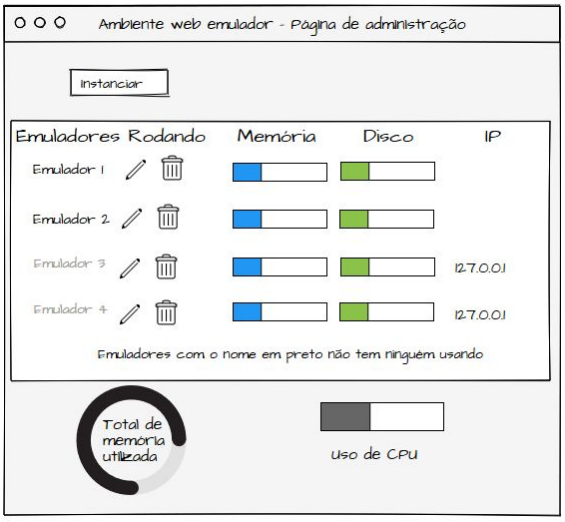

Figura 22. Página de administração. (AUTOR, 2019)

A figura 22 mostra como é feita a instanciação de um novo emulador. Acessando pelo menu superior, o administrador pode criar o emulador manualmente e pode customizar a porta que ele deseja que o emulador tenha, e a quantidade de memória. Se esse campos forem deixados em branco, no caso da porta, o sistema atribui uma porta disponível ao emulador automaticamente, e no caso da quantidade de memória, é atribuído um valor padrão para o emulador.

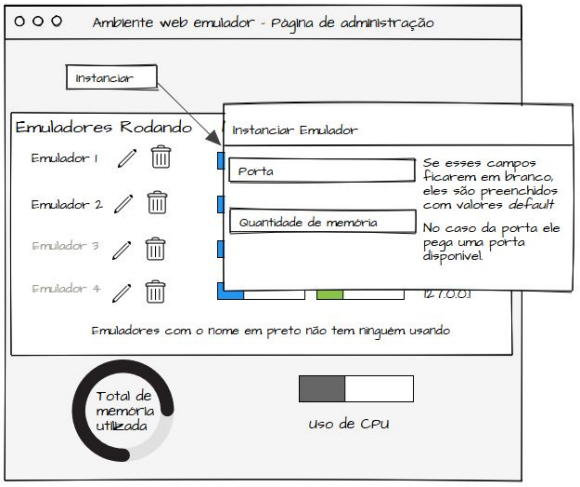

Figura 23. *Modal* que permite a instanciação do emulador pelo admin. (AUTOR, 2019)

Quando o admin desejar instanciar um emulador, basta clicar no “instanciar” no menu superior da página de administração e um painel abre pedindo a porta e uma quantidade de memória para essa instância, se qualquer um dos dois forem deixados em branco, serão assumidos valores padrão. A figura 23 mostra o painel.

Para remover ou alterar os recursos de um emulador já existente, os botões da lixeira e do lápis (figura 22) executam respectivamente essas funções. Para remover o emulador basta clicar no botão e confirmar o encerramento do emulador. Já para alterar, o administrador clica no botão de editar do emulador, e muda a quantidade do recurso que ele deseja, conforme a figura 24 mostra.

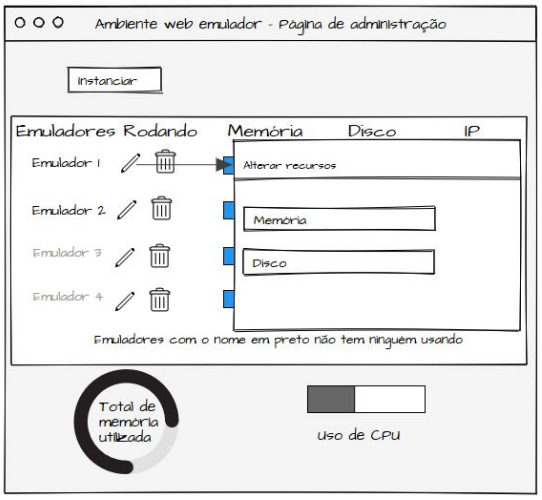

Figura 24. Opção de alterar recursos de um emulador. (AUTOR, 2019)

4.1.5 Diagramas de sequência:

A seguir, são apresentados alguns diagramas de sequência de dizem respeito aos casos de uso apresentados anteriormente.

A figura 25 mostra o diagrama de sequência para o primeiro caso de uso, onde o aluno realiza o login no App Inventor, e por meio deste, solicita uma instância do emulador. O App Inventor vai gerar um token único para o aluno, baseado na sessão, e gerar uma requisição HTTP para o sistema do emulador. O controlador das requisições do lado do servidor vai tratar essa solicitação, e criar, ou pegar uma instância inutilizada, para retornar para o App Inventor que apresenta para o aluno. Ou caso não seja possível, é retornada uma mensagem de erro.

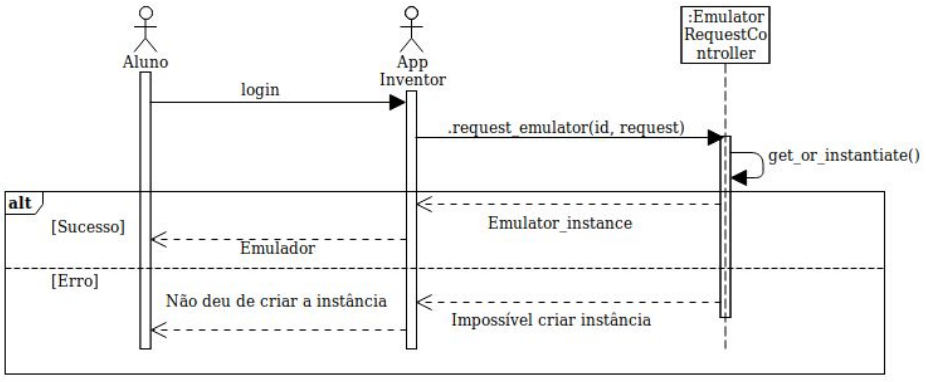

Figura 25. Diagrama de sequência para o caso de uso 01. (AUTOR, 2019)

A figura 26 mostra o diagrama de sequência para o caso de uso 02, onde as funções de administrador são definidas, como criar uma instância, ou remover uma. O admin, através

do ambiente de administração (figura 22) pode executar essas funções através de requisições HTTP.

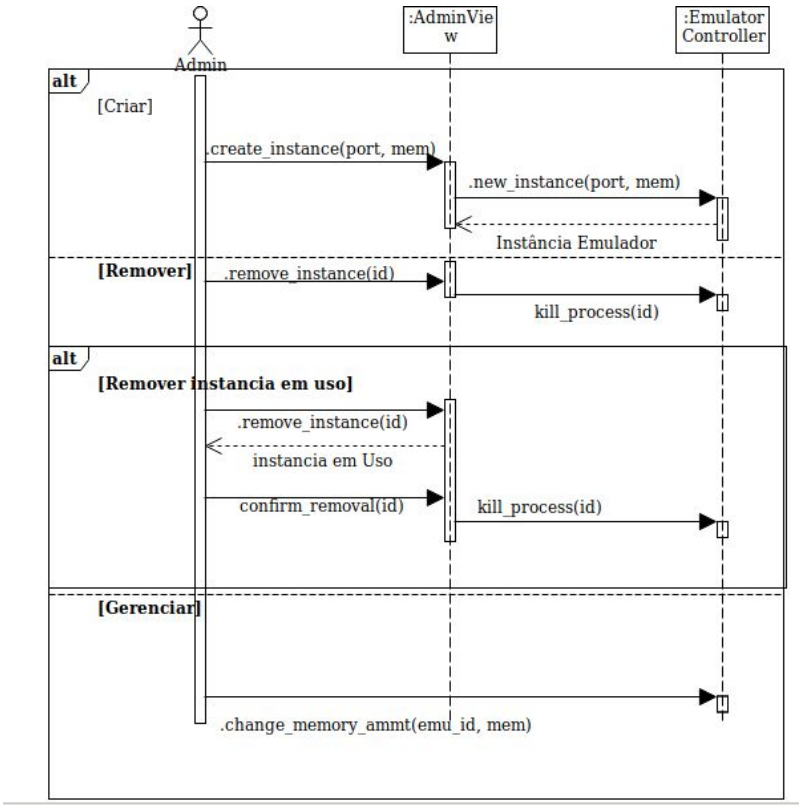

Figura 26. Diagrama de sequência para o caso de uso 02. (AUTOR, 2019)

4.1.6 Tecnologias utilizadas.

A implementação deste sistema implica no uso de diversas tecnologias, que combinadas tornam possível a arquitetura necessária, conforme mostra a tabela 8.

Tecnologia	Uso	Fonte/URL
AVD	Android Virtual Devices serão os emuladores	https://developer.android.com/studio/run/managing-avds
Android SDK	Disponibiliza comandos para gerenciamento das AVDs	https://developer.android.com/studio
Docker	Vai ser o container que vai ter a AVD	https://www.docker.com/
Python+Django	Utilizado para desenvolver o ambiente web	https://www.python.org/ https://www.djangoproject.com/

noVNC	Disponibiliza o container para uso.	https://novnc.com/info.html

Tabela 8. Tecnologias utilizadas e seus usos.

Cada emulador do sistema é uma AVD (*Android Virtual Device* - Dispositivo virtual Android), ele é nativo do Android Studio, porém é disponível em uma forma *standalone* no Android *Software Development Kit (*SDK*),* junto com algumas ferramentas que são úteis para gerenciar os emuladores por meio de linha de comando. Cada AVD rodará dentro de um container Docker, que irá ser disponibilizado para acesso.

O acesso a esses containers contendo as AVDs se dará por meio do ambiente web desenvolvido utilizando a linguagem de programação Python junto de seu framework Django para desenvolvimento de aplicações web, e também utilização do noVNC para permitir o acesso de tais *containers* por meio de websocket.

4.2 Implementação

Nesta seção é apresentada a implementação da solução, mostrando e justificando as tecnologias escolhidas além das decisões de projeto tomadas, abordando também as dificuldades encontradas durante todo o processo de desenvolvimento e os resultados finais obtidos da implementação. Também serão mostrados os resultados dos testes executados na aplicação.

4.2.1 Emulador

O emulador foi desenvolvido utilizando, principalmente, *Android Virtual Devices* e Docker, além de outras tecnologias (que estão contidas no container do docker) para permitir o acesso do emulador pela web.

4.2.1.1 *Android Virtual Device*

Conforme já indicado, para servir de base para o emulador, foi utilizado os AVDs (*Android Virtual Devices*). Esses emuladores fazem parte do Android SDK do Android *Studio.* O objetivo desses emuladores é serem compactos e leves, que são características as quais este trabalho necessitava.

Esse SDK permite a criação de imagens do emulador por linha de comando, e disponibiliza uma série de opções de configuração para tais emuladores, que podem ser definidas durante a criação da imagem, e em alguns casos, no comando que executa o emulador. A tabela 9 mostra alguns exemplos de configurações disponíveis para alterar em uma imagem de AVD via linha de comando.

Comando	Descrição	Exemplo de uso
-memory tamanho	Especificar tamanho da RAM, em MB, de 128 a 4096. Substitui o que foi definido na criação da imagem.	./emulator @emulator -memory 256
-sdcard caminho	especifica o nome de um arquivo de imagem para servir de cartão SD para o emulador.	./emulator @emulator -sdcard umCaminho/sdimage.img
-wipe-data	Retorna a imagem para o estado inicial	./emulator @emulator -wipe-data
-http-proxy proxy	Realiza todas as conexões TCP por meio de um proxy.	./emulator @emulator -http-proxy meuProxy:PORTA
-no-boot-anim	Desativa a animação de inicialização do Android para o emulador ter uma inicialização mais rápida	./emulator @emulator -no-boot-anim

Tabela 9. Opções da linha de comando das AVD

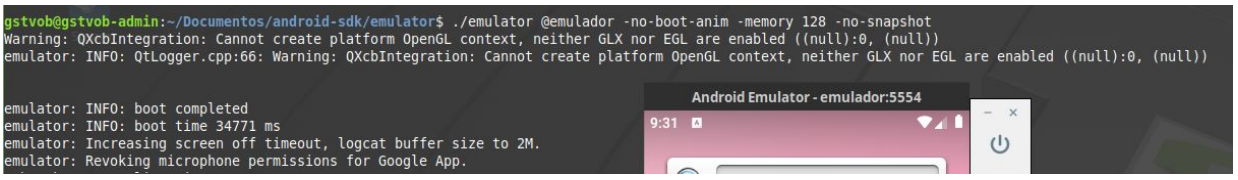

Figura 26 . Rodando emulador na linha de comando. (AUTOR, 2019)

Além das opções que são setadas na criação da imagem e as mostradas na tabela 9, existem outras opções mais avançadas cujo uso não se aplica a este trabalho, portanto não serão detalhadas.

Para atingir os objetivos, foi criada então uma imagem do emulador, e nessa imagem foi instalado o AI2 *Companion* utilizado pela Computação na escola[18]. A figura 27 mostra o emulador rodando, ele foi salvo em um estado que já mantém o Companion aberto para facilitar o uso.

Figura 27. Emulador rodando AI2 Companion. (AUTOR, 2019)

A versão de API, para as AVDs, determina a versão do Android que é utilizada, além da imagem de sistema, que especifica a arquitetura do emulador (X86, X86_64, arm, etc) e outras coisas como, se a imagem vai ter Playstore, ou será uma imagem limpa (default). O nível da API basicamente define se um aplicativo pode rodar no emulador ou não (a página do Android SDK tem uma explicação de como especificar o nivel de API que uma aplicação sendo desenvolvida irá necessitar <https://developer.android.com/studio/publish/versioning.html>).

[18] http://www.computacaonaescola.ufsc.br/?page_id=2960

Durante o desenvolvimento desta parte, a maior dificuldade foi encontrar a versão da API que atendesse aos requisitos propostos (vide capítulo 4):

- Atendesse aos requisitos de desempenho (RNF5, RF6). Que não necessite de muitos recursos para rodar de uma maneira eficiente.
- Permitisse a instalação do Companion na versão mais atual (RF4, RNF7).
- Permitisse a conexão do emulador com o App Inventor (RF4)

Após realizar o estudo da ferramenta, foi descoberto que para ter uma simulação de wifi no emulador é necessário que a versão da API do emulador seja 25 ou superior (*release notes* https://developer.android.com/studio/releases/emulator.html#26-1-3). Também foi descoberto que se baixada a imagem do emulador que utiliza o qemu x86 não é possível utilizar a versão mais recente do *Companion*, portanto foi necessário utilizar a versão x86_64. Combinando essas duas especificações, faltava apenas o pacote da imagem, se seria o *default* ou o que possibilita o acesso da *Google Play Store,* e como a playstore não será utilizada (inviável, pois seria necessário criação de contas e etc.) a versão default é a mais limpa e básica, permitindo o uso do emulador com facilidade e eficiência.

Além disso, na seção de testes (seção 4.2.3) são apresentados alguns bugs encontrados, para resolver alguns, a versão da API foi avançada para 28, dessa forma alguns desses bugs foram removidos. Mas isso será descrito com mais detalhe na seção de testes.

4.2.1.2 Containers Docker

Para facilitar a gerência de recursos dos emuladores, os Android virtual devices foram colocados dentro de containers Docker. A imagem do Docker foi criada a partir de uma imagem do Linux Ubuntu (18.04), e dentro dessa imagem foram instaladas as ferramentas necessárias para rodar um *virtual device*.

```
FROM ubuntu:bionic

ENV HOME /root
ENV DEBIAN_FRONTEND noninteractive
ENV LC_ALL C.UTF-8
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US.UTF-8

RUN dpkg --add-architecture i386
RUN apt-get update && apt-get -y install xvfb x11vnc xdotool wget unzip tar supervisor net-tools openbox
ADD supervisord.conf /etc/supervisor/conf.d/supervisord.conf
RUN apt-get install openjdk-8-jdk -y

ENV DISPLAY :0

WORKDIR /root/
RUN wget -O - https://github.com/novnc/noVNC/archive/v1.1.0.tar.gz | tar -xzv -C /root/ && mv /root/noVNC-1.1.0 /root/novnc && ln -s /root/novnc/vnc_lite.html /root/novnc/index.html
RUN wget -O - https://github.com/novnc/websockify/archive/v0.8.0.tar.gz | tar -xzv -C /root/ && mv /root/websockify-0.8.0 /root/novnc/utils/websockify
```

Figura 28. Começo do Dockerfile. (AUTOR, 2019)

Dentro desse Docker foi instalado o x11vnc junto com as ferramentas necessárias para permitir que o container seja acessível com uma GUI pelo navegador. Foram baixadas as versões mais atuais do repositório do próprio ubuntu bionic (18.04)

Aplicação	Uso
X11Vnc	Programa de acesso remoto a computadores rodando X11 (interface gráfica linux).
wget, unzip, tar	Aplicações utilitárias para manipular arquivos dentro do container.
Xvfb	Display virtual (x11).
openbox	Gerenciador de janelas.
net-tools	Ferramentas para rede.
supervisor	Utilitário para monitorar e reiniciar serviços/processos.
noVNC+websockify	Aplicações para permitir o acesso ao X11Vnc no navegador.
openjdk-8	Versão necessária para rodar AVDs

Tabela 10. Pacotes necessários no container

No começo do Dockerfile é possível ver algumas coisas sendo ajustadas, mas as principais a serem notadas são o noVNC em conjunto com o websockify. Em ambos esses pacotes é apenas feito um “wget”, e extraído os conteúdos. Também é feito um *link* simbólico nos arquivos html do noVNC (só para mostrar corretamente a página *web*).

```
RUN wget https://dl.google.com/android/repository/sdk-tools-linux-4333796.zip
RUN unzip sdk-tools-linux-4333796.zip

RUN rm sdk-tools-linux-4333796.zip

RUN mkdir android-sdk
RUN mv tools android-sdk/tools
COPY android .android
COPY rc.xml /root/.config/openbox/rc.xml

RUN echo "y" | ./android-sdk/tools/bin/sdkmanager --install "system-images;android-28;default;x86_64"
RUN echo "y" | ./android-sdk/tools/bin/sdkmanager --install "platform-tools"
RUN echo "y" | ./android-sdk/tools/bin/sdkmanager --install "platforms;android-28"
RUN echo "y" | ./android-sdk/tools/bin/sdkmanager --install "emulator"


EXPOSE 8080

CMD ["/usr/bin/supervisord"]
```

Figura 29. Final do Dockerfile. (AUTOR, 2019)

No final do dockerfile então é instalado o Android SDK e os pacotes necessários para rodar o AVD. Também é inserido no container a imagem que já foi criada do emulador, e adicionado as configurações do openbox. No final é exposta a porta 8080 que serve para acessar o noVNC (navegador) e também é dado início ao *supervisor*.

O *supervisor* é um utilitário que permite que um usuário monitore e controle processos em sistemas operacionais "*UNIX-like*".

No arquivo do *supervisor*, que na imagem 28 é possível ver o arquivo .conf sendo importado no container, são definidos os processos que vão ser executados quando o container for inicializado. Entre esses processos, está o emulador.

```
[program:emulator]
command=/root/android-sdk/tools/emulator @emulador -no-boot-anim
autorestart=true
stdout_logfile=/dev/fd/1
stdout_logfile_maxbytes=0
redirect_stderr=true
```

Figura 30. Emulador sendo rodado no supervisor. (AUTOR, 2019)

```
[program:x11vnc]
command=/usr/bin/x11vnc
autorestart=true
stdout_logfile=/dev/fd/1
stdout_logfile_maxbytes=0
redirect_stderr=true

[program:novnc]
command=/root/novnc/utils/launch.sh --vnc localhost:5900 --listen 8080
autorestart=true
stdout_logfile=/dev/fd/1
stdout_logfile_maxbytes=0
redirect_stderr=true
```

Figura 31. novnc e x11vnc sendo executados no supervisor. (AUTOR, 2019)

4.2.1.3 Juntando as partes do emulador.

Para juntar o emulador então, foi criada uma imagem de um *Android Virtual Device*, e nessa imagem foi inserido o AI2 Companion que é utilizado pela Computação na Escola, junto com o App Inventor do mesmo projeto. Essa imagem foi criada com configurações que visam um melhor desempenho, como utilização de memória e remoção de alguns recursos que não

serão utilizados, e todas essas configurações podem ser alteradas durante a inicialização do emulador pela linha de comando.

```
./emulator @emulador -no-boot-anim -memory 128
```

Figura 32. Rodando emulador na linha de comando. (AUTOR, 2019)

Depois de criada a imagem, foi criado um *container* Docker, e dentro deste *container* foram feitas as configurações necessárias para rodar a AVD (seção 4.2.1.2). Com a imagem Docker feita, agora basta subir o *container* para o emulador poder ser acessado pela web.

```
$ docker run --privileged -p 8080:8080 android_emulator
```

Figura 33. Rodando o container com a AVD e noVNC. (AUTOR, 2019)

O Docker é rodado com o parâmetro privileged pois a AVD necessita de virtualização (KVM) e como é um *container* isso não foi possível, portanto é necessário “burlar” essa necessidade rodando o docker com a flag “--privileged”. Também é mapeada a porta 8080 do host para a porta 8080 do *container* pois é a porta na qual o noVNC está escutando (figura 33). A imagem 34 mostra então, o container sendo acessado pelo navegador.

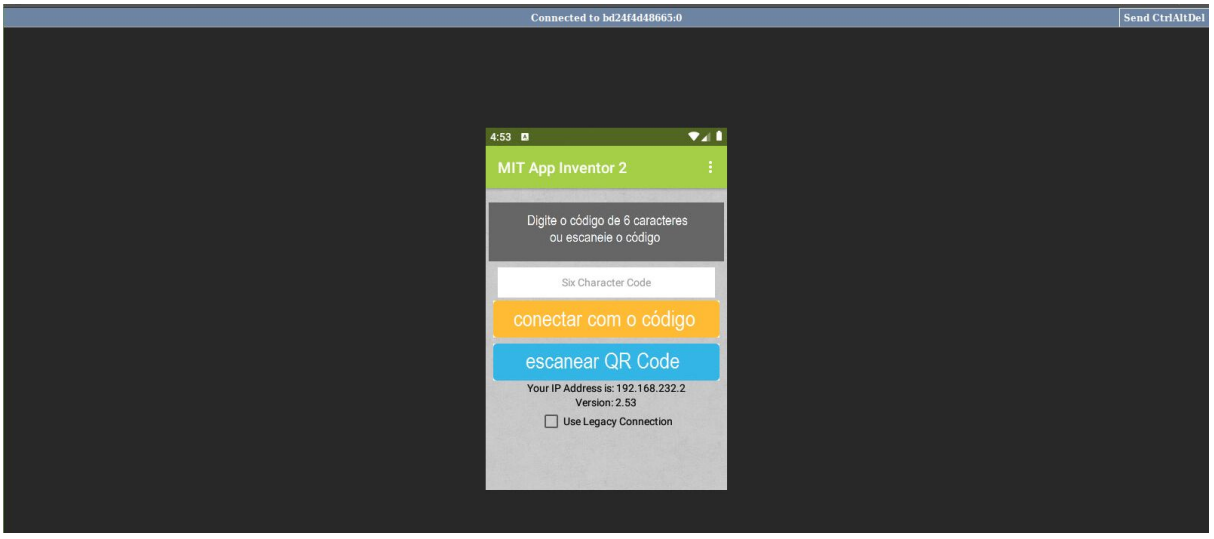

Figura 34. Emulador rodando no navegador. (AUTOR, 2019)

4.2.2 Ambiente Django

Para permitir não somente o acesso, mas também a instanciação e remoção de emuladores, se faz necessário um ambiente web de administração. As seguintes tecnologias foram empregadas para criação do ambiente web:

Nome	Site	Por que foi escolhido?
Python 3.7	https://www.python.org/	Python é uma linguagem de programação simples e poderosa que serve para muitos propósitos.
Django 2.2.4	https://www.djangoproject.com/	Django é um *framework* simples, porém completo, que agiliza o desenvolvimento de aplicações web.
Python Docker API	https://docker-py.readthedocs.io/en/stable/	É uma API que faz tudo o que é necessário em relação aos *containers* Docker de uma máquina *host*
Bootstrap	https://getbootstrap.com/	*Framework* css e js que disponibiliza muitos componentes que agilizam o desenvolvimento de aplicações *web*
JQuery	https://jquery.com/	Framework javascript e de fácil uso
ChartJS	https://www.chartjs.org/	Biblioteca javascript para plotagem de gráficos, utilizado para os gráficos de memória e disco.
subprocess, shutil	https://www.python.org/	Bibliotecas nativas do Python, utilizadas para fazer operações no sistema operacional.

Tabela 11. Tecnologias utilizadas no desenvolvimento do ambiente de administração.

Foi criada então uma aplicação Django, e com o uso deste, e das ferramentas listadas apresentadas, foi possível criar o ambiente que permite realizar as operações no emulador. As aplicações Django geralmente tem modelos, que são os objetos/entidades do sistema, e as *views*, que são os métodos que farão a lógica de uma URL. Como nenhum dado do ambiente web será guardado em um banco de dados, o arquivo *models.py* foi descartado. Porém, apesar de não existirem modelos na aplicação do ambiente, o Django ainda necessita

de um banco de dados pois são utilizados alguns modelos padrão do django, como usuário e autenticação.

Nas *views* são definidas as operações que são executadas toda vez que uma URL é acessada (essas por sua vez são definidas no arquivo de urls). Essas views geram dados que são passados a templates (arquivos HTML), que por sua vez consomem os dados, e geram a visualização da página. A seguir serão apresentadas as funcionalidades que estão disponíveis para executar através do ambiente Django. São apresentadas as telas e elementos do ambiente web, organizadas por caso de uso.

UC01 - UC02 - Página inicial:

Através da página inicial, é possível executar qualquer ação, além de serem exibidos gráficos para mostrar o uso de memória e disco. São mostrados na página inicial os emuladores que estão rodando, a porta na qual eles estão rodando e os botões de ação para esses emuladores. A figura 35 mostra a página inicial

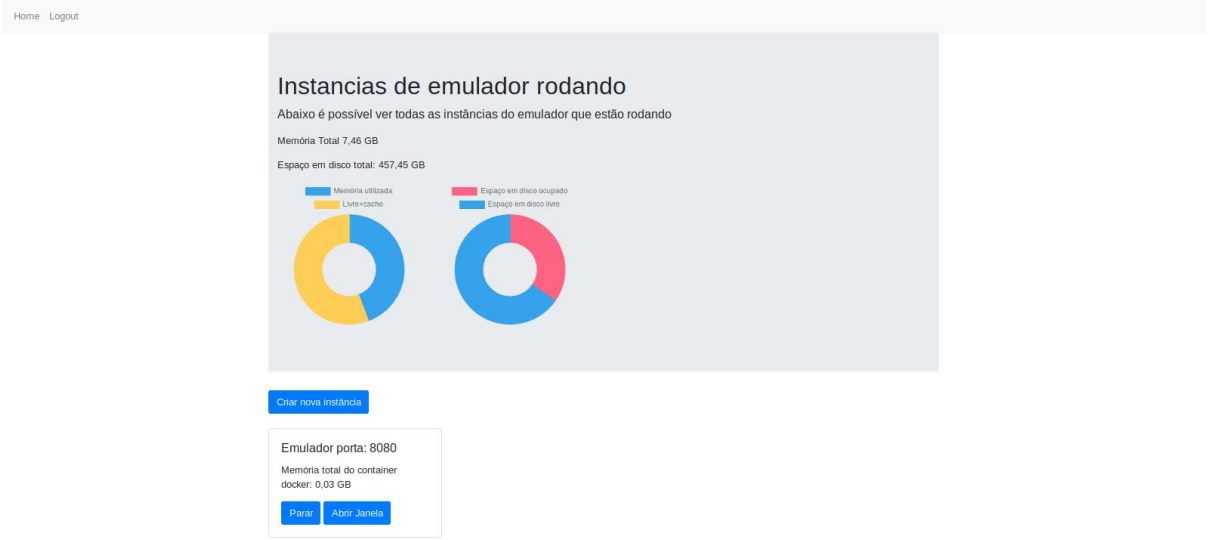

Figura 35. Página inicial do emulador. (AUTOR, 2019)

Quando é feito uma requisição GET para a URL “/”, o arquivo de urls do Django redireciona essa requisição para a execução da função *index* no arquivo de *views* da aplicação. Essa função, antes de redirecionar a página para essa tela, gera os dados para os gráficos, utilizando as bibliotecas do python, shutil e subprocess, que permitem a execução de comandos do sistema operacional. Esses dados gerados são postos em um dicionário e enviados a um template, que os consome e gera os gráficos utilizando a biblioteca javascript ChartJS. Além desses dados, também são buscados os *containers* que estão rodando,

utilizando a API Docker para o python, esses emuladores que estão rodando também são enviados para o template e consumidos por este para mostrar a lista dos emuladores rodando.

UC01 - UC02 - Instanciação e parada de emuladores.

Como mostrado anteriormente, é possível instanciar emuladores através da tela inicial, o botão “Criar nova instância” realiza uma requisição GET que redireciona no django para a função *instantiate*. Dentro dessa função, utilizando a API do Docker, tenta-se criar um container com uma porta (do host) 8080, se não for possível é feita uma tentativa com a porta 8081, se não conseguir novamente, tenta com a 8082, e assim por diante, até conseguir criar uma instância do emulador. Se a criação desse emulador for ultrapassar a quantidade de memória máxima estipulada para ser utilizada, é retornada uma mensagem de erro, caso contrário é feito um redirecionamento para a página inicial, onde o container criado poderá ser visualizado.

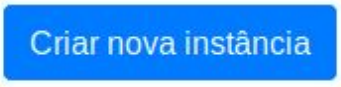

Figura 36. Botão de criar nova instância. (AUTOR, 2019)

Figura 37. Instância criada. (AUTOR, 2019)

Dentro do *card* de cada emulador existem os botões para parar o emulador, e um para abrir uma janela redimensionada para o emulador.

Dentro desse card existe um *input* escondido cujo atributo *value* é o id do container e, ao clicar no botão parar, uma requisição POST é disparada e o Django executa a função stop,

que através da API do docker pega o *container* que tem o id que foi enviado através do POST, e termina a execução deste *container* e redireciona para a tela inicial novamente.

```
def stop(request):
    client= docker.from_env()
    container = request.POST.get("emulator")
    client.containers.get(container).stop()
    return redirect("/")
```

Figura 38. Função para parar instância. (AUTOR, 2019)

O botão de abrir janela simplesmente executa um javascript que abre uma janela, já redimensionada, redirecionando essa janela para o container apropriado.

Figura 39. Janela aberta já redimensionada. (AUTOR, 2019)

UC01 - Url para acesso a emuladores.

Para facilitar o acesso dos usuários aos emuladores disponíveis, foi implementado também um serviço que redireciona para um emulador que esteja livre, e se não houver nenhum emulador livre, é instanciado um novo.

A função chamada opera de forma parecida com a função de criar uma instância. Também com o uso da API do Docker, a função vai tentando criar um container em alguma porta, partindo da porta 8080, e incrementando de um em um. Toda vez que não é possível criar um container com uma determinada porta, é verificado se o container que foi encontrado

com a tal porta está sendo acessado. Essa verificação é feita utilizando a biblioteca *subprocess* do python, que chama o utilitário netstat para checar conexões na porta. Uma vez que um *container* foi encontrado ou instanciado, a função redireciona o navegador para o *container* na porta.

4.2.3 Testes

Nesta seção são apresentados testes como forma de verificar e validar as funcionalidades, tanto do emulador, quanto a da página de administração das instâncias. Foram feitos testes de carga, relativos ao emulador, testes unitários, feitos nas URLs do ambiente de administração, e testes de aceitação, onde o sistema foi avaliado como um todo, baseando-se nos requisitos funcionais.

4.2.3.1 Testes de carga

Os testes de carga foram realizados rodando emuladores em conjunto com o ambiente web rodando, com o seguinte objetivo:

- Avaliar se é possível rodar várias instâncias de emulador, sem que essas instâncias ocupem todos os recursos do servidor, permitindo testadas até 15 instâncias rodando simultaneamente. **[RF1, RF2, RNF5]**

Este teste foi rodado em um computador com o sistema operacional Linux Mint[19], versão 19. Esse computador tem 8GB de memória e utiliza um processador intel i5-7500, além de um disco de 1TB de espaço e 7200RPM.

Além disso, o computador que estava servindo de “servidor” é diferente do computador que estava abrindo o site e aplicação. Portanto no “servidor” estava rodando apenas o ambiente web (Django web server) e os emuladores que eram instanciados pelo outro PC.

A primeira iteração de teste foi feita com 5 instâncias, como evidencia a imagem 40. A imagem 41 mostra o uso de memória dos 5 emuladores rodando. É possível ver também que o uso de CPU (1.07GB de memória, 15% a 19% de uso por núcleo de CPU) dessas cinco instâncias também é baixo.

[19] https://www.linuxmint.com/

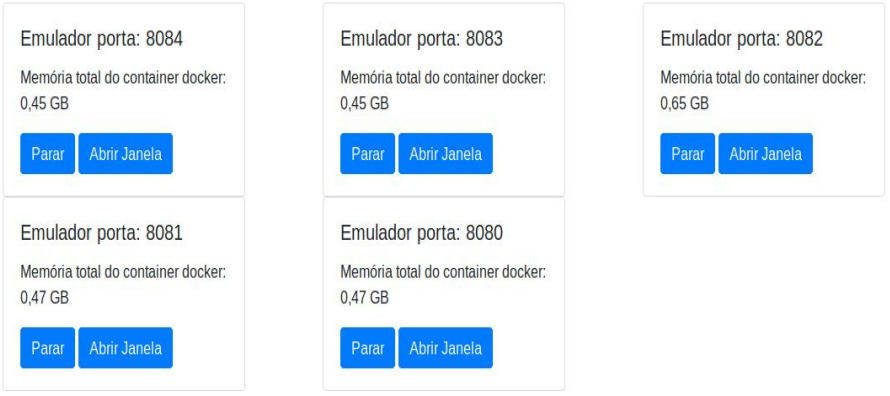

Figura 40. Cinco instâncias rodando. (AUTOR, 2019)

Figura 41. Uso de memória e CPU por cinco instâncias. (AUTOR, 2019)

Após esse teste com cinco instâncias, foi realizado um teste com 15 instâncias, como mostra a imagem 42. É possível ver na imagem 43 também o uso de memória e o de CPU (2.59GB de memória e 36% a 59% de uso por núcleo da CPU), que logicamente, são mais elevados que os do teste anterior, porém ainda estão a uma margem bem abaixo da quantidade total de memória disponível.

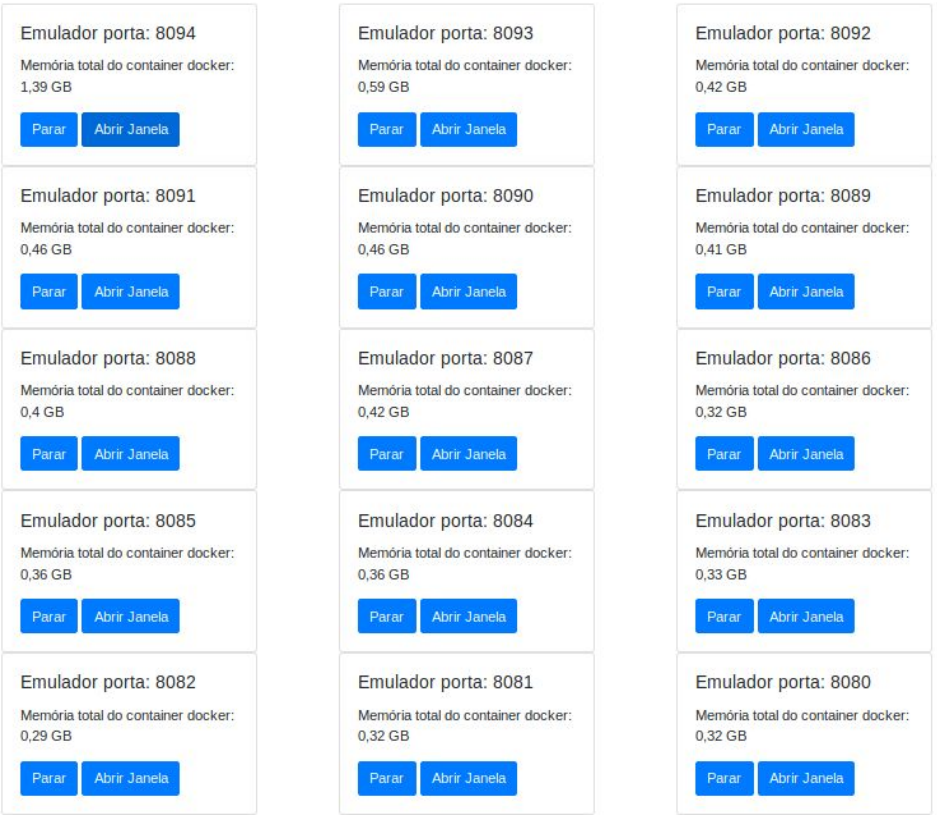

Figura 42. 15 Instâncias rodando. (AUTOR, 2019)

Figura 43. Uso de recursos de 15 instâncias rodando. (AUTOR, 2019)

Conforme foi evidenciado nesses testes de carga, a aplicação desenvolvida atende então aos requisitos funcionais RF1 e RF2, além do requisito não funcional RNF5.

4.2.3.2 Testes unitários

Nos testes unitários, foi utilizada a plataforma de testes do Django para testar os *endpoints* do ambiente web. Os testes foram feitos com o intuito de avaliar a validade e funcionalidade das URLs do ambiente web, verificando se elas servem o propósito corretamente, executando as ações que deveriam e da forma que deveriam.

Os *endpoints* testados por este teste foram:

- /, o URL index.

- /instantiate, que cria uma instância e retorna para o index.
- /stop, que termina a execução de uma determinada instância.
- /getinstance, que pega uma instância que não está sendo utilizada ou cria uma nova.

A biblioteca de testes do Django define algumas coisas que devem ser criadas, tal como uma classe que tenha herança da classe base de testes `TestCase`, e dentro dessa classe criada devem ser definidas as funções que executarão os testes.

Antes de começar os testes, foi definida uma função de `setUp` para criar um usuário no sistema e realizar o *login* deste.

```
def setUp(self):
    self.client = Client()
    User = get_user_model()
    user = User.objects.create_user("temp", "temp@gmail.com", "temporary")
    self.client.login(username="temp", password="temporary")
```

Figura 45. Função setup. (AUTOR, 2019)

Nessa função setup também é definido o *client,* este que será o agente que irá executar as requisições para os *endpoints*.

O primeiro caso de testes é o *index*. Para isso, foi definida a função `test_index_GET` para fazer uma requisição à URL “ / “, e verificar se o template e o código de *status* da requisição foram gerados corretamente. Também é verificado que não existe nenhum emulador rodando neste momento, pois se trata do primeiro teste, logo espera-se que nenhum emulador esteja rodando ainda.

```
def test_index_GET(self):
    response = self.client.get(reverse("index"))
    self.assertEquals(response.status_code, 200)
    self.assertTemplateUsed(response, "adminenv/instancias.html")
    self.assertTrue(not response.context[len(response.context)-1]["emulators"])
```

Figura 46. Função de teste da página index. (AUTOR, 2019)

No segundo caso de testes, é testada o *endpoint instantiate*, que instancia um emulador. O que é feito, primeiramente, é um teste similar com o primeiro teste (figura 46) para verificar que não existe nenhum emulador rodando ainda. Depois de verificado a não existência de emuladores, é feita uma requisição *GET* para a *endpoint* que cria o emulador. A função que é executada por essa URL, no final dela, redireciona para a página *index*, portanto dessa vez o *status* da resposta esperado é 302 ao invés de 200. No final do teste é realizado mais um teste na página *index* para verificar que agora há emuladores rodando.

```
def test_instantiate_GET(self):
    #Assert that there is no emulator running
    response = self.client.get(reverse("index"))
    self.assertEquals(response.status_code, 200)
    self.assertTemplateUsed(response, "adminenv/instancias.html")
    self.assertTrue(not response.context[len(response.context)-1]["emulators"])
    #Assert that an emulator started running
    response = self.client.get(reverse("instantiate"))
    self.assertEquals(response.status_code, 302)
    response = self.client.get(reverse("index"))
    self.assertFalse(not response.context[len(response.context)-1]["emulators"])
```

Figura 47. Teste *instantiate*. (AUTOR, 2019)

O terceiro teste é um *POST* para o *endpoint stop.* Esse teste começa indo para a *index* novamente, e verifica que existe um emulador rodando (criado no teste anterior), e depois, é guardado o ID desse container que está rodando. Após essas operações, é feita uma requisição *POST* passando como parâmetro o id do emulador. No final de tudo, é feito mais um GET para o *index,* para verificar que realmente não existe mais nenhum emulador rodando.

```
def test_stop_POST(self):
    #Assert that there is an emulator running due to the previous test
    response = self.client.get(reverse("index"))
    self.assertEquals(response.status_code, 200)
    self.assertFalse(not response.context[len(response.context)-1]["emulators"])

    #Getting the container's id and executing the stop endpoint
    emulators = response.context[len(response.context)-1]["emulators"]
    em_id = emulators[0][0][0].id
    response = self.client.post(reverse("stop"), {
        "emulator": em_id
    })
    self.assertEquals(response.status_code, 302)

    response = self.client.get(reverse("index"))
    self.assertEquals(response.status_code, 200)
    self.assertTrue(not response.context[len(response.context)-1]["emulators"])
```

Figura 48. Teste POST *stop.* (AUTOR, 2019)

O último teste foi feito para verificar que quando o *endpoint* “/getinstance”, quando é chamado, ou redireciona para um emulador não utilizado, ou cria um emulador para redirecionar.

Esse teste começa instanciando um emulador, pois os anteriores foram todos parados, e testando na página *index* que o emulador realmente está lá. Ele guarda o ID desse

emulador instanciado e executa um GET no “*getinstance*”, e verifica que o *status* da resposta HTTP é de redirecionamento (302) e que a URL para qual ele foi redirecionado é a url com a porta 8080 (porta do primeiro emulador criado.). Em seguida, é parada a execução desse emulador, de forma similar ao que foi feito no teste do *stop* (figura 48), e é enviado mais uma requisição GET para o “*getinstance”*, dessa forma, agora que o emulador foi parado, a função que será executada irá criar um emulador na porta 8080 e redirecionar o usuário para lá. Depois é feito uma verificação na *index* para mostrar que existe um emulador rodando. No fim os emuladores são parados, apenas para terminar a execução dos emuladores que foram executados.

```
def test_getinstance_GET(self):
    # Instantiate an emulator and get its id
    self.client.get(reverse("instantiate"))
    response = self.client.get(reverse("index"))
    self.assertEquals(response.status_code, 200)
    self.assertFalse(not response.context[len(response.context)-1]["emulators"])
    emulators = response.context[len(response.context)-1]["emulators"]
    em_id = emulators[0][0][0].id

    # This emulator will be in port 8080, so as no one is using it
    # getinstance should redirect to hostname:8080
    response = self.client.get(reverse("get_instance"))
    self.assertEquals(response.status_code, 302)
    self.assertEquals(response.url, "http://localhost:8080")
```

Figura 49. Primeira parte do teste *getinstance*. redirecionamento apenas. (AUTOR, 2019)

```
# if i stop the emulator, and run the getinstance again
# it'll create an emulator at port 8080 and redirect to it.
self.client.post(reverse("stop"), {
    "emulator": em_id
})
response = self.client.get(reverse("index"))
self.assertEquals(response.status_code, 200)
self.assertTrue(not response.context[len(response.context)-1]["emulators"])

response = self.client.get(reverse("get_instance"))
self.assertEquals(response.status_code, 302)
self.assertEquals(response.url, "http://localhost:8080")

response = self.client.get(reverse("index"))
self.assertEquals(response.status_code, 200)
self.assertFalse(not response.context[len(response.context)-1]["emulators"])
emulators = response.context[len(response.context)-1]["emulators"]
em_id = emulators[0][0][0].id

# final stop just to remove the running emulator
self.client.post(reverse("stop"), {
    "emulator": em_id
})
response = self.client.get(reverse("index"))
self.assertEquals(response.status_code, 200)
self.assertTrue(not response.context[len(response.context)-1]["emulators"])
```

Figura 50. Segunda parte do teste, finaliza e requisitar "*getinstance".* (AUTOR, 2019)

Ao rodar todos esses testes, é verificado que as funcionalidades do emulador fazem o que deveriam, todos os testes rodam sem problemas, e retornam os resultados esperados, conforme mostra a figura 51.

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
----------------------------------------------------------------------
Ran 4 tests in 56.089s

OK
Destroying test database for alias 'default'...
```

Figura 51. Resultado dos testes rodados. (AUTOR, 2019)

4.2.3.3 Testes de aceitação

Todos os testes de aceitação foram feitos com um fornecedor de requisitos. Cada um dos casos de uso e requisitos funcionais foram testados pelo fornecedor de requisitos. Foram realizados três ciclos e nesses ciclos foram encontrados os seguintes bugs:

- Quando abria o companion o emulador mostrava um aviso de erro na *System UI*
- Ao clicar em um botão do painel lateral do emulador, o mesmo reiniciava.

O primeiro problema foi resolvido simplesmente subindo o nível de API da AVD de 25 para 28. Já o segundo, a solução permanente foi desabilitar o painel lateral do emulador, já que não é uma parte essencial do emulador, e apresenta problemas.

.

Realização dos Testes de Aceitação

Para realização dos testes de aceitação foi utilizado como exemplo de app, o Jogo do Mosquito desenvolvido pela Computação na Escola para o ensino de programação:

- Página do aplicativo
  - http://www.computacaonaescola.ufsc.br/?page_id=1474.
- Código fonte
  - https://drive.google.com/file/d/1Xt8Mb0dPtz3PqvSpKd_4a43ybVga3uC0/view?usp=sharing

Na sequência, o atendimento de cada um dos requisitos funcionais é apresentado.

RF - 1 Ambiente que disponibiliza emuladores

A execução do emulador, pode começar pelo ambiente que permite a instanciação e execução dos emuladores, além de possibilitar a parada dos mesmos. E também é possível acessar um emulador através da URL “/getinstance”.

Figura 52. Tela inicial do ambiente web. (AUTOR, 2019)

RF - 2 - Instanciando e acessando o emulador.

Se utilizando o ambiente de gerenciamento, o emulador pode ser instanciado através do botão “Criar emulador” como mostra a imagem 53, e ai para acessar o emulador basta clicar no botão “Abrir janela”. Caso não esteja utilizando o ambiente, a URL “/getinstance” faz o serviço completo (Imagem 54).

Figura 53. Emulador Rodando. (AUTOR, 2019)

Figura 54. Pegando instância através da URL.(AUTOR, 2019)

RF - 3 - *Containers* com Dispositivos virtuais

Utilizando qualquer um dos métodos anteriores então, obtém-se o emulador.

Os emuladores, como discutido no capítulo 5, são AVDs dentro de *containers* Docker, e são disponibilizados na web por meio da tecnologia noVNC. O emulador já é inicializado com a aplicação do *AI2 Companion* do CnE já aberta, para facilitar a execução, como evidencia a figura 55.

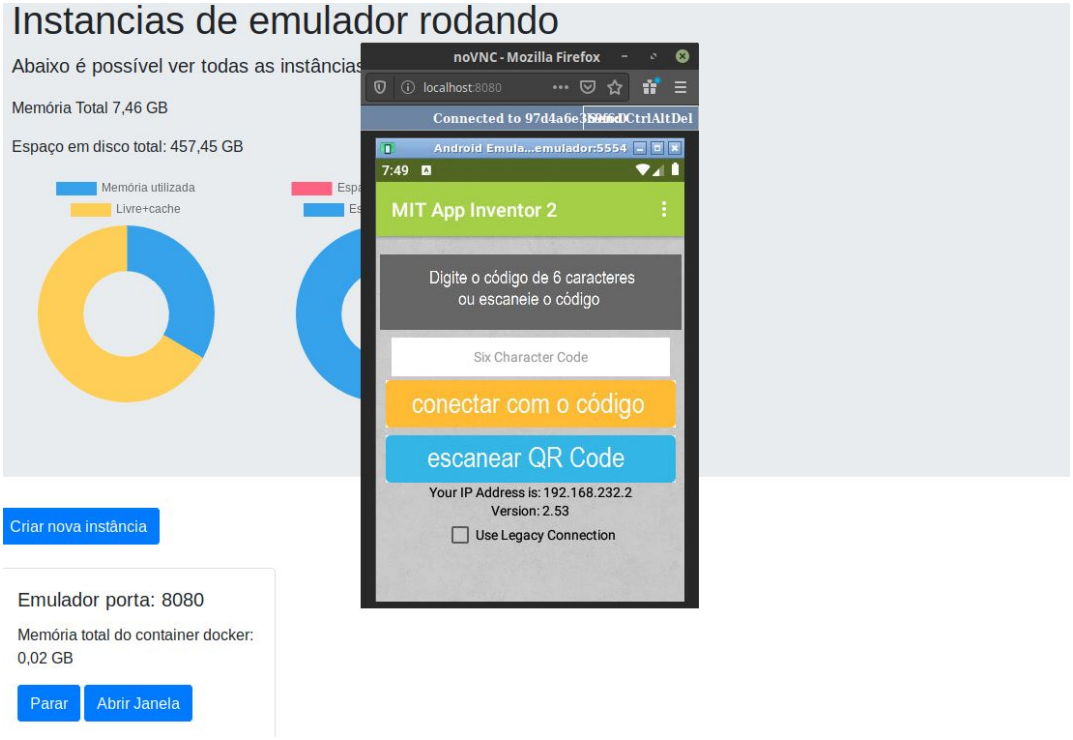

Figura 55. Emulador aberto. (AUTOR, 2019)

RF - 4 - Conexão App Inventor ↔ MIT Companion

Utilizando qualquer um dos métodos descritos acima, e obtendo o emulador, o próximo passo é executar no emulador uma aplicação desenvolvida no *App Inventor*.

Como as figura 56 e 57 mostram, basta abrir a janela do App inventor, escolher o projeto, clicar em conectar e escolher a primeira opção “Assistente AI”. Uma janela irá aparecer mostrando um código, basta digitar esse código no emulador, clicar em conectar, que a aplicação vai rodar na instância.

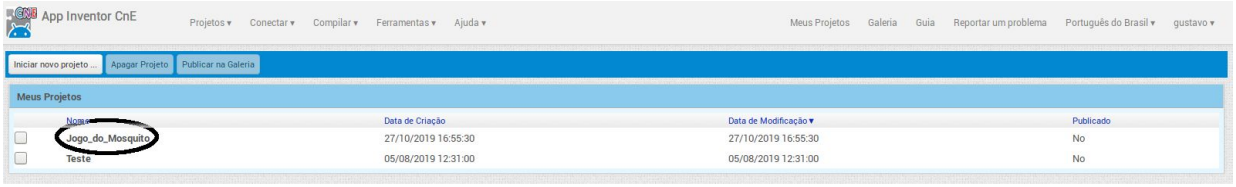

Figura 56. Aba de projetos do *App Inventor* CnE

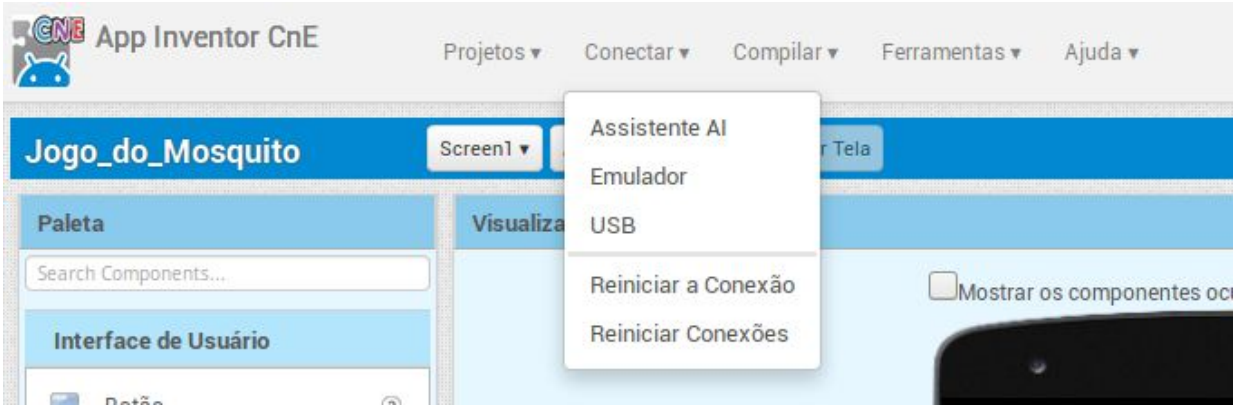

Figura 57. Conectar ao emulador. (AUTOR, 2019)

Figura 58. Tela do assistente AI. (AUTOR, 2019)

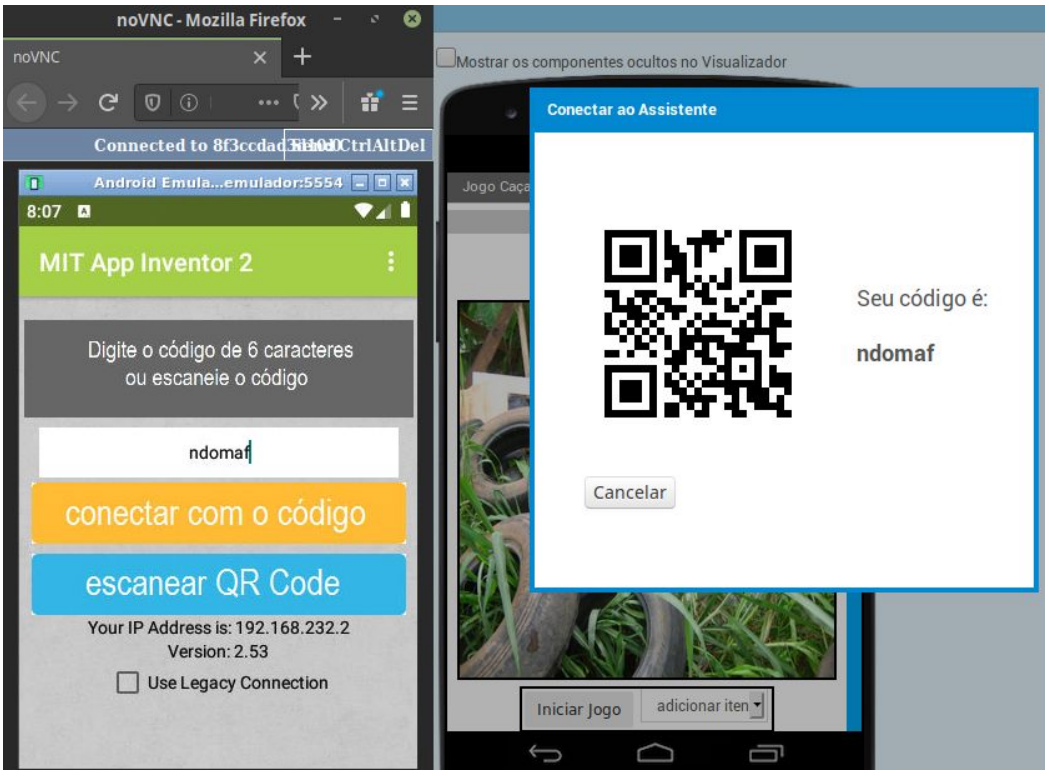

Figura 59. Código para conexão *Companion* <-> *App Inventor.* (AUTOR, 2019)

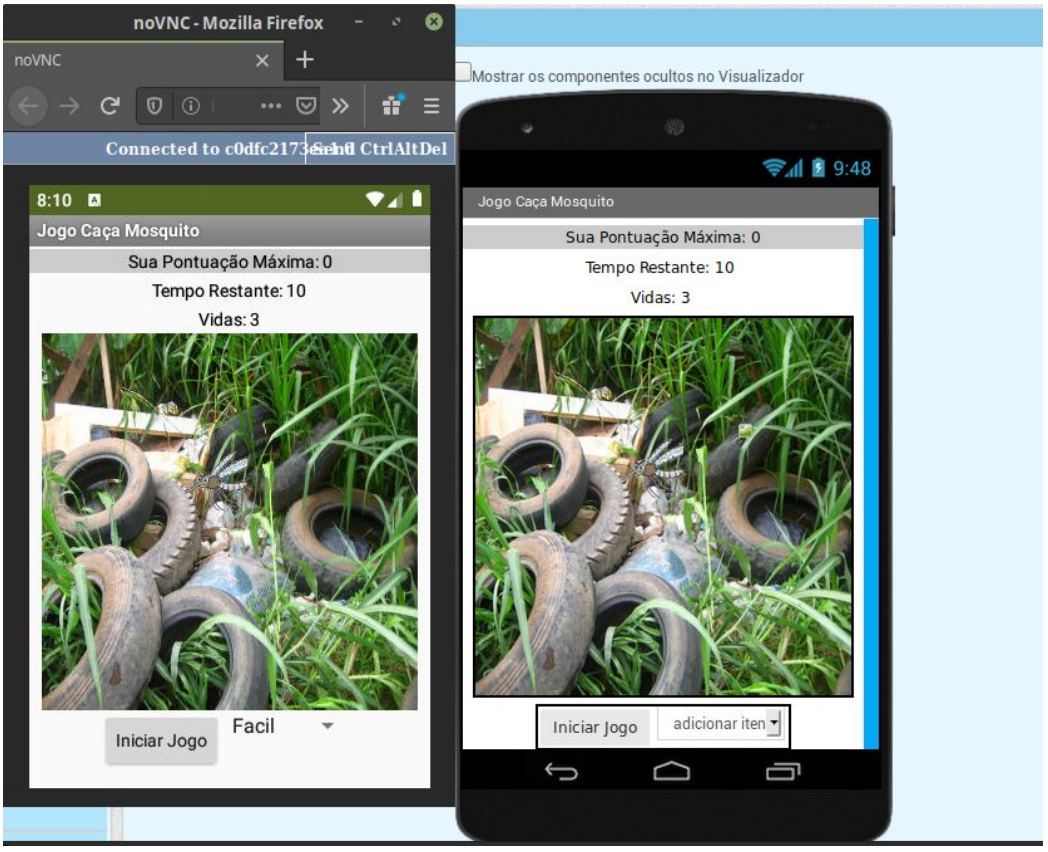

Figura 60. Emulador conectado ao *App Inventor.* (AUTOR, 2019)

RF - 5 - Módulo de administração de containers e RF - 6 - Gerenciamento de Recursos.

As imagens 52, 53 e 55 já demonstram a funcionalidade destes requisitos.

5. Resultados Obtidos

Conforme visto nas seções de testes, o conjunto de ferramentas implementado atende aos requisitos propostos, tanto do ponto de vista de desenvolvedor, quanto do ponto de vista do usuário.

Assim, nesta seção, é realizada uma comparação do ambiente implementado (**Android Web Emulator**) com as ferramentas identificadas no estado da arte, tomando por base os mesmos critérios utilizados na análise do estado da arte. A tabela 12 mostra a mesma comparação feita na seção 3.4, porém agora é adicionada a aplicação desenvolvida neste trabalho com o intuito de compará-la às aplicações encontradas durante o levantamento do estado da arte.

	APK Online	**Appetize IO**	**Run That App**	**Genymotion**	**Android Web Emulator**
Gratuito	Sim	Não	Não	Não	Sim.
Tipo de Solução	*Android Virtual Device*	WebSocket com ADB HTML5 e JS	WebSocket com ADB HTML5 e JS	*Android Virtual Device*	*Android Virtual Device*
Upload de arquivos	Upload para servidor do host e Upload para AVD persistente	Upload para servidor do host(roda no emulador)	Upload para servidor do host(roda no emulador)	Upload para AVD persistente (*Drag and drop*)	Upload para AVD persistente (*Drag and drop*)
Ferramentas para controlar múltiplas instâncias	não.	não.	não.	Sim.	Sim.
Compatibilidade com Navegadores (Google Chrome e Mozilla Firefox)	Total, porém roda melhor em Google Chrome	Total	Total	Total	Total
Integração App Inventor	Não	Não	Não	Não	Sim.
Gestão de uso de	Não.	Não.	Não.	Não.	Sim.

memória					

Tabela 12. Comparação das ferramentas

Conforme a comparação realizada na seção 3.4, foi concluído que nenhuma das soluções existentes atendiam completamente os critérios definidos. Sendo por ser software licenciado ou por falta de alguma *feature*, nenhuma das apĺicações eram adequadas. A solução desenvolvida neste, no entanto, trabalho atende aos requisitos que foram estipulados para o problema.

Tirando a solução *Android Web Emulator* (que é o software desenvolvido), das outras aplicações, o GenyMotion parecia ser o mais promissor. Não fosse o fato de ser um software licenciado, ele bastaria como solução para o problema apresentado no capítulo 1.

Das outras soluções, todas elas não disponibilizam um ambiente que permita o gerenciamento das instâncias do emulador, por exemplo. E o mais importante, nenhuma delas apresenta algum tipo de integração com o App Inventor, que pode ser considerado uma das motivações para o desenvolvimento deste trabalho.

Portanto, é possível ver que a aplicação desenvolvida, preenche o vazio que as demais aplicações, encontradas durante o levantamento do estado da arte, não pareciam preencher, fazendo assim, nenhuma dessas aplicações que foram encontradas, adequadas para solucionar completamente o problema que foi apresentado no capítulo 1.

5.2 Ameaças à validade

Como em toda pesquisa, algumas ameaças à validade das conclusões podem ser elencadas. A seguir serão listadas as possíveis ameaças à validade dos desenvolvimentos do presente trabalho.

- Não houve aplicação em ambiente real. Devido a limitações de disponibilidade de ambientes de aplicação de ensino de computação da Computação na Escola no momento da finalização deste trabalho, não foi possível aplicar o sistema desenvolvido com alunos em sala de aula;
- A comparação com outras ferramentas foi realizada pelo autor do trabalho, que também implementou a ferramenta. Esse fato pode comprometer a isenção das avaliações;
- Por não ter tido aplicação em ambiente real, não foram realizados de usabilidade, o que pode comprometer a validade da solução.

6. Conclusão

Neste trabalho é apresentado um ambiente web de emulação Android. Esse ambiente tem como objetivo facilitar os testes de aplicações desenvolvidas com a ferramenta *App Inventor,* a fim de auxiliar o ensino de computação nos projetos desenvolvidos pela Iniciativa Computação na Escola. A aplicação completa consiste de um módulo de gerenciamento das instâncias do emulador por meio de um ambiente web, e dos emuladores em si, que também podem ser acessados via web quando iniciados. As instâncias de emuladores foram desenvolvidas utilizando *Android Virtual Devices* e *containers* Docker, e o ambiente de gerenciamento foi desenvolvido com Python e o *framework* de desenvolvimento de aplicações web Django.

Antes de iniciar o desenvolvimento foram realizados estudos de temas que estão relacionados com os principais conceitos envolvidos no desenvolvimento da solução. Temas como acesso remoto, emuladores Android, e desenvolvimento de aplicações web fazem parte desse estudo.

A fim de encontrar quais aplicações atualmente ofereciam um ambiente web com acesso a um (ou mais) emuladores Android, foi realizada uma busca pelos estado da arte de emuladores Android online. Com as soluções encontradas foi feita uma comparação considerando diversos critérios como:

- Se a aplicação é uma aplicação gratuita
- O tipo de tecnologia utilizada para disponibilizar os emuladores.
- A possibilidade de realizar upload de arquivos externos e rodá-los no emulador.
- Se existe uma ferramenta para controlar as múltiplas instâncias do emulador.
- A compatibilidade com navegadores.
- Integração com *App Inventor*
- Gestão do uso de memória.

Realizando a comparação das soluções encontradas foi possível verificar que nenhuma das aplicações disponíveis atendia completamente a todos os critérios, portanto o software desenvolvido por este trabalho foi modelado a partir destes aspectos e dos requisitos elencados, com o objetivo de resolver completamente os problemas apresentados.

Após o estudo da fundamentação teórica, o levantamento do estado da arte e a modelagem da solução, foi dado início ao desenvolvimento da aplicação que atendesse totalmente aos requisitos levantados. O ambiente web para gerenciar as instâncias do

emulador foi desenvolvido utilizando Python e Django, enquanto os emuladores em si, foram utilizadas as tecnologias de *Android Virtual Devices* e dos *containers* Docker.

Dado o final do desenvolvimento da aplicação, foram testadas as funcionalidades da aplicação, tanto do ambiente web, realizando testes unitários e de aceitação, quanto ao emulador, foram realizados testes de carga e também de aceitação.

Realizados os testes, o ambiente desenvolvido foi realizada então uma comparação com as aplicações encontradas no levantamento do estado da arte, a partir da qual encontram-se evidências iniciais de que a aplicação desenvolvida durante este trabalho, atende a todos os requisitos e aspectos aos quais as aplicações do estado da arte estavam sendo comparadas. Vale também apontar que conforme descrito no capítulo 5, existem algumas configurações que podem ser aplicadas nas imagens do emulador. Entre essas configurações, estão a habilitação de emulação de gps, acelerômetro e etc. Foi apontado no capítulo 2 que o emulador atual do *App Inventor* não disponibiliza tais funcionalidades. Apesar do emulador desenvolvido não disponibilizar essas funcionalidades (já que foi optado pela criação de uma imagem minimalista), habilitá-las seria apenas questão de simples configuração de uma imagem nova, e atualizar o *container* Docker.

Como trabalhos futuros, sugere-se: melhoras no desempenho dos emuladores, através de redução de necessidades de recursos; integração maior do emulador com o *App Inventor* do computação na escola; ampliações nas interfaces e funcionalidades tanto do emulador quanto do ambiente de gerenciamento; criação de um emulador autoral para substituir o uso das AVDs, o que permitiria maior customização do ambiente e etc.

7. Referências

APP INVENTOR. Disponível em: <http://appinventor.mit.edu/explore>. acessado em 03/2019.

BARBOSA, A F. Pesquisa sobre o uso da internet por crianças e adolescentes no Brasil. TIC kids online brasil 2014. Brasil, 2015.

BNCC. Base nacional Currícular Comum, Disponível em: <http://basenacionalcomum.mec.gov.br/image s/BNCC_EI_EF_110518_versaofinal_site.pdf>. Acessado em 05/2019.

CHATZINIKOLAKIS, G.; PAPADAKIS, S. Motivating K-12 students learning fundamental Computer Science concepts with App Inventor. Interactive Mobile Communication Technologies and Learning - IMCL, 2014.

CSTA, K12. Computer Science Standards K-12, Disponível em: <https://cdn.ymaws.com /www.csteachers.org/resource/resmgr/Docs/Standards/CSTA_K-12_CSS.pdf>. Acessado em 03/2019.

CUNHA, JAILSON. Ensino de Programação para alunos do Ensino Básico: Um levantamento das pesquisas realizadas no Brasil. 2017, Disponível em: <https://repositorio.ufpb.br/jspui /bitstream/123456789/3328/1/JCS14062017.pdf>.

GOMES, MARCOS CÉSAR PIRES. Os benefícios do ensino de linguagem de programação no currículo regular. Disponível em: <https://administradores.com.br/artigos/os-beneficios-do-ensino-de-linguagem-de-programacao-no-curriculo-regular>.

PNC. Parâmetro nacional currícular, Disponível em: <http://portal.mec.gov.br/seb/arquivos/pdf/introducao .pdf>.Acessado em 05/2019

SEBRAE. 4 habilidades desenvolvidas pelo pensamento computacional, Disponível em: <http://cer. sebrae.com.br/4-habilidades-desenvolvidas-pelo-pensamento-computacional/>.

TANCICLEIDE C. S. GOMES, JEANE C. B. DE MELO. App Inventor for Android: Uma Nova Possibilidade para o Ensino de Lógica de Programação. CBIE, 2013.

VICTOR, M. DEL BARRIO. Study of the techniques for emulation programming. FIB UPC, 2001

WOLBER, D. App inventor and real-world motivation. Proceedings of the 42nd ACM Technical Symposium on Computer Science Education - SIGCSE ‘11, 2011.

WOLBER, D. App Inventor: Create Your Own Android Apps. Sebastopol (CA), USA: O’Reilly, 2011.

Apêndice I - Artigo

Ambiente web integrado com App Inventor para a execução de aplicações Android

Gustavo Borges França[1]

[1]Departamento de Informática e Estatística (INE) Universidade Federal de Santa Catarina(UFSC) – Florianópolis, SC – Brazil

gustavo.borgesfr@gmail.com

Abstract. *With the intense presence of technology in the everyday life, the understanding of computer science in concepts is becoming very important. Such concepts are usually taught only in college years. With that in mind, some projects have been created to bring this knowledge to the elementary and middle school. In that environment, the software App Inventor is used to teach programming to children, and in the process of learning how to develop applications, problems like the lack of a cellphone, or poor wifi connection, when the student need to test and execute the developed application interrupts the learning process. To solve these problems this work shows the development of an web based android emulator that can be used to test these applications.*

Resumo. *Com a crescente presença da tecnologia no cotidiano, conhecimentos de conceitos atrelados à ciências da computação estão se tornando muito importantes. Tais conceitos são ensinados apenas no ensino superior. Portanto, alguns projetos foram criados para levar esses conceitos para os ensinos fundamental e médio. Nestes ambientes, a software App Inventor é utilizado para ensinar programação, e no processo de aprendizagem, problemas como a falta de celulares, wifi de baixa qualidade, quando o aluno precisa testar e executar o aplicativo desenvolvido, interrompe o processo de aprendizagem. Como solução, este trabalho apresenta um software que permite o acesso à um emulador através de um navegador.*

1. Introdução

O ensino de programação favorece o desenvolvimento do raciocínio lógico, da capacidade de abstração, além de noções de causa e efeito (quando faço "isto", acontece "aquilo") e também capacidade de concentração e resolução de problemas (GOMES, 2015).

Na Educação Básica, a Computação, quando presente, tem sido apenas de forma instrumental, envolvendo assuntos como edição de textos, planilhas, e não o desenvolvimento do pensamento computacional, que é o que realmente desenvolve e possivelmente atrai jovens para a área, ou muitas vezes trata a disciplina como um diferencial mercadológico (BARBOSA, 2015).

Neste contexto, existem projetos que levam a computação para os ensinos fundamental e médio, como o Computação na Escola. Uma das ferramentas que é utilizada nesses projetos para o ensino de programação nestes projetos é o *App Inventor*,

software desenvolvido pelo MIT para desenvolvimento de aplicativos para a plataforma Android.

Durante o processo de ensino de programação, chega-se em uma etapa onde é necessário testar e executar o aplicativo que foi desenvolvido, no caso de aplicativos desenvolvidos com o *App Inventor* existem algumas formas de se fazer isso. Uma delas é com o uso de um celular, que ou está conectado via USB com o computador. A outra utilizando outra ferramenta do MIT chamada AI2 Companion em conjunto com o celular.

Essas alternativas levantam dois grandes problemas, caso o aluno não tenha um celular disponível, ou a rede wifi da escola for instável/inexistente, ambas as alternativas são impraticáveis.

Como solução para tais problemas, este trabalho apresenta o desenvolvimento de um emulador que é acessado através do navegador, com isso, não é necessário um celular, já que o aplicativo será executado no emulador, e não é necessário uma conexão de rede sem fio.

2. Conceitos importantes

O MIT *App inventor* é uma ferramenta de programação visual, de código aberto, desenvolvido com a linguagem de programação Java, criada pela Google, e hoje em dia é mantido pelo *Massachusetts Institute of Technology* (MIT). Essa ferramenta serve para desenvolver aplicações para o sistema operacional Android através de um ambiente de programação em blocos. Ela é muito útil para ensinar recém-chegados ao mundo da programação a criar aplicações para Android, e aprender sobre programação em geral. Seu ambiente possibilita o ensino de conceitos de lógica de programação de uma forma atraente e motivadora para estudantes dos ensinos básico e médio (GOMES e MELO *et al.,* 2013).

Dentro do ambiente do *App Inventor* existem algumas formas de realizar o *live testing* de uma aplicação, uma delas é por meio do aplicativo MIT AI2 *Companion*[20] que, em tempo real, através de um aparelho Android permite o *debugging* da aplicação desenvolvida.

Android Virtual Device (AVD) é um emulador de Android compacto e eficiente. Ele vem junto com o Android SDK, o qual é permite a criação de imagens personalizadas desses emuladores, alterando parâmetros como, quantidade de memória, habilitação de recursos como gps, tamanho de tela e etc. Também é possível configurar algumas dessas opções ao executar o emulador através de parâmetros na linha de comando na hora da execução do emulador.

O noVNC é um cliente de VNC (*Virtual Network Computing*) que é acessado via navegador. VNC é um sistema que permite o compartilhamento de um desktop gráfico via rede. O noVNC roda em qualquer navegador (exceto IE10) pois faz uso apenas de Javascript e HTML5, fazendo uso de *websocket* e canvas do HTML.

3. Modelagem e Desenvolvimento

[20] http://appinventor.mit.edu/explore/ai2/setup-device-wifi

O *software* foi modelado em duas grandes partes principais, um ambiente administrativo, que através deste serão feitas as operações de criação e parada das instâncias, além de permitir um monitoramento do uso de recursos destas. A segunda parte são os emuladores propriamente ditos.

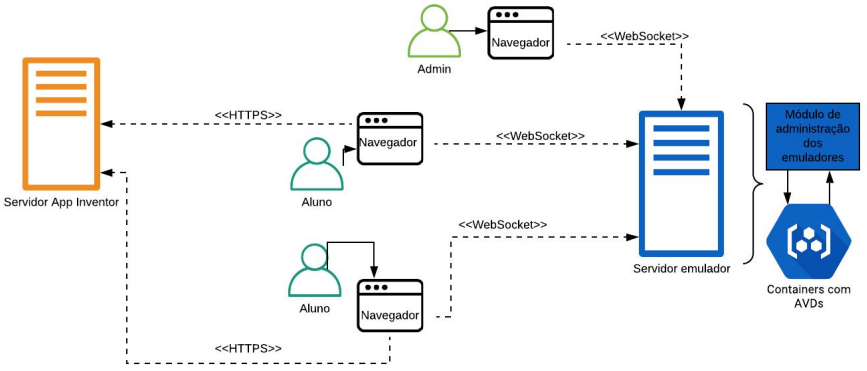

Figura 1. Arquitetura geral da solução.

Foram identificados dois casos de uso principais, um deles trata do acesso do aluno às instâncias dos emuladores, e o outro trata do gerenciamento das instâncias de emuladores, por alguma espécie de administrador, através do ambiente web.

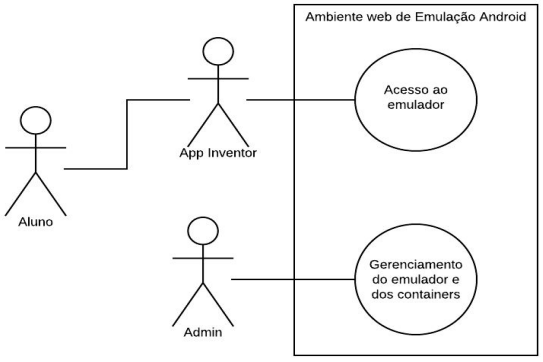

Figura 2. Diagrama de casos de uso.

Na implementação, para a parte do ambiente web, foram empregadas, principalmente, as tecnologias python em conjunto com seu framework de desenvolvimento *web* Django. E na parte dos emuladores foram utilizadas as AVDs em conjunto com *containers* Docker e noVNC.

A parte do ambiente *web*, utiliza uma API para a manipulação de *containers* Docker (Python docker API), dentro destes *containers* Docker se encontram AVDs previamente configuradas, com MIT AI2 Companion já instalados e rodando, além do noVNC, para permitir o acesso do emulador através de navegadores.

4. Testes

Para validar a solução desenvolvida, foram realizados três tipos de testes. Testes de carga, para verificar a possibilidade de executar diversos emuladores ao mesmo

tempo, sem que haja um uso excessivo de recursos do servidor. Testes unitários, para validar o ambiente web, verificando cada *endpoint* disponibilizado por este. E por fim, um teste geral de funcionalidade, neste teste o *software* desenvolvido foi utilizado com um aplicativo real, desenvolvido com o *App Inventor.*

No teste de carga foram postas para rodar 15 instâncias do emulador, além do ambiente web, em um servidor com 8GB de memória RAM, 1TB de hd e um processador i5 7500 de 4 núcleos. O resultado foi utilização de memória entre 2.5GB até 3GB e utilização de cerca de 40% de CPU.

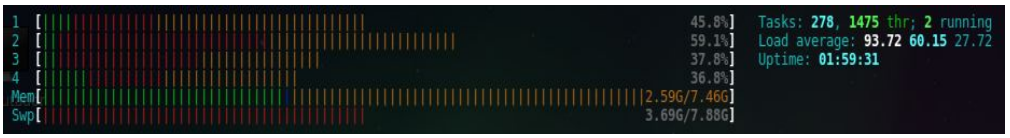

Figura 3. Resultado do teste de carga.

Já nos testes unitários, cada *endpoint* do ambiente *web* foi posto à prova, utilizando a API de testes unitários do próprio Django. O objetivo destes testes era demonstrar que o ambiente executava a função que deveria executar, que essa função funciona, e que retorna a resposta HTTP correta. Essas funções seriam criar, remover e acessar uma instância do emulador. Foi feito um teste para cada, e um teste que executa todas elas.

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
----------------------------------------------------------------------
Ran 4 tests in 56.089s

OK
Destroying test database for alias 'default'...
```

Figura 4. Resultado dos testes unitários

E por último, um teste de funcionalidade, que executa um aplicativo real, desenvolvido com o *App Inventor.* Para realização deste teste foi utilizado como exemplo de app, o Jogo do Mosquito desenvolvido pela Computação na Escola para o ensino de programação:

- Página do aplicativo
  - http://www.computacaonaescola.ufsc.br/?page_id=1474.
- Código fonte
  - https://drive.google.com/file/d/1Xt8Mb0dPtz3PqvSpKd_4a43ybVga3uC0/view?usp=sharing

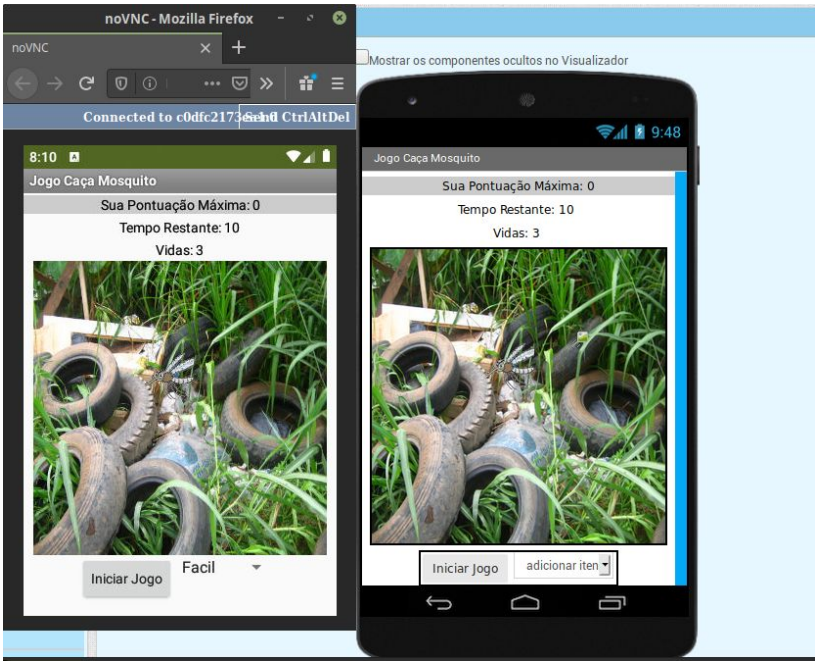

Figura 5. Emulador (direita) rodando aplicação desenvolvida no *App Inventor* (esquerda)

5. Conclusão

Neste trabalho é apresentado o desenvolvimento de um ambiente web integrado ao *App Inventor* para execução de aplicativos Android. O objetivo principal do presente trabalho foi o desenvolvimento de um ambiente web que permitisse a execução de aplicativos desenvolvidos com o *App Inventor* para a plataforma Android.

Neste contexto, foi realizado um estudo da fundamentação teórica de conceitos correlatos a este trabalho, além de um levantamento do estado da arte de ambientes que permitem execução de aplicativos Android pelo navegador.

O *software* desenvolvido não foi testado em ambiente real, porém foram realizados testes que validam suas funcionalidades, e demonstram que seu uso é perfeitamente viável em ambientes escolares como solução aos problemas levantados.

Referências

BARBOSA, A F. (2015) “Pesquisa sobre o uso da internet por crianças e adolescentes no Brasil”, TIC kids online brasil 2014.

Chatzinikolakis, G.; Papadakis, S. (2014) “Motivating K-12 students learning fundamental Computer Science concepts with App Inventor. Interactive Mobile Communication Technologies and Learning”, IMCL

CSTA, K12. (2017), “Computer Science Standards K-12” Disponível em: <https://cdn.ymaws.com/www.csteachers.org/resource/resmgr/Docs/Standards/CSTA_K-12_CSS.pdf>. Acessado em 03/2019.

Cunha, J. (2017), “Ensino de Programação para alunos do Ensino Básico: Um levantamento das pesquisas realizadas no Brasil”. Disponível em: <https://repositorio.ufpb.br/jspui/bitstream/123456789/3328/1/JCS14062017.pdf> Acessado em 03/2019.

Gomes, M. C. (2015), “Os benefícios do ensino de linguagem de programação no currículo regular”. Disponível em: <https://administradores.com.br/artigos/os-beneficios-do-ensino-de-linguagem-de-programacao-no-curriculo-regular>.

Tancicleide, C. S. GOMES; Jeane C.B(2013), “App Inventor for Android: Uma Nova Possibilidade para o Ensino de Lógica de Programação”. CBIE

Victor, M. DEL BARRIO (2001), ”Study of techniques for emulation programming”. FIB UPC

Wolber, D. (2011), “App inventor and real-world motivation. Proceedings of the 42nd ACM Technical Symposium on Computer Science Education”. SIGCSE ‘11

Wolber, D. (2011), “App Inventor: Create Your Own Android Apps”. Sebastopol (CA), USA: O’Reilly.

Apêndice II - Código Fonte

Todo código fonte se encontra no repositório do GitHub e pode ser acessado através do link: https://github.com/gstvob/TCC2. Além disso, é possível encontrar a imagem dos emuladores já montada no repositório DockerHub através do link: https://hub.docker.com/r/gstvob/android_emulator