Projeto Tiktak

Bárbara Aparecida de Castro Silva
Leonardo Santana Oliveira

MAC0499 - Trabalho de Formatura Supervisionado

Instituto de Matemática e Estatística
Universidade de São Paulo

Orientador: Alfredo Goldman vel Lejbman

São Paulo, 2015

Resumo

Atualmente, com tantos softwares disponíveis no mercado, um desenvolvedor precisa avaliar como os usuários do seu sistema estão utilizando as funcionalidades oferecidas, para saber onde é possível se empenhar mais e o que pode ser deixado de lado. Neste trabalho de conclusão de curso, resolvemos melhorar um projeto já iniciado, cujo objetivo é coletar e visualizar estatísticas de uso dentro de um sistema *Java*: o projeto Tiktak. Quando acoplado ao sistema a ser analisado, o Tiktak irá coletar dados relacionados à utilização da aplicação. Informações relacionadas ao uso do sistema, como as partes mais usadas e os erros que o mesmo está gerando, serão registradas e posteriormente visualizadas através de uma interface gráfica (um *dashboard*).

Palavras-chave: Tiktak, *Java*, análise.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Contexto

Hoje em dia, todos nós somos usuários de algum software. Todos os dias nos deparamos com alguma aplicação digital que tem como objetivo resolver nosso problema de uma forma automatizada ou até mesmo nos distrair, como é o caso dos milhares de jogos disponíveis no mercado. Diariamente surgem diversas ferramentas para isso, e quando nos deparamos com essas novas ferramentas o primeiro passo é aprender a usá-las. A interação do usuário com essas ferramentas é feita através de uma interface. Quanto melhor a interface, mais fácil será o aprendizado e, consequentemente, a interação.

Mas fazer uma boa interface nem sempre é tão simples. É difícil pensarmos qual a melhor forma de distribuir as informações na tela. Hoje em dia o que os bons desenvolvedores buscam, ou deveriam buscar, é a criação de uma interface intuitiva, clara, para que rapidamente o usuário encontre o que procura.

E quando a interface está pronta, como o desenvolvedor sabe se o usuário está usando o seu programa da maneira esperada? Todas as informações que estão na tela são relevantes? Ocorreram erros durante a navegação?

É muito comum utilizarmos apenas algumas partes de certos softwares. Por exemplo, ao acessarmos nossa conta bancária na internet, normalmente deixamos de lado uma grande quantidade de funcionalidades. É de interesse do desenvolvedor aprender mais sobre como os usuários reagem ao seu sistema.

1.2 O Projeto

Idealizado pelo aluno Renan Oliveira de Melo, atualmente cursando o Doutorado no IME-USP, o objetivo do Tiktak é monitorar aplicações *Java* de uma maneira simples e intuitiva ao desenvolvedor. Ele é composto por uma API (Interface de Programação de Aplicações) que coletará os dados e uma aplicação web chamade de *dashboard* que mostrará os dados graficamente.

Uma API é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas utilizar seus serviços.

No nosso caso, o usuário utilizará a API com a funcionalidade de monitorar o seu sistema, porém ele não vai precisar se preocupar com o modo como isso é feito. Ele apenas usará o serviço através de um método que mostraremos mais adiante.

1.2.1 Como Funciona

A API do Tiktak pode ser colocada em qualquer aplicação *Java* (mobile, web ou desktop). Ela mandará os dados via *Web Service* para o *dashboard* e os dados coletados poderão ser visualizados

graficamente. Além disso, se a aplicação não tiver conexão com a internet, a API poderá guardar esses dados em um arquivo *JSON* e este poderá ser importado no *dashboard*.

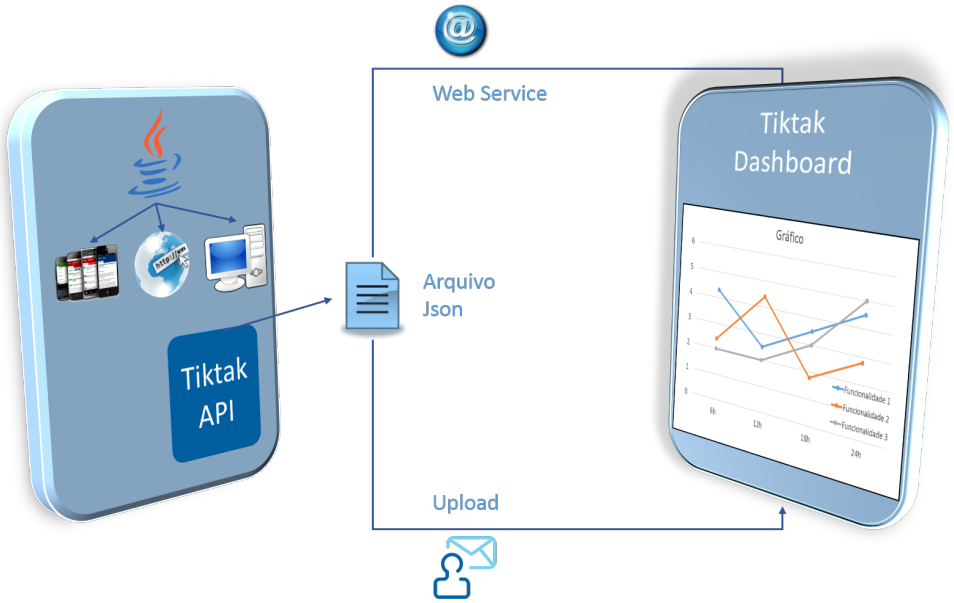

Figura 1.1: *Fluxo de dados do Tiktak.*

Por exemplo: um website importa o Tiktak para avaliar quais links estão tendo um bom número de acessos e quais não estão; um programa importa o Tiktak para avaliar o quanto suas funções estão sendo utilizadas, e descobrir quais funções remover e quais melhorar.

1.2.2 Por que *Java*?

A linguagem *Java* foi escolhida devido à sua grande popularidade nos últimos tempos (ver figuras 1.2 [IEE] e 1.3 [TBE]). É uma linguagem amplamente utilizada para aplicações web e *mobile*.

Language Rank	Types	Spectrum Ranking
1. Java		100.0
2. C		99.2
3. C++		95.5
4. Python		93.4
5. C#		92.2
6. PHP		84.6
7. Javascript		84.3
8. Ruby		78.6
9. R		74.0
10. MATLAB		72.6

Figura 1.2: *Índice do IEEE sobre as linguagens mais populares.*

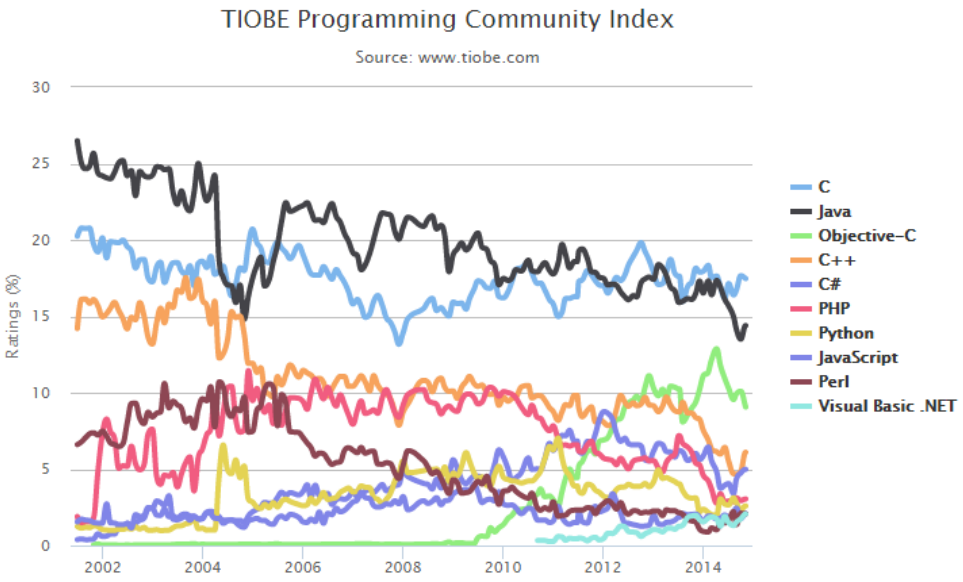

Figura 1.3: *Índice Tiobe sobre as linguagens mais populares.*

1.3 Organização do Trabalho

No Capítulo 2, detalhamos como o projeto começou e apresentamos todas as tecnologias que precisaram ser estudadas e/ou utilizadas para alcançar a implementação desejada.

No Capítulo 3, falamos sobre os testes realizados e mostramos o manual do Tiktak, além de casos de uso.

Incluímos, ainda, apreciações pessoais e críticas no Capítulo 4.

Capítulo 2

Desenvolvimento

2.1 Versão Inicial

O projeto foi implementado, inicialmente, na disciplina MAC0342 - Laboratório de Programação Extrema, ministrada em 2013 pelo professor Alfredo Goldman.

A versão inicial do Tiktak consistia na API, que coletava os dados e os armazenava em um arquivo *JSON* (*JavaScript Object Notation*), e em um *dashboard* bem simples, sem o uso do banco de dados, que recebia esse arquivo via upload do usuário e exibia os dados graficamente.

Esta versão inicial do *dashboard* foi implementanda com o uso do *framework Jmine*, uma coleção de bibliotecas prontas para uso em *Java*. No entando, o *Jmine* ainda é relativamente novo e possui pouca documentação disponível na internet, o que o torna muito complexo. Por este motivo, decidimos adaptar a versão inicial do Tiktak, removendo o *Jmine* e o substituindo por bibliotecas mais simples.

2.2 O que foi feito

Tirar o *Jmine* do *dashboard* significou começar do zero esta aplicação. Desta vez, ela foi criada utilizando tecnologias mais populares e adotamos um banco de dados para armazenar as informações de uma maneira mais organizada e segura. Uma estrutura para captar informações via *Web Service* (uma solução para a comunicação entre aplicações diferentes) também foi criada.

Esta estrutura também foi adicionada à API do Tiktak. Agora ela possui a capacidade de mandar os dados para o *dashboard* via *Web Service*. Além disso, o que já existia foi melhorado: hoje o usuário consegue configurar a API de uma maneira mais fácil e flexível. O código da API foi refatorado para melhorar a sua leitura.

2.3 Tecnologias

Como é possível notar, o desenvolvimento do projeto requer o conhecimento de muitas tecnologias. O estudo delas é de fundamental importância. A seguir nos aprofundamos em cada uma das tecnologias que foram usadas para tornar o projeto possível.

2.3.1 *Jmine*

Jmine [JMN] é um *framework*: uma biblioteca que fornece um conjunto de componentes que usamos para implementar algum aspecto de uma aplicação. Exemplos comuns são *Hibernate* para persistência de bancos de dados, *Spring* para gerenciamento de *beans*[1] e *Wicket* para interfaces web.

[1] Componentes de software utilizados para encapsular muitos objetos em um objeto único.

Jmine, por outro lado, foi construído usando esses e outros *frameworks* amplamente aceitos, proporcionando uma nova camada de abstração. Ele tornou-se uma plataforma que oferece diversos recursos baseados nestes *frameworks*. Ao utilizar a plataforma, o desenvolvedor ganha estes recursos prontos para serem usados e/ou estendidos.

Inicialmente, o *dashboard* do Tiktak foi implementado utilizando o *Jmine*, mas, por se tratar de uma plataforma complexa e ainda pouco documentada, resolvemos substituí-la por *frameworks* mais simples. Isso proporcionou mais liberdade e velocidade ao desenvolvimento do *dashboard*.

2.3.2 *MySQL*

Por se tratar de uma aplicação que pode guardar um volume grande de dados, o *dashboard* foi implementado utilizando um SGBD.

Um Sistema de Gerenciamento de Banco de Dados (SGBD) - do inglês *Data Base Management System* (DBMS) - é o conjunto de softwares responsáveis pelo gerenciamento de uma base de dados. Seu principal objetivo é retirar da aplicação cliente as responsabilidades de gerenciar o acesso, e manipular e organizar os dados.

O SGBD escolhido no desenvolvimento do *dashboard* foi o *MySQL*.

O *MySQL* [MSL] é um SGBD que utiliza a linguagem *SQL* (Linguagem de Consulta Estruturada, do inglês *Structured Query Language*) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo, atrás apenas do *Oracle* [DBE].

O *dashboard* do Tiktak utiliza o *MySQL* para guardar os dados coletados.

2.3.3 *Java EE*

As aplicações web de hoje em dia já possuem regras de negócio bastante complicadas. Codificar essas muitas regras já representa um grande trabalho. Além dessas regras, conhecidas como requisitos funcionais de uma aplicação, existem outros requisitos que precisam ser atingidos através da nossa infraestrutura: persistência em banco de dados, transação, acesso remoto, *Web Services*, gerenciamento de *threads*, gerenciamento de conexões HTTP, cache de objetos, gerenciamento da sessão web, balanceamento de carga, entre outros. Esses requisitos são chamados de não-funcionais.

Se fôssemos também os responsáveis por escrever código que tratasse desses outros requisitos, teríamos muito mais trabalho a fazer. Tendo isso em vista, a *Sun* criou uma série de especificações que, quando implementadas, podem ser usadas por desenvolvedores para tirar proveito e reutilizar toda essa infraestrutura já pronta. O *Java EE* (*Java Enterprise Edition*) [JEE] consiste de uma série de especificações bem detalhadas, dando uma receita de como deve ser implementado um software que faz uso de cada um desses serviços de infraestrutura.

Dentre as APIs disponibilizadas pelo *Java EE*, as utilizadas no desenvolvimento do *dashboard* foram:

- *JavaServer Pages* (*JSP*)
- *Java Servlets*
- *JavaServer Faces* (*JSF*)
- *Java Persistence API* (*JPA*)

A seguir, veremos mais detalhes dessas especificações e o motivo de serem utilizadas neste trabalho.

2.3.4 *JSP* e *Servlets*

Como o *dashboard* pode ser colocado na nuvem, para impedir que outras pessoas vejam os dados da sua aplicação, ele necessita de algum mecanismo de segurança. A forma clássica de autenticação

de usuário é feita através de uma tela de login, e é desta maneira que um usuário do *dashboard* pode se autenticar e acessar a aplicação.

Uma pesquisa rápida na internet mostra que existem várias formas de criar um mecanismo de autenticação para uma aplicação *Java*. Algumas delas possuem uma configuração bem complexa que geralmente ocorre no servidor. Para simplificar a vida do usuário, utilizamos uma forma de autenticação que não necessita de configurações adicionais no servidor. Essa autenticação é realizada dentro de um *Servlet*.

Servlet ("servidorzinho"em tradução livre) é uma classe *Java* usada para estender as funcionalidades de um servidor. Os campos usados na tela de login do *dashboard* foram criados através da tecnologia *JavaServer Pages* (*JSP*).

Esta tecnologia permite que os desenvolvedores da web e designers desenvolvam e mantenham de uma maneira fácil e rápida páginas web dinâmicas. Como parte da família da tecnologia *Java*, a tecnologia *JSP* permite o rápido desenvolvimento de aplicações baseadas na web que são independentes de plataforma. A tecnologia *JSP* separa a interface do usuário a partir de geração de conteúdo, permitindo aos designers alterar o layout geral da página sem alterar o conteúdo dinâmico subjacente. *JSP* é uma extensão da tecnologia *Java Servlets*.

2.3.5 *JSF*

Mais adiante, veremos um *framework* muito utilizado na camada de visualização do *dashboard*: o *Primefaces*. Veremos também que ele é baseado em *JSF*, por isso detalhamos bem esta tecnologia aqui.

JSF (*JavaServer Faces*) é uma tecnologia que nos permite criar aplicações *Java* para web utilizando componentes visuais pré-prontos, de forma que o desenvolvedor não se preocupe com *JavaScript* e *HTML*. Basta adicionarmos os componentes (calendários, tabelas, formulários) e eles serão renderizados e exibidos em formato HTML.

O *JSF* ainda tem a vantagem de ser uma especificação do *Java EE*, isto é, todo servidor de aplicações *Java* tem que vir com uma implementação de *JSF* e há diversas outras disponíveis.

A implementação mais famosa do *JSF* é a *Oracle Mojarra*, utilizada no *dashboard* do Tiktak.

2.3.6 *JPA*

Para facilitar a comunicação entre aplicações *Java* que seguem o modelo orientado a objetos e os SGBDs que seguem o modelo relacional, podemos utilizar ferramentas que automatizam a transição de dados entre as aplicações e os SGBDs. Essas ferramentas são conhecidas como ferramentas *ORM* (*Object Relational Mapper*).

As ferramentas *ORM* oferecem mecanismos de consultas independentes da linguagem *SQL*. Dessa forma, o acoplamento entre as aplicações e os SGBDs diminui drasticamente.

JPA (*Java Persistence API*) é uma API padrão da linguagem *Java* para persistência de dados que define um meio para ferramentas *ORM* e deve ser implementada por *frameworks* que desejem seguir tal padrão.

A principal ferramenta *ORM* para *Java* utilizada no mercado é o *Hibernate* [HBN]. O *Hibernate* é uma especificação *JPA* que foi criada com o objetivo de padronizar as ferramentas *ORM* para aplicações *Java* e consequentemente diminuir a complexidade do desenvolvimento. É a especificação utilizada para a construção e mapeamento do banco de dados do *dashboard*; isto simplificou a conexão entre as classes *Java* e o *MySQL*. Com apenas algumas anotações, as classes foram mapeadas e as tabelas criadas no banco, sem a necessidade de escrever código *SQL*, o que aumentou a produtividade.

2.3.7 *Web Container*

Uma aplicação web *Java* deve ser implantada em um *Web Container* para obter os recursos fundamentais de que ela necessita. Um *Web Container* é responsável:

- pelo envio e recebimento de mensagens HTTP
- por permitir que as aplicações sejam acessadas simultaneamente por vários usuários de uma maneira eficiente
- por permitir que as páginas de uma aplicação web sejam geradas dinamicamente

Além disso, um *Web Container* possui *JSF*, *JSP* e *Servlets* como funcionalidades.

Os dois *Web Containers* mais importantes do mercado são *Tomcat* e *Jetty*. Também podemos utilizar um servidor de aplicação *Java EE* como o *JBoss*, *Glassfish* ou *WebSphere*, pois eles possuem um *Web Container* internamente.

O *dashboard* do Tiktak utiliza o *Tomcat* como *Web Container*, tendo como foco principal o recebimento de informações (logs de um determinado sistema) via *Web Service*. Por isso, ele foi implementado como uma aplicação web, ou seja, ele pode ser colocado em um servidor na nuvem, onde irá receber os logs. Além disso, seu acesso pode ser feito através de um navegador de internet. Porém, para funcionar corretamente, ele precisa de um *Web Container*. Todos os testes realizados durante o desenvolvimento do *dashboard* utilizaram o *Tomcat*, escolhido por ser bastante utilizado e, consequentemente, ter bastante documentação disponível na internet.

2.3.8 *JSON*

JSON (*JavaScript Object Notation*) [JSN] é um formato leve de troca de dados. Para humanos, é fácil de ler e escrever. Para máquinas, é fácil de analisar e gerar. É baseado em um subconjunto da linguagem de programação *JavaScript*. *JSON* é um formato de texto que é completamente independente de linguagem, mas que usa convenções familiares aos programadores de linguagens como *C*, *C++*, *C#*, *Java*, *JavaScript*, *Perl*, *Python*, e muitas outras. Essas propriedades fazem do *JSON* uma linguagem ideal de troca de dados.

JSON é baseado em duas estruturas:

- uma coleção de pares nome/valor. Em várias linguagens, isso é interpretado como um objeto, registro, *struct*, dicionário, tabela de *hash*, lista com chaves ou *array* associativo.
- uma lista ordenada de valores. Na maioria das linguagens, isso é interpretado como um *array*, vetor, lista ou sequência.

Essas estruturas de dados são universais. Praticamente todas as linguagens de programação modernas suportam essas estruturas de uma forma ou de outra. Faz sentido que um formato de dados que é permutável com linguagens de programação também seja baseado nessas estruturas.

Os dados coletados pelo Tiktak são gravados em um arquivo *JSON*. Este arquivo é então lido pelo *dashboard* e mantido no banco de dados, para gerar os gráficos sobre uso do sistema.

2.3.9 *Web Service*

Web Service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Os *Web Services* são componentes que permitem às aplicações enviar e receber dados em formato *XML*. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato *XML*.

Esta tecnologia foi utilizada para dar ao Tiktak a possibilidade de enviar os dados coletados automaticamente ao *dashboard*, para que o usuário não precise fazer o upload do arquivo *JSON*.

2.3.10 *JAX-RS*

Para conseguir enviar os dados da API ao *dashboard* via *Web Service*, algumas tecnologias foram analisadas. Uma delas foi a *W3C* (*WSDL*, *SOAP* e *XML*), mas a complexidade de evoluir um *Web Service* que segue os padrões *W3C* é alta e por isso optamos pela arquitetura *REST* (*Representational State Transfer*). Em geral, *Web Services* que seguem os princípios *REST* são mais fáceis de implementar e evoluir.

Na plataforma *Java*, há especificações que definem o modelo de programação de *Web Services* que seguem os princípios *REST*. A principal especificação para esse tipo de *Web Service* é a *API for RESTful Web Services JAX-RS* [JAX-RS]. Ela possui bastante documentação na internet e por isso foi escolhida no desenvolvimento da estrutura de *Web Service* do projeto Tiktak.

2.3.11 *Primefaces* e *Bootstrap*

Criar o *layout* de uma aplicação web pode ser uma tarefa bem difícil. Pensando nisso, muitos *frameworks* foram criados para facilitar essa parte do desenvolvimento. A camada visual do *dashboard* (menu, gráficos, botões, etc) foi desenvolvida com ajuda de dois *frameworks*: *Bootstrap* [BSP] e *Primefaces* [PFS].

Primefaces é um *framework* baseado em *JSF* e *Bootstrap* um *framework* para *HTML*, *CSS* (*Cascading Style Sheets*) e *JS* (*JavaScript*). Ambos foram utilizados para incrementar a parte visual do *dashboard*, e escolhidos devido à sua simplicidade e grande documentação disponível na internet.

2.3.12 *Maven*

O projeto Tiktak utiliza muitos *frameworks*. Para auxiliar no download e no gerenciamento de todas essas bibliotecas, utilizamos a ferramenta *Maven* [MVN].

Maven é uma ferramenta de compreensão e gerenciamento de projetos de software. Baseando-se no conceito de um modelo de objeto de projeto (da sigla em inglês POM), o *Maven* pode gerenciar a construção de um projeto a partir de uma peça central de informação: o arquivo *pom.xml*.

Este arquivo reúne todas as dependências do projeto e de onde elas devem ser baixadas. A partir deste arquivo, o *Maven* constrói o projeto buscando todas essas dependências instantaneamente.

O *Maven* possui muitas extensões com diversas funcionalidades. Uma delas gera o arquivo .war de uma aplicação web. O arquivo .war do *dashboard* disponibilizado para download foi gerado com o auxílio do *Maven*.

2.3.13 *JUnit*

Para chegar ao total funcionamento do Tiktak, foram necessários muitos testes.

Uma técnica bastante utilizada no desenvolvimento de software é a criação de testes automatizados. Eles servem para mostrar de uma forma rápida se o seu programa (ou parte dele) está se comportando como o esperado. Para exemplificar, podemos examinar um detalhe da implementação da API. Nela, foram criados testes para verificar se determinados métodos estavam funcionando corretamente. Sem esses testes, a única forma de verificar o funcionamento desses métodos seria usar a API em outro projeto, o que consumiria mais tempo, principalmente em caso de falha.

Para a automatização dos testes da API, o *framework JUnit* [JNT] foi utilizado. O *JUnit* é muito simples e facilita muito a criação de testes. Ele possui alguns métodos que tornam seu código de teste bem legível e validações mais fáceis.

Capítulo 3

Testes e Resultados

Além do aprendizado das variadas tecnologias, para alcançar a implementação funcional do Tiktak uma grande bateria de testes foi necessária.

Muitos testes foram feitos com a ajuda do *JUnit*, como foi mencionado. Porém, para realizar a maioria dos testes principais sobre a API e o *dashboard*, foi criada uma aplicação web específica[1].

Esta aplicação web era uma página simples com três botões, cada um associado a uma funcionalidade, para que cada clique alimentasse os logs capturados.

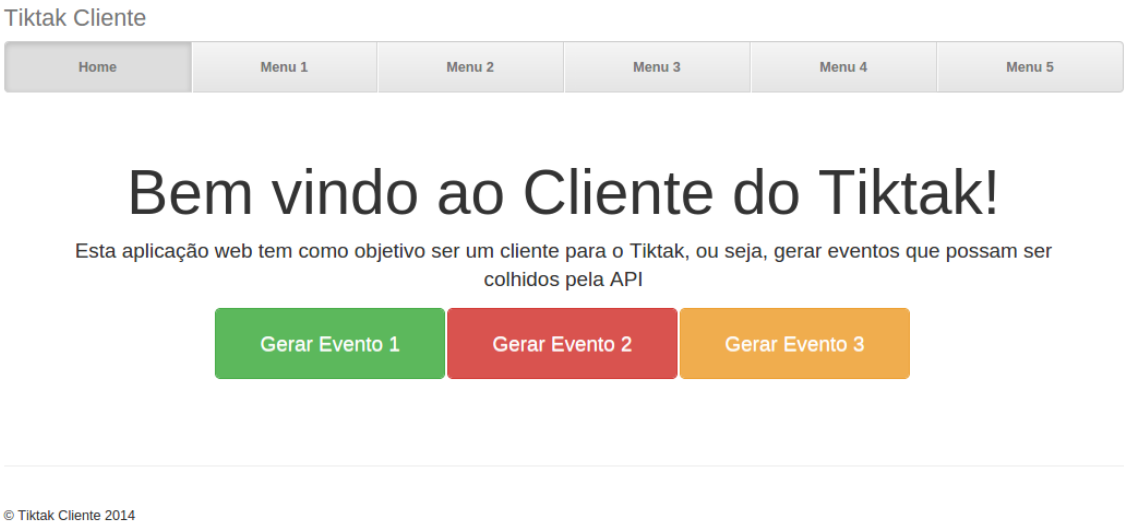

Figura 3.1: *Tela da aplicação cliente.*

Após tantos testes, o Tiktak implementado é extremamente simples e direto, permitindo ao usuário escolher se quer utilizar um *Web Service* ou fazer o upload do arquivo *JSON*.

O *dashboard* apresenta vários filtros ao usuário, de modo que ele pode ver seus dados por funcionalidades, usuário e período (data, mês ou ano).

A seguir apresentamos o manual do Tiktak; nele estão detalhados todos os passos e requisitos para que se possa utilizar a API, bem como o *dashboard*. Ao final do capítulo, apresentamos um caso de uso e algumas telas que mostram a simplicidade da aplicação.

[1] O código da aplicação pode ser visto em https://bitbucket.org/leoso/tiktak-cliente

3.1 API

3.1.1 Download

Faça o download da API [BACS] [LSO] e coloque o arquivo *tiktak-api-1.0.0.jar* no local onde estão as dependências do seu projeto.

3.1.2 Configuração

A API deve ser configurada através da passagem de algumas propriedades como parâmetro. São elas:

- modo: seu valor deve ser *json* ou *webservice*. Descreve o que o Tiktak deve fazer com os dados coletados. Para *json*, o Tiktak guardará os logs em um arquivo *JSON*. Para *webservice*, a API guardará os dados em memória para enviá-los ao *dashboard*, via *Web Service*.
- sistema: seu valor deve ser uma *string* representando o nome do sistema que está sendo monitorado
- diretorio: seu valor deve ser uma *string* representando o local onde o Tiktak deve guardar o arquivo de log (esta propriedade só faz sentido para *modo=json*).
- limite.fila: seu valor deve ser um inteiro representando a quantidade máxima de logs que podem estar guardados na fila em um instante (esta propriedade só faz sentido para *modo=webservice*). Como o *dashboard* irá consumir os dados coletados pela API com uma frequência pré-definida, ele pode demorar para coletar esses dados. Por isso, a API tem a opção de limitar a quantidade de eventos guardados; se esse limite for alcançado, os eventos mais antigos serão descartados e não haverá risco de um gasto excessivo de memória.
- limite.envio: seu valor deve ser um inteiro representando a quantidade máxima de logs que o Tiktak deve mandar de uma vez via *Web Service* (esta propriedade só faz sentido para *modo=webservice*).

Caso alguma propriedade não seja fornecida, a API usará como padrão os seguintes valores:

```
1 modo=webservice
2 sistema=Sistema
3 limite.fila=100
4 limite.envio=50
```

Se o *modo* escolhido for *json* e não houver diretório definido, o arquivo de log será guardado no local onde foi executado o comando *Java*. Para descobrir qual é esse diretório, utilize:

```
1 System.getProperty("user.dir");
```

3.1.3 Uso

Para utilizar a API em uma aplicação web via *Web Service* (modo padrão), primeiro deve-se configurar o arquivo *web.xml*, incluindo as seguintes linhas:

```
<display-name>Restful Web Application</display-name>
    <servlet>
        <servlet-name>Jersey REST Service</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</
            servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>br.org.tiktak.webservice</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Jersey REST Service</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
```

Feito isto, basta instanciar a API e utilizar esta instância quando necessário para registrar os eventos.

Exemplo 1:

Instanciando sem definir propriedades:

```
TikTak tiktak = new TikTak();
```

Registrando determinado evento:

```
tiktak.log("usuario", "evento");
```

Exemplo 2:

Instanciando com propriedades:

```
Properties properties = new Properties();
properties.setProperty("sistema", "MeuSiteDeCompras");
properties.setProperty("limite.fila", "60");
TikTak tiktak = new TikTak(properties);
```

Registrando determinado evento:

```
tiktak.log("usuario", "pagina1");
```

Neste exemplo, a API registra os eventos do MeuSiteDeCompras, trabalhando via *Web Service* e mantendo uma fila de no máximo 60 eventos em memória.

Exemplo 3:

Instanciando:

```
TikTak tiktak = new TikTak("enderecoDoArquivo/tiktak.properties");
```

Registrando determinado evento:

```
1 tiktak.log("usuario", "pagina1");
```

As propriedades podem ser definidas previamente em um arquivo. Neste exemplo, para ter as mesmas propriedades do Exemplo 2, basta que o conteúdo do arquivo *tiktak.properties* seja:

```
1 sistema=MeuSiteDeCompras
2 limite.fila=60
```

Caso seu sistema não possua conexão com a internet, uma forma de monitorá-lo é gravando os eventos em um arquivo *JSON*.

Veja um exemplo:

Instanciando:

```
1 TikTak tiktak = new TikTak("enderecoDoArquivo/tiktak.properties");
```

Registrando determinado evento:

```
1 tiktak.log("usuario", "evento");
```

Exemplo de conteúdo do arquivo de configuração:

```
1 modo=json
2 sistema=MeuSistema
3 diretorio=/home/user/
```

3.2 *Dashboard*

3.2.1 Requisitos

Há dois requisitos para o funcionamento do *dashboard*:

- um servidor: o *dashboard* é uma aplicação web e precisa de um servidor onde possa ser hospedado (por exemplo, *Tomcat* ou *Glassfish*)
- *MySQL*: é necessário ter o *MySQL* instalado no local onde o *dashboard* ficará hospedado. Por padrão, o *dashboard* vem com as seguintes configurações:

  database=tiktak-dashboard

  senha=dashboard

3.2.2 Download

Faça o download do arquivo *tiktak-dashboard-1.0.0.war* [BACS] [LSO] e hospede-o em um servidor.

3.2.3 Telas

As telas do *dashboard* são muito simples e intuitivas:

- *Web Service*: esta tela tem como funcionalidade começar e parar o consumo dos eventos registrados pela API (ver figura 3.2). Para isso, é preciso passar o endereço da aplicação web e também o intervalo de tempo que o *dashboard* deve esperar para tentar consumir os eventos.

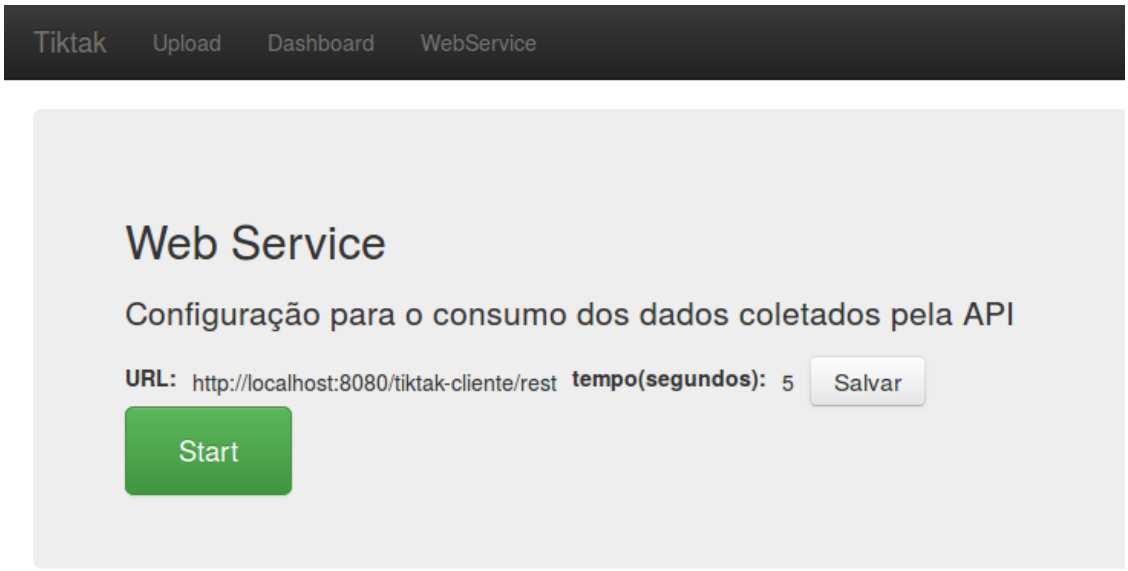

Figura 3.2: *Tela do* Web Service.

- Upload: para fazer o upload de um arquivo *JSON*, o usuário só precisa selecioná-lo e carregá-lo nesta tela (ver figura 3.3). Ao receber o arquivo, o *dashboard* guarda as informações no banco de dados.

Figura 3.3: *Tela de upload.*

- *Dashboard*: aqui os dados são exibidos graficamente, de acordo com os filtros escolhidos pelo usuário (ver figura 3.4).

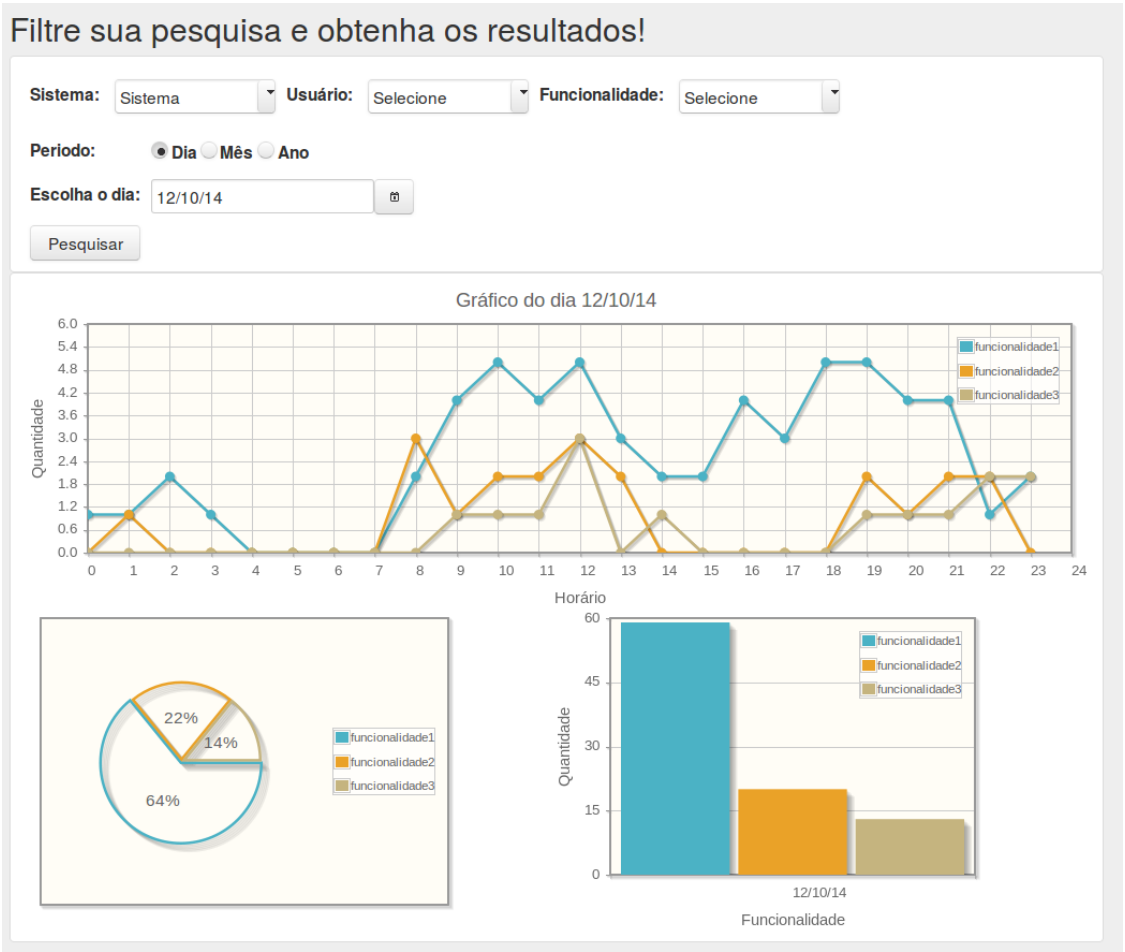

Figura 3.4: *Exemplo de dados no* dashboard.

3.3 Caso de Uso

Para testarmos a API em um caso real, criamos um site para a equipe de futebol dos alunos do IME: http://futebolimeusp.com.br/. O site foi feito em *Java* e serviu para um teste mais consistente e final da API.

A seguinte configuração foi utilizada para incluir a API no site:

```
1 Properties properties = new Properties();
2 properties.setProperty("sistema", "Futebol-Ime-USP");
3 TikTak instancia = new TikTak(properties);
```

Como vimos, esta configuração diz ao Tiktak que o nome do sistema a ser monitorado será *Futebol-Ime-USP*. Como o modo padrão é o *webservice*, não foi necessário passar essa propriedade.

Após instanciar a API, chamamos o seu método *log* em locais estratégicos para conseguir monitorar as páginas do site conforme as nossas necessidades.

3.3.1 Exemplo 1

Quando entramos na página *Contato*, a seguinte linha de código é executada:

```
1 tiktak.log("Qualquer", "Contato");
```

Dessa forma, monitoramos os acessos à esta página. Como não temos a informação de quem está acessando (o site não necessita de login), passamos como usuário uma palavra de nossa escolha (neste caso, a palavra *Qualquer*). Em um site onde um cadastro é necessário, além de monitorarmos a página acessada poderíamos registrar o autor da visita.

Figura 3.5: *Gráficos do* dashboard *ao pesquisarmos pelo sistema* Futebol-Ime-USP*, com usuário* Qualquer *e funcionalidade* Contato*, em um determinado dia.*

3.3.2 Exemplo 2

No site, há uma página dedicada ao elenco da equipe. Nela é possível clicar em cada um dos jogadores para ler mais informações sobre eles. E se quisermos saber quem é o jogador mais popular da página? Podemos utilizar o Tiktak para responder a essa pergunta. Cada vez que o perfil de um atleta é acessado, a seguinte linha de código é executada:

```
1 tiktak.log("Visitou Elenco", this.getJogador().getApelido());
```

O método *getApelido* retorna o apelido do jogador (retornado pelo método *getJogador*) cujo perfil foi acessado. Dessa forma, é possível registrar todos os acessos a perfis dos jogadores e, consequentemente, responder qual deles é o mais popular. Neste exemplo, passamos como usuário a frase *Visitou Elenco*. Isto pode ajudar a filtrar melhor a pesquisa no *dashboard* (o que será ilustrado adiante).

3.3.3 Exemplo 3

Vimos alguns exemplos de locais em que a API foi utilizada. Agora veremos como o *dashboard* recebe os dados.

O mesmo servidor do site foi utilizado para hospedar o *dashboard*. Foi necessário ligar o consumo dos dados da API na página *Web Service*.

Como vimos, na tela de *Web Service* precisamos definir de onde a aplicação deve consumir os dados e com que frequência. Como não se trata de um site muito acessado, colocamos o consumo como 60 segundos, ou seja, uma vez por minuto o *dashboard* colhe os dados registrados pela API.

Por padrão, a API envia 50 eventos por vez (propriedade *limite.envio*), ou seja, por minuto o *dashboard* irá buscar pelos 50 eventos mais antigos registrados pela API.

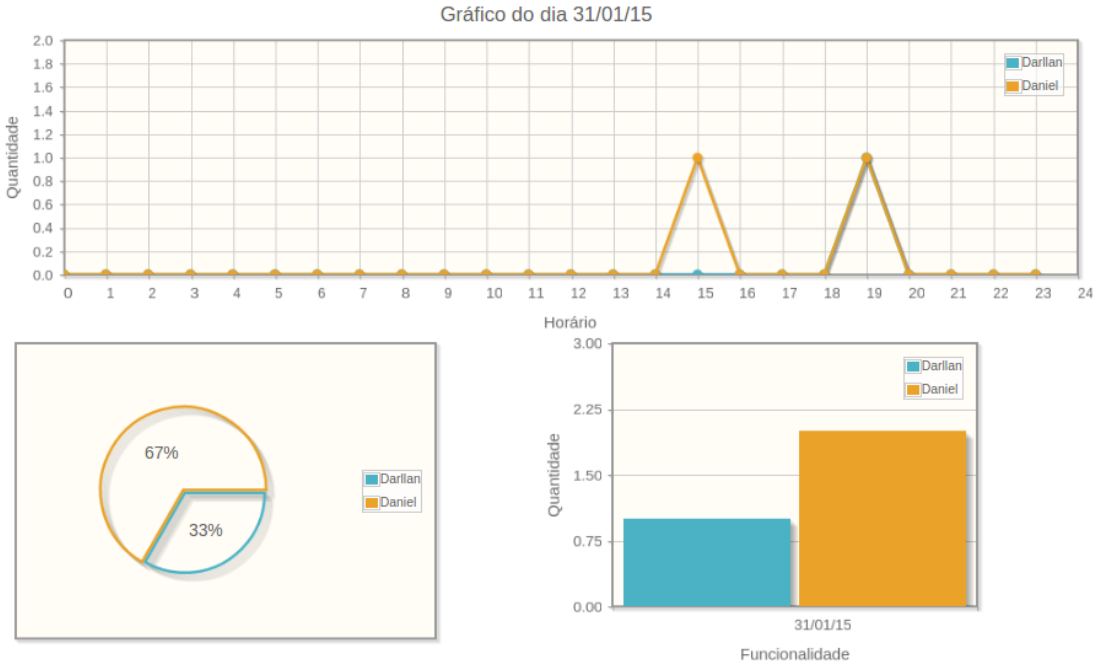

Figura 3.6: *Gráficos do* dashboard *ao pesquisarmos pelo sistema* Futebol-Ime-USP*, com usuário* Visitou Elenco*, em um determinado dia.*

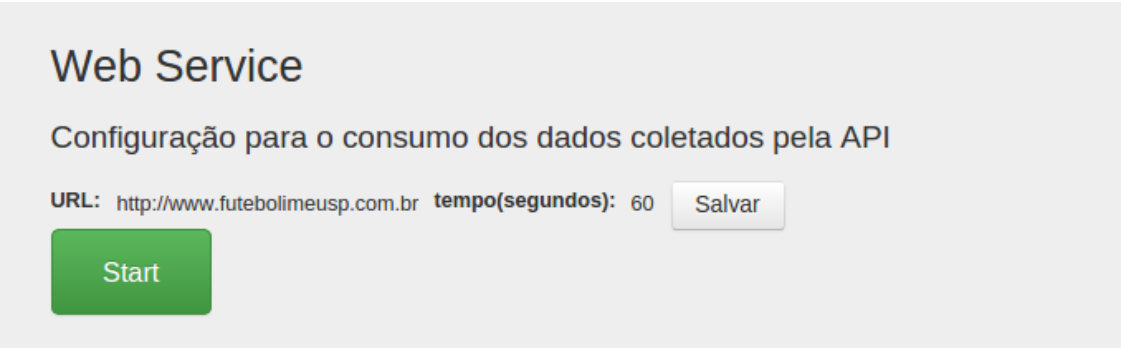

Figura 3.7: *Tela de* Web Service *no* dashboard *no site do futebol do IME.*

Capítulo 4

Parte Subjetiva

Foi um longo caminho até chegarmos aqui, e ninguém disse que seria fácil. Para terminar este trabalho, mostramos uma visão pessoal sobre o desenvolvimento do projeto e a graduação como um todo.

4.1 Bárbara

4.1.1 Desafios e Frustrações

Não tive muito contato com *Java* durante esses anos no IME, e foi um desafio aprender o suficiente da linguagem para entender e implementar este trabalho, além de todas as tecnologias envolvidas.

Foi particularmente difícil conciliar todas as atividades neste segundo semestre: TCC, trabalho, matérias com muitos projetos, etc. Mas o resultado foi muito satisfatório.

4.1.2 Disciplinas Relevantes

Falando não apenas sobre o trabalho, mas sobre todo o curso, nada disso teria sido possível se não fosse pelo professor Roberto Hirata Jr. ministrando MAC0110 (Introdução à Computação). A disciplina foi dada da maneira mais paciente possível, e eu, que não tinha nenhuma experiência com programação, consegui construir uma ótima base de conhecimento para o que viria a seguir.

Outras disciplinas fundamentais foram MAC0122 (Princípios de Desenvolvimento de Algoritmos) e MAC0323 (Estruturas de Dados). Além de nos acostumar com a lógica por trás de construir um programa, introduziram as estruturas mais importantes.

MAC0426 (Sistemas de Bancos de Dados) foi importante para o *dashboard* e também continuará sendo um ótimo conhecimento para a vida.

4.1.3 Próximos Passos

O Tiktak final se mostrou útil e muito simples para sistemas *Java*. Seria interessante seguir com implementações para outras linguagens entre as mais populares, como *C* e *Python*.

4.1.4 Considerações Finais

O desenvolvimento do Tiktak foi uma ótima oportunidade de aprendizado.

Gostaria de agradecer aos meus pais, que me proporcionaram a educação que me trouxe até a USP, e aos meus amigos, que ajudaram muito durante todos esses anos, alegrando os dias difíceis.

Por último, deixo um agradecimento especial ao professor Alfredo Goldman, que esteve disponível para nos ajudar o ano todo, e ao Leonardo, que trilhou este caminho comigo por todos estes anos.

4.2 Leonardo

4.2.1 Desafios e Frustrações

Há 7 anos eu começava minha jornada no Instituto de Matemática e Estatística da USP. Sempre fiz escola pública e passar em um dos vestibulares mais concorridos do país foi uma tarefa nada fácil. A estrada foi longa, mais longa do que eu imaginava. Não imaginei que seria tão difícil chegar até aqui. Algumas vezes me senti desmotivado, principalmente nas horas em que o esforço não levou a resultados. Mas tudo serviu como aprendizado. Cresci muito com os erros e hoje tenho muito orgulho de estar completando este ciclo na minha vida.

4.2.2 Disciplinas Relevantes

Durante o curso, muitas vezes precisei aprender sozinho, aprender a buscar respostas para perguntas nada triviais. Trabalhar com prazos apertados me tornou mais disciplinado. Por esses motivos, de certa forma, todas as disciplinas me acrescentaram algo, e este trabalho é fruto de todo esse aprendizado. Porém, algumas delas podem ser destacadas:

- MAC0110 - Introdução à Computação, MAC0122 - Princípio de Desenvolvimento de Algoritmos, e MAC0323 - Estrutura de dados: me deram uma base em programação que me ajudou durante todo o curso e, consequentemente, durante o trabalho de formatura.
- MAC0342 - Laboratório de Programação Extrema: foi fundamental para este trabalho. Foi nela que eu tive contato com o Tiktak e com o seu criador, Renan Oliveira. Além disso, conheci o excelente professor Alfredo Goldman, que gentilmente aceitou ser orientador do projeto. Aprendi várias ferramentas que não só utilizei durante o trabalho de formatura, como utilizo até hoje no mercado de trabalho.
- MAC0316 - Conceitos Fundamentais de Linguagens de Programação e MAC0332 - Engenharia de Software: me deram a base que facilita o aprendizado de qualquer linguagem de programação. Por exemplo, elas me ajudaram a aprender *Java* e orientação a objetos (linguagem e paradigma utilizados neste trabalho).
- MAC0426 - Sistemas de Bancos de Dados e MAC0439 - Laboratório de Bancos de Dados: me ajudaram a manipular o banco de dados do *dashboard*.

4.2.3 Próximos Passos

O Tiktak é um produto que ainda pode ser aprimorado e divulgado. Criar uma API para outras linguagens e disponibilizar um *dashboard* na nuvem são duas das muitas coisas que podem ser exploradas. Pretendo ainda trabalhar e aprender muito com ele.

4.2.4 Considerações Finais

O curso foi trabalhoso, passei por muitas dificuldades e elas só foram vencidas porque eu tive pessoas muito especiais ao meu lado.

Gostaria de agradecer a meus pais e familiares, que me apoiaram durante toda a minha formação; a todos os amigos que fiz no instituto e que, de certa forma, me ajudaram a chegar até aqui.

Por último, gostaria de agradecer ao professor Alfredo Goldman por ter aceitado ser orientador deste projeto e à minha amiga Bárbara por todos esses anos de muito trabalho e amizade.

Meu muito obrigado a todos!

Referências Bibliográficas

[BACS] Bárbara. https://linux.ime.usp.br/~bacs/mac0499/. 12, 14

[BSP] Bootstrap. http://getbootstrap.com/. 9

[DBE] DB-Engines. http://db-engines.com/en/ranking. 6

[FJ-21] Caelum. Java para Desenvolvimento Web. https://www.caelum.com.br/apostila-java-web.

[HBN] Hibernate. http://hibernate.org/. 7

[IEE] IEEE. As linguagens mais populares pelo ranking da IEEE. http://info.abril.com.br/noticias/it-solutions/2014/07/quais-as-linguagens-mais-populares-este-grafico-responde.shtml. 2

[JAX-RS] JAX-RS. https://jax-rs-spec.java.net/. 9

[JEE] Java EE. http://www.oracle.com/technetwork/java/javaee/overview/index.html. 6

[JMN] Jmine. http://www.jmine.com.br/wiki/index.php/Página_principal. 5

[JNT] JUnit. http://junit.org/. 9

[JSN] JSON. http://json.org/. 8

[K12] K19. Desenvolvimento Web com JSF2 e JPA2 http://www.k19.com.br/downloads/apostilas/java/k19-k12-desenvolvimento-web-com-jsf2-e-jpa2.

[LSO] Leonardo. https://linux.ime.usp.br/~lsoliveira/mac0499/. 12, 14

[MSL] MySQL. http://www.mysql.com/. 6

[MVN] Maven. http://maven.apache.org/. 9

[PFS] Primefaces. http://www.primefaces.org/. 9

[TBE] Tiobe. Tiobe Index for November 2014. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. 2