CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Análise de Sistemas e Tecnologia da Informação

Clayton José de Oliveira

TÉCNICAS PARA RECUPERAR, MANIPULAR E ANALISAR DADOS REMOTOS ATRAVÉS DA API KIVA

Americana, SP

2015

CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Análise de Sistemas e Tecnologia da Informação

Clayton José de Oliveira

TÉCNICAS PARA RECUPERAR, MANIPULAR E ANALISAR DADOS REMOTOS ATRAVÉS DA API KIVA

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Análise de Sistemas e Tecnologia da Informação, sob a orientação da Profª. Drª. Maria Cristina Aranda

Área de concentração: desenvolvimento de *software*.

Americana, S. P.

2015

Oliveira, Clayton José

O46t Técnicas de recuperar, manipular e analisar dados remotos através da API KIVA. / Clayton José de Oliveira. – Americana: 2015.

52f.

Monografia (Graduação em Tecnologia em Análise de Sistemas e Tecnologia da Informação). -- Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.

Orientador: Prof. Dr. Maria Cristina Aranda

1. Desenvolvimento de software I. Aranda, Maria Cristina II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU:681.3.05

Clayton José de Oliveira

TÉCNICAS PARA RECUPERAR, MANIPULAR E ANALISAR DADOS REMOTOS ATRAVÉS DA API KIVA

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Bacharel em Análise de Sistemas e Tecnologia da Informação pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Desenvolvimento de *software*.

Americana, 10 de Dezembro de 2015.

Banca Examinadora:

Maria Cristina Aranda (Membro)
Doutora
Fatec Americana

Maria Elizete Luz Saes (Membro)
Mestra
Fatec Americana

Antonio Alfredo Lacerda (Membro)
Especialista
Fatec Americana

AGRADECIMENTOS

Gostaria de agradecer a todos que de alguma forma me ajudaram e incentivar a realizar este trabalho.

RESUMO

Este trabalho apresenta algumas técnicas que podem ser utilizadas para a recuperação, manipulação e armazenamento de dados obtidos de servidores remotos. A Internet é uma grande rede de sistemas que disponibilizam recursos. Algumas empresas usam essa capacidade da Internet para divulgar resultados ou para ampliar seus negócios. As empresas escolhidas trabalham com microfinanciamento e para isso utilizam páginas e serviços *web* de diferentes formas: uma possui uma Application Programming Interface (API) criada especificamente para que se tenha acesso a todos os dados relevantes sobre suas atividades e a outra apenas divulga suas atividades através de tabelas em uma página HyperText Markup Language (HTML) comum. Este trabalho consiste em analisar a documentação da API e também a estrutura das páginas para que se possa ter acesso aos dados relevantes sobre as atividades das duas empresas. Para isso foi feito um estudo da API do site Kiva, que proporcionou acesso de forma simples e direta aos dados da empresa na forma de arquivos JavaScript Object Notation (JSON). Para efeito de comparação foi usado o site myc4.com, o qual apenas disponibiliza os dados na forma de tabelas e por isso foi necessário analisar a estrutura das páginas HTML e, através da identificação das marcações de tabela, recuperar os dados relevantes. Todos os dados recuperados tanto pela API quanto diretamente do código HTML podem ser armazenados para análise, visualização ou manipulação futuras.

Palavras Chave: serviços *web*, análise de dados, notação de objetos Javascript.

ABSTRACT

This paper presents some techniques that can be used for recovery, handling and storage of data obtained from remote servers. The Internet is a large network of systems that provide resources. Some companies use this ability of the Internet to disseminate results or grow their business. The chosen companies work with microfinance and for that use web pages and services in different ways: one has an API designed specifically in order to have access to all relevant data on its activities and the other only disclose their activities through tables on a common HTML page. This work consists in analyzing the Application Programming Interface (API) documentation and also the structure of the page so you can have access to the relevant data on the activities of both companies. For this purpose we analized the Kiva site API, which provided access in a simple and direct way to enterprise data as JSON files. For comparison, we used the site myc4.com, which only makes the data available in the form of tables and therefore, it was necessary to analyze the structure of HTML pages and, by identifying the table tags, retrieve the relevant data. All data retrieved by both the API or directly as HTML code can be stored for analysis, visualization and future manipulation.

Keywords: *web services, data analysis, JavaScript object notation.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. Introdução

O presente trabalho foi realizado com o objetivo de estudar e desenvolver um método prático e confiável de acessar e manipular dados disponíveis na Internet, tornando esses dados disponíveis para posterior análise e interpretação.

O maior problema para a obtenção de dados armazenados de forma remota é a forma em que eles estão disponibilizados. Algumas instituições disponibilizam ferramentas para facilitar o acesso a esses dados, enquanto outras apenas os disponibilizam para visualização, geralmente em forma de tabelas em páginas HTML.

Este trabalho estuda formas de obter dados de diferentes fontes usando ferramentas como uma Application Programming Interface (API) disponibilizada pelo provedor dos dados ou uma técnica de identificação e extração de dados (parser) em arquivos HTML. Os dados obtidos podem ser armazenados em um banco de dados ou visualizados em uma ferramenta de análise e visualização de dados.

Esse estudo poderá servir de base para a elaboração de ferramentas para a obtenção, armazenamento e análise de dados disponíveis em diferentes plataformas.

O principal objetivo do presente trabalho é criar códigos para acessar dados disponíveis remotamente na forma de arquivos JSON, XML ou HTML. Os dados obtidos devem ter a capacidade de serem armazenados localmente e analisados.

Além do objetivo principal, este trabalho também se concentrou nos seguintes objetivos:

- Estudar a documentação da API do site Kiva.org;
- Criar códigos para acessar os dados disponibilizados através dessa API;
- Estudar técnicas de identificação e extração de dados para obter dados do site Myc4.com;
- Criar um banco de dados para o armazenamento, consulta e alteração dos dados obtidos a partir da base de dados do Kiva;
- Visualizar esses dados para que possam ser analisados.

No desenvolvimento deste trabalho foram estudados alguns aspectos do JSON e XML necessários para que a coleta e manipulação de dados fossem possíveis. Não foi feito um estudo aprofundado sobre as duas linguagens citadas. O funcionamento

da organização Kiva está descrito de forma superficial para que o leitor entenda como essa instituição de microcrédito opera, sem precisar se aprofundar em detalhes.

Este trabalho pode também ser usado como referência para os interessados em formas de obtenção, manipulação e análise de dados, pode ser usado também como base para um estudo mais profundo sobre o tema e seus desenvolvimentos futuros.

2. A Internet

A Internet é um grande conjunto de diferentes tipos de redes interligadas (COULOURIS, DOLLIMORE e KINDBERG, 2007). A interação entre os programas executados nos computadores conectados à Internet é feita com o envio de mensagens utilizando um meio de comunicação compartilhado. As mensagens são enviadas seguindo um padrão descrito nos protocolos Internet.

Os autores definem serviço como “uma parte distinta de um sistema de computador que gerencia um conjunto de recursos relacionados e apresenta sua funcionalidade para usuários e aplicativos. ”

Segundo Coulouris, Dollinmore e Kindberg (2007), uma interface de serviço web geralmente consiste em um conjunto de operações que podem ser usadas por um cliente na Internet. Essas operações são disponibilizadas por diferentes recursos, como programas, objetos ou banco de dados.

Serviço web é definido por Haas (2005) como:

> “Um serviço web é um sistema de software projetado para suportar interação interoperacional máquina-máquina” através de uma rede. Ela possui uma interface descrita em um formato processável por máquina (especificamente WSDL). Outros sistemas interagem com o serviço web de uma maneira prescrita por sua descrição usando mensagens SOAP, tipicamente transmitidas usando HTTP com uma serialização XML em conjunto com outras normas relacionadas à Web”.

Nos próximos tópicos serão apresentadas algumas informações relevantes para a compreensão do trabalho realizado.

A World Wide Web

A world wide web (BERNES-LEE, 1991) *apud* (COULOURIS; DOLLIMORE; KINDBERG, 2007) é um grande sistema que tem como finalidade a publicação e o acesso de recursos e serviços pela Internet. Diferentes tipos de documentos e conteúdos podem ser acessados através de um navegador.

Esse sistema foi criado como um meio de troca de informações no Centro Europeu de Pesquisa Nuclear (Conseil Européen pour la Recherche Nucléaire - CERN). A web tem como característica ser organizada numa estrutura de hipertextos

entre seus documentos. Seu conteúdo é necessariamente organizado e contem referências para outros recursos ou documentos.

A web é basicamente formada por um conjunto de hipertextos distribuídos (HAAS, 2005). Todo seu conteúdo pode ser considerado como recursos da web e ser identificado por uma URI (Uniform Resource Identifier). Através de um navegador, podemos recuperar uma representação desses recursos. Usa-se o método HTTP GET para recuperar esses recursos. O método GET propicia que os recursos sejam acessados de forma segura, pois não permite mudanças no recurso original; pode ser salvo na forma de um “favorito” pelo navegador e sofrer outras ações típicas desses programas.

O protocolo HTTP possui outros métodos para a manipulação de recursos que permitem a criação, atualização e até a apagar recursos. Esses métodos são respectivamente: POST, PUT e DELETE.

JavaScript Object Notation – JSON

A notação de objetos JavaScript é definida por Ecma International (2013) como um formato leve para a troca de dados. JSON foi derivada da linguagem de programação ECMAScript®, mas é independente de linguagens de programação.

Na página “Introdução ao JSON”, o site (JSON.ORG, 2015) nos diz: “JSON é em formato texto e completamente independente de linguagem, pois usa convenções que são familiares às linguagens C e familiares, incluindo C++, C#, Java, JavaScript, Perl, Python e muitas outras. Estas propriedades fazem com que JSON seja um formato ideal de troca de dados.”

As estruturas de dados que constituem o JSON são universais e estão presentes em praticamente todas as linguagens de programação modernas (JSON.ORG, 2015). Essas estruturas podem ser um conjunto de pares nome/valor ou uma lista ordenada de valores.

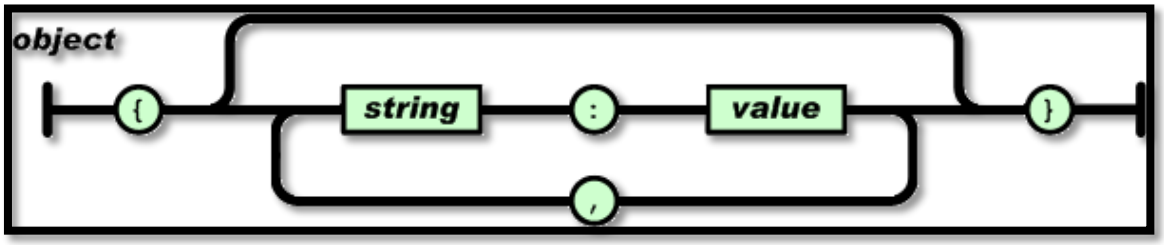
Apresentação de dados em JSON

Dentro dessa notação, os dados são apresentados como: objetos, vetores, valores, cadeias de caracteres e números. As figuras a seguir, retiradas da página “Introdução ao JSON” ilustram essas estruturas:

Objeto

A representação de um objeto (conjunto de pares de nomes e valores) é feita entre chaves de abertura e de fechamento { }. O nome do par é seguido pelo símbolo : (dois pontos). Cada item nome/valor é separado do seguinte por uma vírgula. (JSON.ORG, 2015)

Figura 1: representação de um objeto em JSON

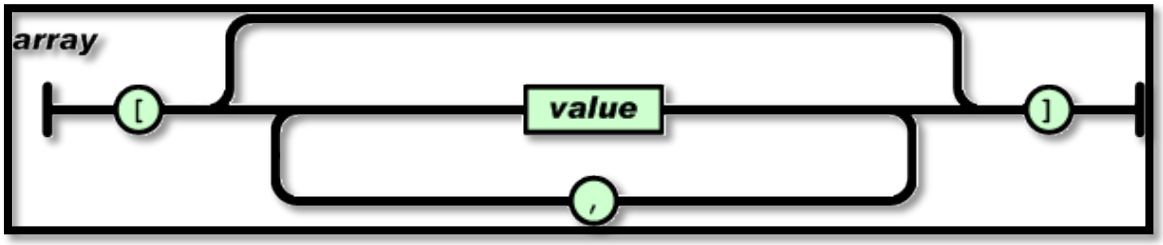


Fonte: (JSON.ORG, 2015)

Vetor

Um conjunto de valores ordenados é chamado de vetor. Esta estrutura é representada entre colchetes de abertura e fechamento [ ]. Os valores que constituem um vetor são separados por vírgulas. (JSON.ORG, 2015)

Figura 2: representação de um vetor em JSON

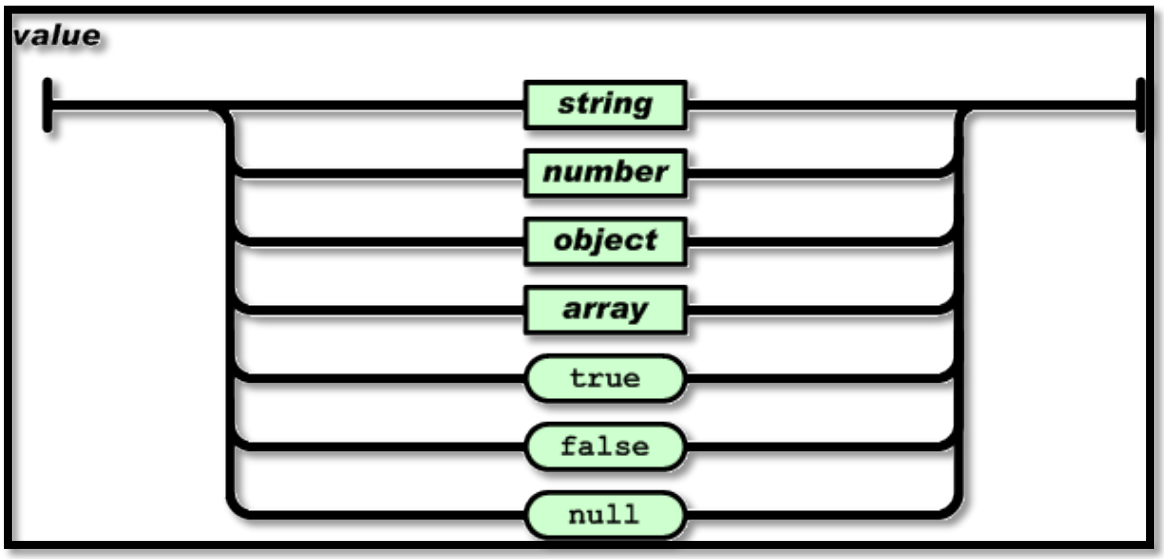


Fonte: (JSON.ORG, 2015)

Valor

Valores em JSON podem ser uma string (cadeia de caracteres), um número, um valor boleano (true ou false), um valor nulo (null), ou até um objeto ou um vetor. Estas estruturas de dados podem ser aninhadas. (JSON.ORG, 2015)

Figura 3: representação de um valor em JSON

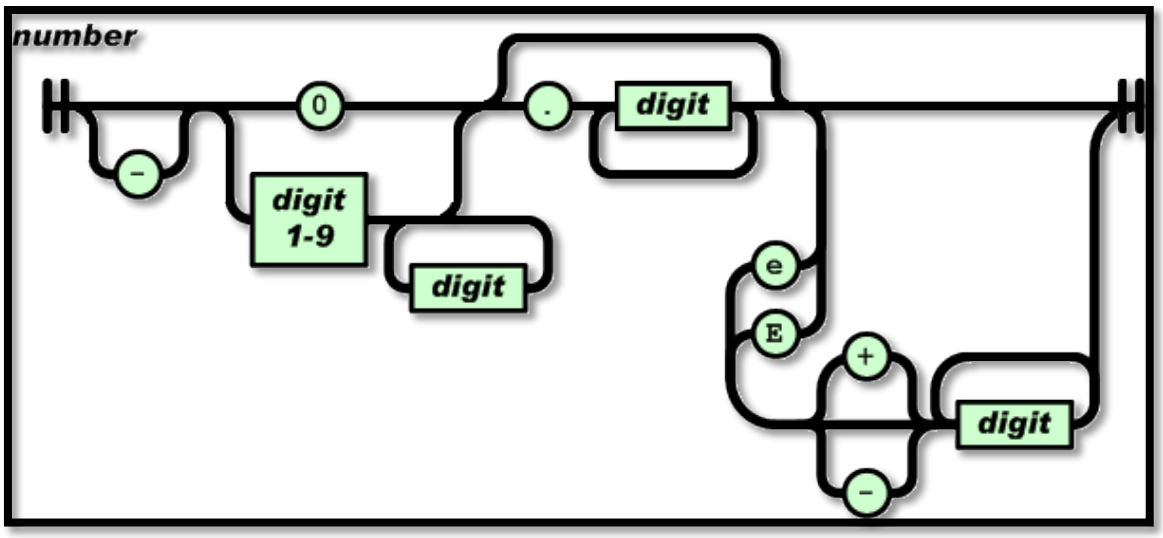


Fonte: (JSON.ORG, 2015)

Número

Números são representados de forma semelhante a um número em outras linguagens, como C ou Java. A diferença é que em JSON um número não é representado nos sistemas hexadecimal e octal. (JSON.ORG, 2015)

Figura 4: representação de um número em JSON

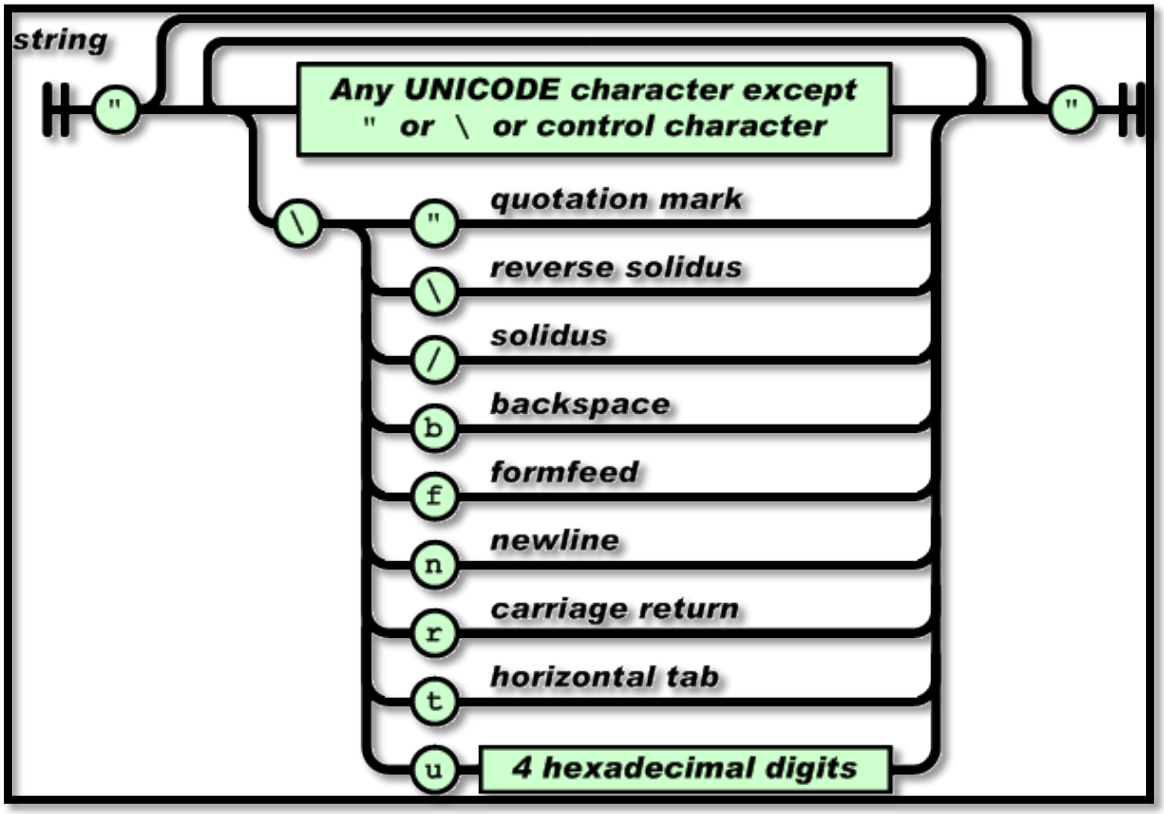


fonte: (JSON.ORG, 2015)

Cadeia de caracteres - string

Dados formados por caracteres Unicode constituem uma string. Essa coleção pode conter nenhum ou mais caracteres. Uma string JSO é semelhante com uma cadeia de caracteres Java ou C. (JSON.ORG, 2015)

Figura 5: representação de uma string em JSON



Fonte: (JSON.ORG, 2015)

Extensible Markup Language - XML

Extensible Markup Language (XML) é uma formatação de texto simples e flexível derivada de SGML (ISO 8879). Projetada inicialmente para enfrentar os desafios da publicação eletrônica em larga escala. Essa linguagem também está desempenhando um papel importante na troca de dados na web (W3C, 2015).

A linguagem XML foi desenvolvida para ser de fácil entendimento para as pessoas e simples de ser usada por computadores em sua tarefa de estocar e transportar dados. A seguir, dois exemplos de arquivos XML retirados do site w3schools.com (W3SCHOOLS, 2015):

Figura 6: Exemplo simples de XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Fonte: (JSON.ORG, 2015)

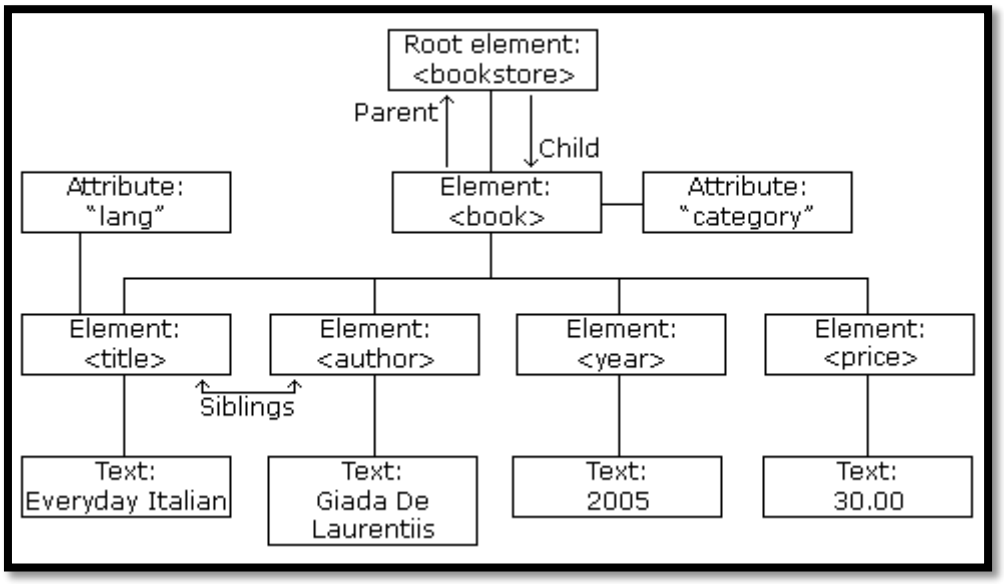
Figura 7: Outro exemplo da linguagem XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
<food>
  <name>Belgian Waffles</name>
  <price>$5.95</price>
  <description>Our famous Belgian Waffles with plenty of real maple syrup</description>
  <calories>650</calories>
</food>
<food>
  <name>French Toast</name>
  <price>$4.50</price>
  <description>Thick slices made from our homemade sourdough bread</description>
  <calories>600</calories>
</food>
<food>
  <name>Homestyle Breakfast</name>
  <price>$6.95</price>
  <description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
  <calories>950</calories>
</food>
</breakfast_menu>
```

Fonte: (JSON.ORG, 2015)

Documentos XML tem a estrutura de uma árvore com um elemento raiz e ramificações como mostra a figura a seguir retirada do site w3schools.com (W3SCHOOLS, 2015):

Figura 8: Estrutura de um documento XML



Fonte: (W3SCHOOLS, 2015)

REST

REST é um estilo arquitetural para sistemas de hipermídia distribuída (HAAS, 2005) e proporciona uma interface uniforme para recursos. Essa interface está presente em alguns serviços web, como o Kiva por exemplo, e fazendo uso dos métodos HTTP citados acima, é utilizada para a manipulação e acesso a recursos. Os dados recuperados usando REST são representados normalmente na forma de documentos XML ou JSON.

3. A organização Kiva

Kiva é uma organização sem fins lucrativos com a missão de conectar pessoas através de empréstimos em dinheiro que ajudam a aliviar a pobreza. Aproveitando a Internet e uma rede mundial de instituições de micro finanças, Kiva permite que indivíduos emprestem pequenas quantias para ajudar a criar oportunidades em todo o mundo. (KIVA MICROFUNDS, 2015).

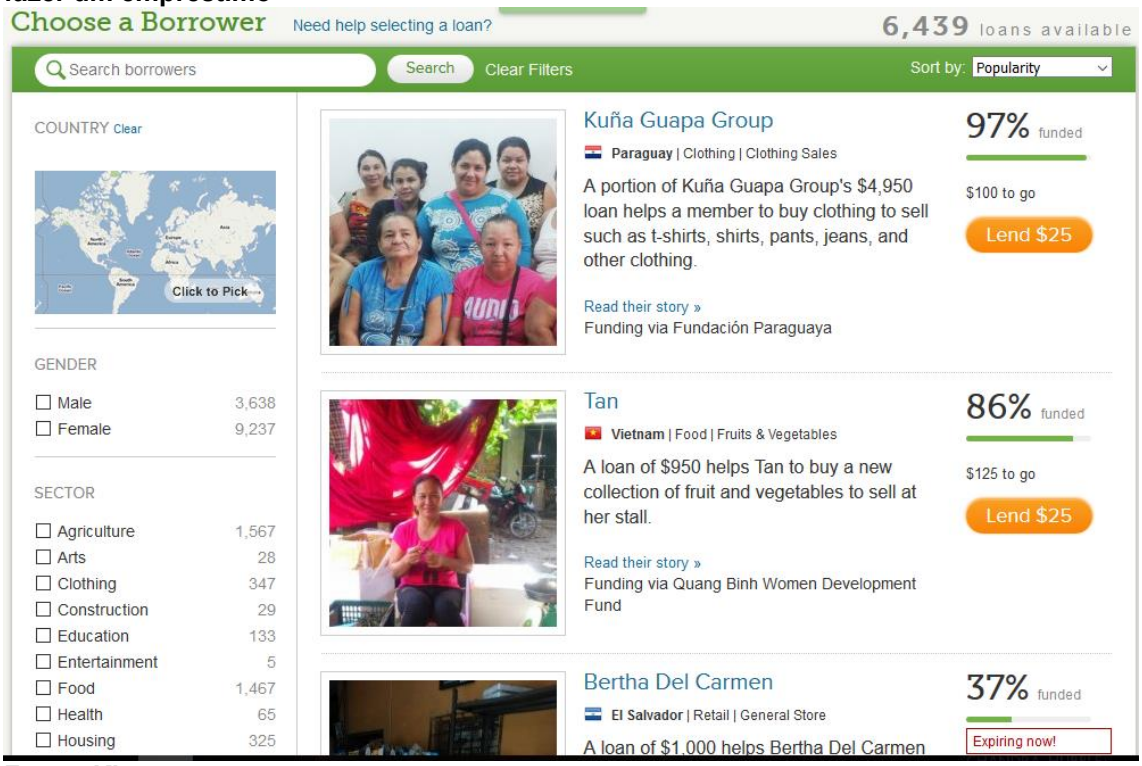
Um empréstimo começa quando alguém interessado escolhe um mutuário. Essa escolha é feita baseada nas histórias dos muitos mutuários que estão cadastrados na base de dados do Kiva. Para controlar esses dados e os empréstimos, o Kiva possui parceiros de campo nos países onde atua. Esses parceiros podem ser Organizações não governamentais, pequenas instituições financeiras locais, escolas etc. Todos esses parceiros compartilham a missão social de aliviar a pobreza e melhorar a vida das pessoas.

Após a escolha do mutuário, o parceiro de campo responsável concede o empréstimo e atualiza o diário do empréstimo no sistema à medida que o processo vai se desenvolvendo. Muitos dos empréstimos são concedidos até antes de se conseguir uma pessoa disposta a emprestar para esses mutuários. Então os parceiros de campo trabalham para divulgar essas pessoas, promovendo suas histórias para que os fundos usados para os empréstimos sejam coletados.

Qualquer pessoa com acesso à Internet pode se tornar um emprestador. Emprestando uma pequena quantia para ajudar no projeto em que está interessada. O Kiva atua como um agregador, reunindo todos os pequenos empréstimos em um financiamento colaborativo para os mutuários. Os emprestadores assumem o risco para o empréstimo que ele escolheu.

Conforme os mutuários vão realizando os pagamentos dos empréstimos, os parceiros de campo repassam o dinheiro de volta para o Kiva. Os fundos ficam então disponíveis novamente para os emprestadores. Normalmente as pessoas escolhem usar esse dinheiro para financiar outras pessoas, mas também podem retirá-lo ou fazer uma doação para o Kiva.

Figura 9: Página do Kiva contendo as histórias das diferentes pessoas para quem se pode fazer um empréstimo



Fonte: Kiva.org

O Kiva foi fundado como uma organização sem fins lucrativos em outubro de 2005 e em 10 anos provendo credito alcançou as seguintes estatísticas (KIVA MICROFUNDS, 2015):

- Está presente em 83 países;
- Financiou 905.086 empréstimos, sendo que 75% para mulheres;
- Atingiu um montante emprestado de 774 milhões de dólares;
- Possui 1,4 milhão de emprestadores que financiaram 1,8 milhões de mutuários;
- Das pessoas que são ajudadas pelos empréstimos, 202.234 vivem em zonas de conflito; 600.910 vivem em países pouco desenvolvidos e 397.958 são fazendeiros;
- Proporcionou 22.627 empréstimos educacionais e proporcionou acesso à energia limpa para 26.022 mutuários.

4. Convenções da API Kiva

RESTful

A API Kiva é um serviço web (KIVA MICROFUNDS, 2015), que foi desenvolvida para funcionar com ferramentas comuns da Internet, como navegadores, através dos protocolos da internet (HTTP). A API Kiva é orientada a recursos que utiliza URLs que apontam para dados como chamadas de métodos. Esses dados podem ser consultados ou modificados. Essa tecnologia é chamada “REST”. A API Kiva não se limita a uma definição formal dessa tecnologia e por isso é considerada uma API para serviços web RESTful.

Para realizar uma consulta sobre algum dos recursos do Kiva é necessário realizar uma requisição HTTP GET usando o URI apropriado para o recurso que se deseja obter informações. O método POST poderia ser usado para criar um recurso, como um empréstimo, mas a API apenas permite que consultas sejam realizadas pelos usuários.

Formato da resposta

Uma chamada para a API sempre gera algum tipo de resposta. Se for bem-sucedida, essa chamada terá uma resposta HTTP que conterá dados e o status 200 ok. Os dados contidos na resposta podem ser requisitados em dois formatos: JSON ou XML.

Abaixo um exemplo de uma resposta bem-sucedida da API retirado do site build.kiva.org:

JSON:

```
{
        "people":
                [
                        {"name":"Jeremy","id":2},
                        {"name":"Roma","id":38},
                        {"name":"Zvi","id":21}
                ]
}
```

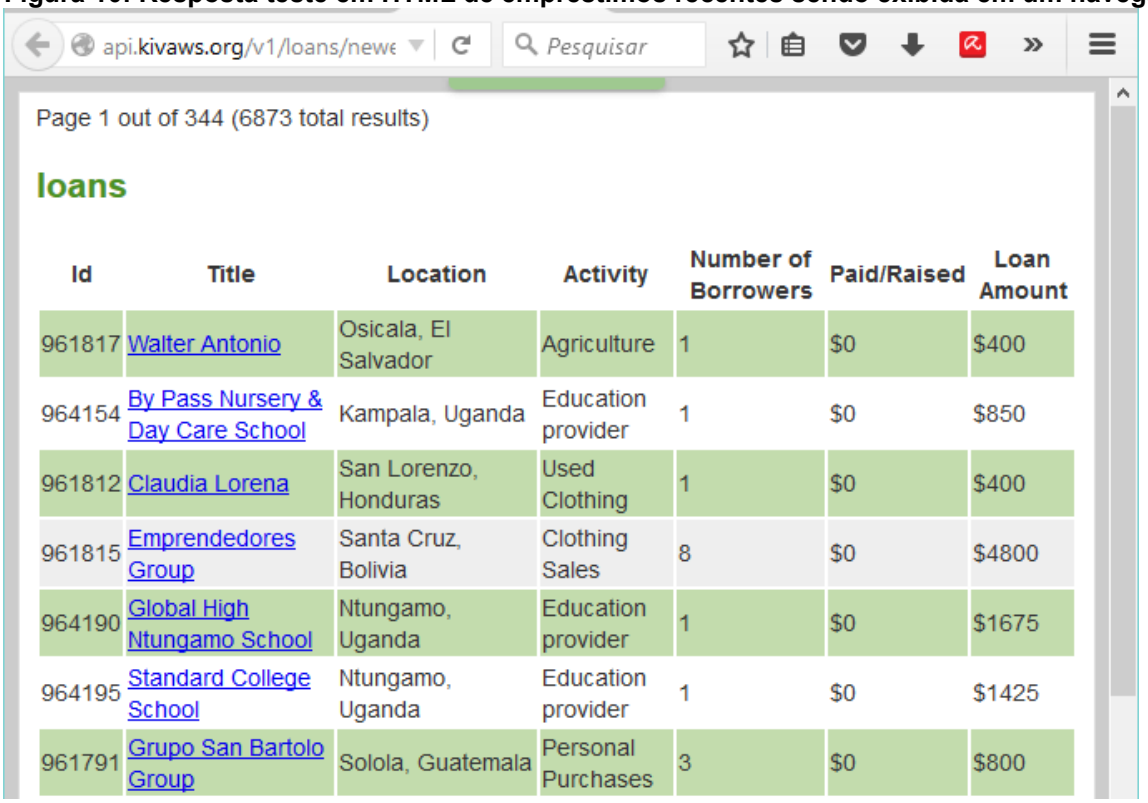
XML:

```
<response>
    <people type="list">
        <person><name>Jeremy</name><id>2</id></person>
        <person><name>Roma</name><id>38</id></person>
        <person><name>Zvi</name><id>21</id></person>
    </people>
</response>
```

Resposta Teste

Além de fornecer os dados serializados (representação de um objeto em formato texto) nos formatos JSON e XML para serem utilizados por outros programas, a API Kiva também pode disponibilizar os dados como uma página HTML para ser visualizada em um navegador. Sendo uma API RESTful ela pode ser requisitada em um navegador comum e gerar uma resposta legível para um ser humano.

Figura 10: Resposta teste em HTML de empréstimos recentes sendo exibida em um navegador



Fonte: Kiva.org

Erros

Uma requisição HTTP à API Kiva pode gerar outros códigos de status além do 200 OK. Esses códigos de status são os normais para qualquer requisição HTTP. A seguir uma tabela retirada da documentação da API com alguns erros HTTP mais comuns retornados:

Tabela 1: Erros comuns em requisições HTTP

400 Bad Request	Alguma coisa estava errada com o seu pedido. Provavelmente você passou um valor de parâmetro incorreto.
401 Unauthorized	O pedido que você fez requer autorização e nenhuma credencial foi fornecida ainda. Provavelmente, você esqueceu um cabeçalho de autorização para um recurso que exige isso.
403 Forbidden	O seu pedido foi entendido pelo servidor e rejeitado. Isso é comum quando você solicita recursos sensíveis de forma insegura (por exemplo, não usando HTTPS para acessar dados confidenciais do usuário), ou se suas credenciais de autorização não coincidem com os dados que você está tentando acessar - por exemplo, fornecer credenciais de um credor para acessar dados privados de um parceiro.
404 Not Found	Não foi possível encontrar o recurso que você solicitou. Em outras palavras, seu URI está provavelmente incorreto. Pode ser o caso de um nome de recurso inexistente ou um parâmetro de identificação incorreto.
405 Method Not Allowed	O recurso solicitado é válido, mas ele não suporta o método HTTP usado para acessá-lo. Por exemplo, você pode ter enviado uma requisição POST para um recurso disponível apenas por uma requisição GET.
500 Internal Server Error	Este é nosso erro. Por favor, postar no fórum para que possamos analisá-la.
503 Service Unavailable	Os servidores Kiva estão sobrecarregados ou em manutenção. Neste último caso, vamos normalmente

	adicionar um cabeçalho Retry-After que permite que você saiba quando os serviços estarão novamente disponíveis.

Fonte: build.kiva.org

A API Kiva disponibiliza mais detalhes para cada erro reportado além do status HTTP. Essa resposta é serializada no formato requerido com o código do erro e uma mensagem descrevendo o erro retornado.

A documentação do site usa o seguinte exemplo de uma chamada para o método GET para esta URI:

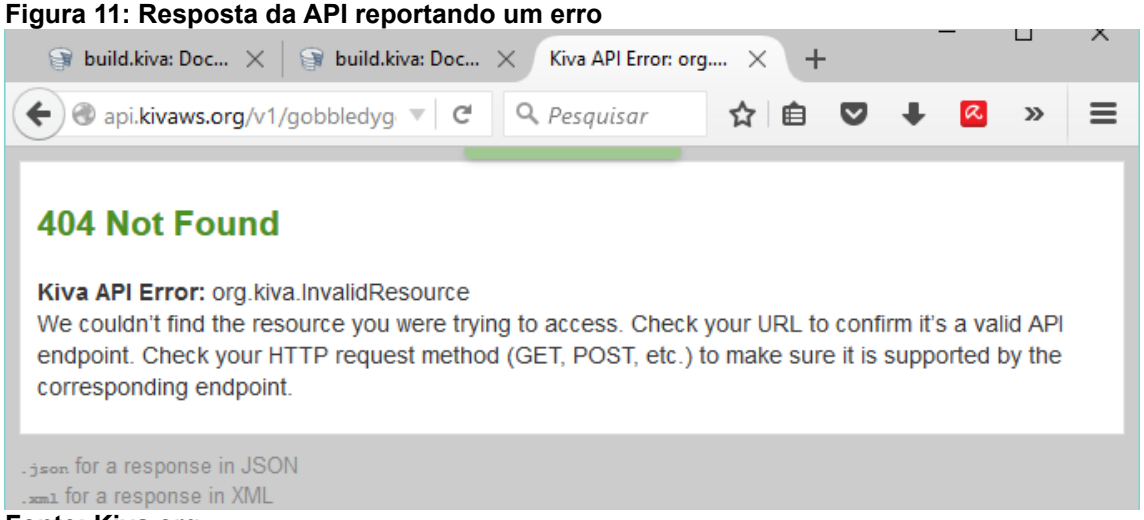
```
http://api.kivaws.org/v1/gobbledygook.json
```

Resposta da API:

```
{
  "code":"org.kiva.InvalidResource"
  "message":"We could't find the resource you were trying to access. Check your URL to confirm it's a valid API endpoint. Check your HTTP request method (GET, POST, etc.) to make sure it is supported by the corresponding endpoint."
}
```

Se a chamada fosse requisitada em XML, a resposta conteria um elemento raiz do erro. Uma resposta teste em HTML também retorna o *status* HTTP além do código e da mensagem da API.

Figura 11: Resposta da API reportando um erro



Fonte: Kiva.org

A mensagem retornada contém uma explicação do que ocorreu e procura ajudar na resolução do problema. O código é uma *string* única que pode ser usada para obter mais detalhes sobre o ocorrido, além do *status* HTTP. Esse código pode ser usado para informar os usuários do que aconteceu. Usando esses códigos os usuários podem receber mensagens personalizadas sobre o erro reportado. Os códigos mais comuns em respostas são:

Tabela 2: Códigos de erros da API com detalhes

org.kiva.InvalidResource	O URI na sua requisição está incorreto, de modo que não sabemos quais os recursos que você está procurando. Talvez o ponto final da API que você deseja chamar não existe ou você pode ter um parâmetro de forma incorreta no URI que está causando problemas.
org.kiva.MalformedParameter	Você informou um valor que estava fora dos limites do que nós permitimos para um determinado parâmetro.
org.kiva.InvalidIdentifier	Você informou um valor para um parâmetro identificador que não corresponde a qualquer coisa que nós conhecemos. Por exemplo, '111' não é um ID válido para um empréstimo.
org.kiva.HttpsRequired	Você está tentando usar HTTP para acessar um recurso que está disponível somente via HTTPS.
org.kiva.ServerMaintenance	Estamos atualizando nossos servidores para fazer nosso serviço melhor e a API Kiva não está disponível no momento. Tente mais tarde.

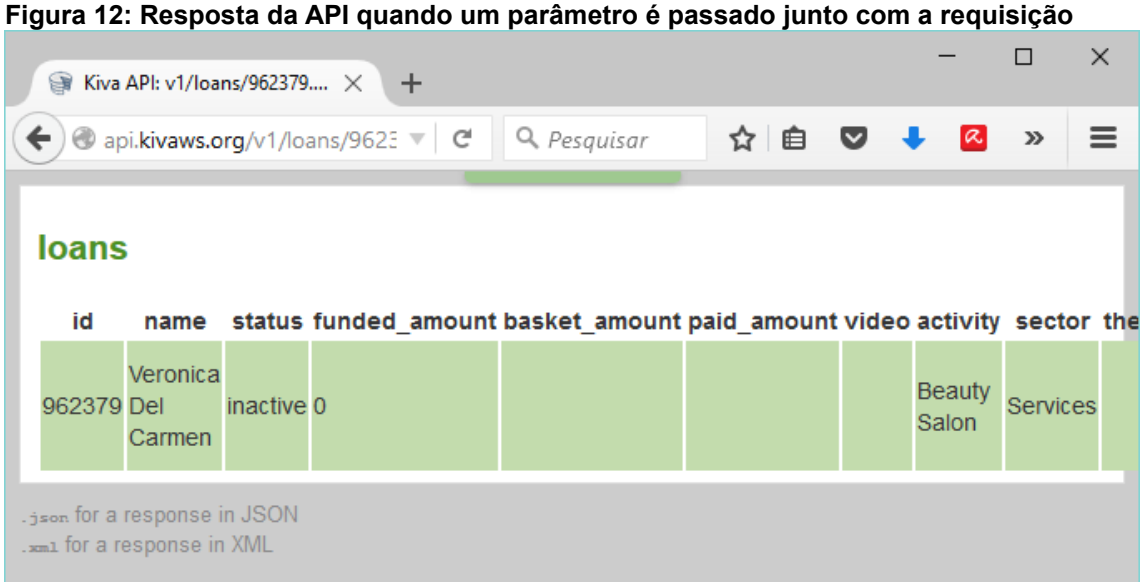
Fonte: build.kiva.org

Parâmetro

Uma requisição para a API só é possível se todos os parâmetros obrigatórios estiverem presentes na URI enviada pela requisição HTTP. Esses parâmetros obrigatórios não possuem um valor padrão. O exemplo a seguir retorna os detalhes sobre um empréstimo:

```
http://api.kivaws.org/v1/loan/962379.html
```

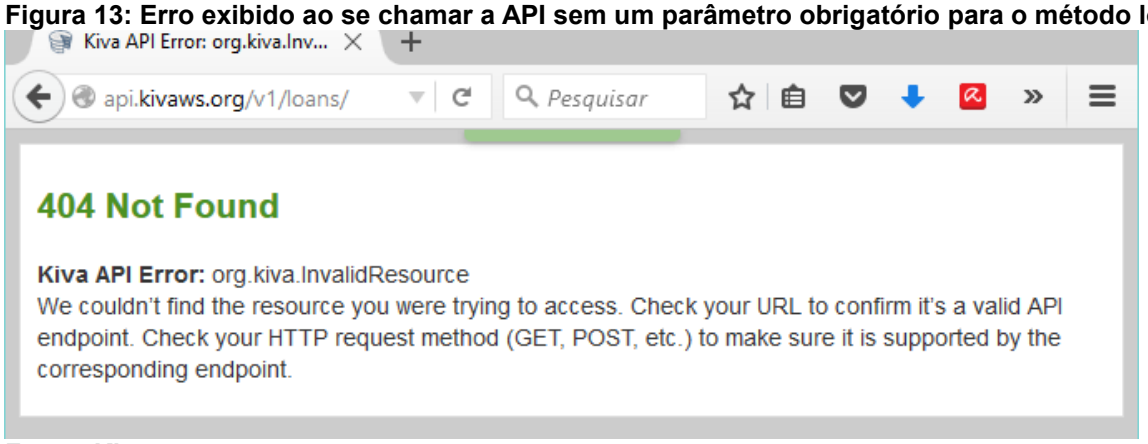
Figura 12: Resposta da API quando um parâmetro é passado junto com a requisição



Fonte: Kiva.org

O valor 962379 é um parâmetro obrigatório para o método loan/<id> da API. Esse valor pode ser mudado para qualquer outro id válido para um empréstimo. Qualquer tentativa de chamar o método loan sem fornecer um id válido resulta na página de recurso inválido da API.

Figura 13: Erro exibido ao se chamar a API sem um parâmetro obrigatório para o método loan

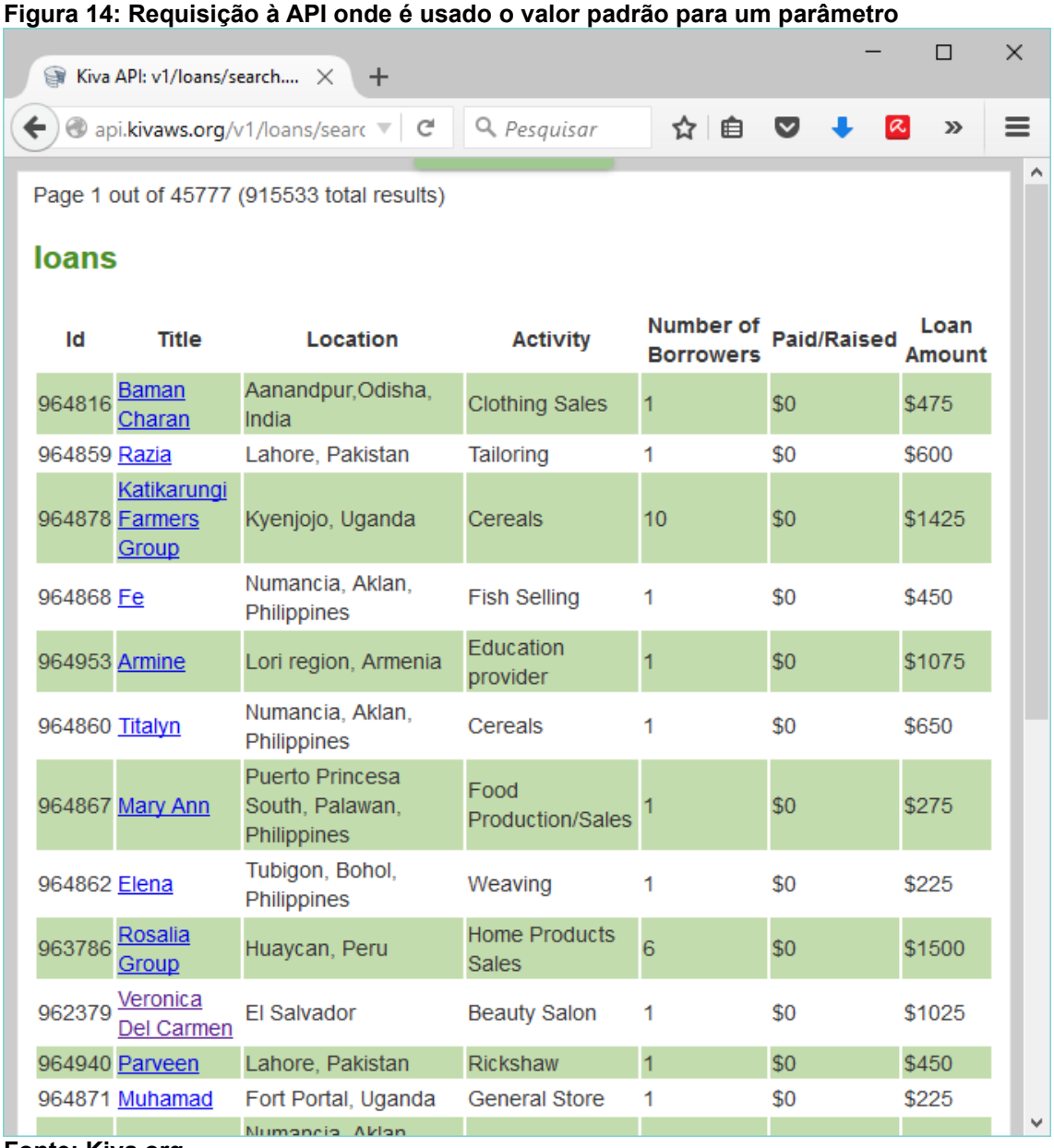


Fonte: Kiva.org

Parâmetros opcionais geralmente se encontram no final da requisição HTTP. Esses parâmetros opcionais podem ter valores padrão.

http://api.kivaws.org/v1/loans/search.html http://api.kivaws.org/v1/loans/search.html?page=1

Figura 14: Requisição à API onde é usado o valor padrão para um parâmetro



Fonte: Kiva.org

A mesma informação é retornada nas duas chamadas acima devido ao parâmetro pages ser opcional e possuir o valor 1 como padrão. Sendo ele passado

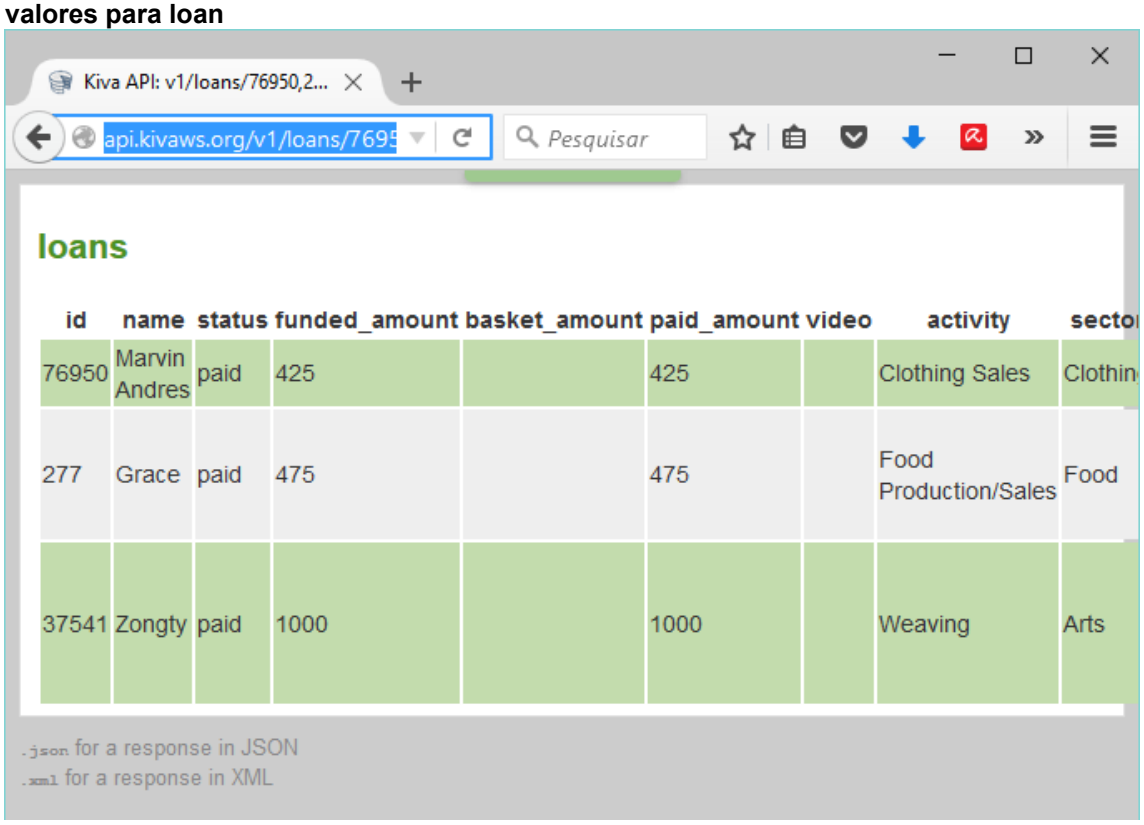
para a requisição HTTP no final da URI, ele não é necessário para que uma chamada a API seja completada.

Toda vez que um chamado à API é realizado todos os parâmetros são validados pela API Kiva. Se algum parâmetro conter valores inválidos a API irá retornar uma mensagem de erro na resposta.

Alguns parâmetros podem ser passados sobre a forma de vetores. Usar vetores torna possível requerer dados de diferentes recursos com uma única requisição HTTP. Exemplo:

```
http://api.kivaws.org/v1/loans/76950,277,37541.html
```

Figura 15: Resposta da API à uma solicitação onde o parâmetro é formado por um vetor de valores para loan



Fonte: Kiva.org

Nesse exemplo são consultados os dados de vários empréstimos com uma única requisição.

Paginação

A paginação é usada para evitar respostas muito grandes da API. Cadastrados no sistema existem milhares de recursos e cada um desses conjuntos de dados

podem ter um tamanho de muitos mega ou gigabytes. O uso da paginação é responsável para que as respostas às requisições sejam de tamanho adequado. Assim, os servidores do Kiva são capazes de responder a milhares de requisições e as aplicações que requisitam esses dados podem trabalhar essas informações.

Os desenvolvedores da API resolveram usar um modelo simples para dividir os dados em páginas de tamanho fixo para cada tipo de dados. Com o uso de um tamanho fixo de dados por página, uma requisição por mais dados é feita simplesmente fornecendo como parâmetro o número da página desejada.

O código a seguir exemplifica a paginação para empréstimos:

Figura 16: Fragmento de um arquivo JSON contendo dados de paginação

```
"paging": {
    "page": 1          // a página atual exibida
    "total": 105       // número total de empréstimos para essa chamada
    "page_size": 20    // número de recursos listados por página
    "pages": 6         // número total de páginas disponíveis
}
```

Fonte: build.kiva.org

Nesse exemplo, para se obter os dados de todos os empréstimos, é necessário chamar esse método da API por mais 5 vezes, variando o valor do parâmetro page de 2 a 6.

Buscando dados através da API Kiva

A função da API Kiva é ajudar o usuário a encontrar dados sobre o Kiva (KIVA MICROFUNDS, 2015). A documentação do site foi feita para ajudar na procura e representação dos dados obtidos em aplicativos e outros produtos. Os termos usados na API foram escolhidos para facilitar o entendimento tanto por programadores quanto por outros usuários do sistema.

A seguir algumas definições usadas para descrever os dados dos recursos disponíveis:

Tabela 3: Recursos disponíveis através da API

Loan (empréstimo)	Um empréstimo é o objeto de dados mais importante no Kiva. A maioria dos outros objetos estão de alguma forma relacionados a um empréstimo. Isso faz sentido; facilitar empréstimos que mudam vidas está no coração da missão do Kiva. Você provavelmente já viu empréstimos referenciados a "empresas" ou "empresários" em outros lugares por causa da estreita relação entre todos esses conceitos. No entanto, na API, se você quer mostrar uma listagem de qualquer uma dessas coisas em sua aplicação, você deve pensar em buscar e apresentar objetos de empréstimo. Por exemplo, a partir de uma perspectiva da API, a guia emprestar no site Kiva.org apresenta uma listagem de empréstimos, e a primeira página do Kiva apresenta um empreendedor por meio de uma busca a um empréstimo que está atualmente captando recursos.
Borrower (mutuário)	Um mutuário é alguém que solicitou um empréstimo. Nós muitas vezes nos referimos a estes indivíduos como "empresários" no Kiva porque acreditamos fortemente no espírito empreendedor dos nossos mutuários que trabalham para fazer a diferença em suas vidas. Um empréstimo pode ter mais de um mutuário, e, neste caso, o empréstimo é considerado um "empréstimo em grupo."
Lender (credor)	Um credor é um usuário registrado no site do Kiva, com a finalidade de emprestar dinheiro e participar na comunidade. A maioria dos credores têm perfis públicos, ou Lender Pages, no Kiva, onde eles podem compartilhar um pouco sobre o que fazem e por que eles emprestam. Nem todos os credores fizeram empréstimos, e nem cada credor optou por exibir informação pública. Credores sem informação pública são referidos como os credores "anônimos".
Partner (parceiro)	Um parceiro, ou Kiva Field Partner, é uma instituição de microcrédito com quem trabalhamos para encontrar e prover empréstimos. Cada empréstimo realizado no Kiva é oferecido por um parceiro para um mutuário e o parceiro trabalha com o

	Kiva para obter financiamento para que o empréstimo com os recursos dos credores. A associação de um empréstimo a um parceiro é muito importante uma vez que o risco associado a um empréstimo correlaciona-se estreitamente com a reputação de um parceiro. É por isso que cada parceiro tem uma classificação de risco.
Journals and Journal Entries (Diário e lançamentos)	Cada empréstimo tem um diário onde o Kiva ou um parceiro pode fornecer atualizações sobre um empréstimo. Cada uma dessas atualizações é chamada de lançamentos. Alguns lançamentos têm fotos ou vídeos que lhes estão associados e fornecem atualizações pessoais sobre o mutuário. Outros são mais informativos ou atualizações automatizadas e são classificados como entradas de diário em lote. Credores podem postar comentários sobre entradas de diário.
Teams (Equipes)	Os credores podem coordenar a sua atividade no Kiva através de equipes de empréstimos, encontrada na guia comunidade, no site do Kiva. As equipes são geralmente formadas em torno de um interesse comum (por vezes relacionadas com empréstimos, às vezes não) ou pode representar grupos do mundo real fora do Kiva, como uma igreja ou uma sala de aula. Atualmente, a maior atividade da equipe está centrada em fazer empréstimos, e cada credor pode creditar um empréstimo para uma equipe cada vez que um empréstimo é feito. Como tal, as equipes estão associadas com ambas as equipes e ocredores. As equipes também têm informações de perfil, semelhantes aos credores, onde eles postam uma foto da equipe e porque a equipe se uniu para realizar empréstimos.

Fonte: Kiva.org

Além desses objetos de dados especializados, a API também utiliza alguns tipos de dados simples que podem ser encontradas em quase todos os bancos de dados: ids, imagens, país, local etc.

5. Usando a API Kiva

A documentação da API Kiva proporciona conhecimento suficiente para gerar requisições HTTP e obter os dados desejados. Foi utilizada a linguagem de programação C# e os programas criados são capazes de fazer chamadas à API, receber respostas corretas e armazenar ou manipular os dados obtidos.

O seguinte código pode ser usado para criar uma requisição HTTP GET e recuperar sua resposta:

```
WebClient w = new WebClient();
string allData = string.Empty;
string buff = "http://api.kivaws.org/v1/loans/newest.json;
allData = w.DownloadString(buff);
```

Na primeira linha do código é criado uma nova instância da classe WebClient, que é uma classe onde se encontram métodos comuns para o envio e recebimento de dados de um recurso especificado em uma URI. A URI é identificada na terceira linha, onde é armazenada em uma *string*. Essa *string* é passada como parâmetro para o método DownloadString da classe WebClient e o recurso retornado como uma *string* e armazenado na variável criada anteriormente.

Uma vez obtido o recurso desejado (uma *string* JSON), pode-se manipulá-lo armazenando o conjunto de dados em um objeto que representa a estrutura desses dados. No exemplo a seguir é criado uma classe RootObject, que representa essa estrutura:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace kivaTest
{
    class RootObject
    {
        public Paging paging { get; set; }
        public List<Loan> loans { get; set; }
    }
}
```

Fonte: Elaborado pelo autor

Essa classe RootObjetc.cs contém a estrutura do recurso que é obtido quando é feita uma requisição para a API do Kiva. O recurso é composto pela paginação e por um objeto raiz, onde serão contidos os dados relevantes do recurso. Para que os dados possam ser manipulados e visualizados, eles precisam ser processados como a seguir:

```
JavaScriptSerializer oJS = new JavaScriptSerializer();
RootObject oRootObject = new RootObject();
oRootObject = oJS.Deserialize<RootObject>(allData);
```

Os dados contidos na string estão serializados e por isso precisam passar por um processo de desserialização para ser usados. Esse processo é feito criando um objeto JavaScriptSerializer e usando seu método Deserialize para armazenar num objeto do tipo RootObject os dados do recurso contidos no parâmetro allData.

Usando paginação

A paginação é usada quando se deseja receber mais de uma página de dados. Muitas vezes os recursos podem exceder o limite de apenas uma página e por isso são divididos em diversas partes para que possam ser enviados de maneira mais eficiente. Esse método ajuda o servidor a responder de forma mais eficiente às requisições de recursos solicitadas. Em C# pode-se utilizar a seguinte classe para representar a paginação fornecida pela API do Kiva:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace kivaTest {
    class Paging
    {
        public int page { get; set; }
        public int total { get; set; }
        public int page_size { get; set; }
        public int pages { get; set; }
    }
}
```

Fonte: Elaborado pelo autor

Essa classe representa os dados de paginação contidos nos dados JSON retornados pelo Kiva. O atributo page corresponde à página atual retornada, total corresponde ao número de recursos (loans, lenders etc) contidos na base de dados para aquela solicitação, page_size é o limite de recursos por página e pages representa o número total de páginas disponíveis. Para acessar recursos que estão divididos em mais de uma página o código anterior pode ser alterado para a seguinte forma:

```
WebClient w = new WebClient();
string allData = string.Empty;
Paging p = new Paging();
int i = 1;

do
{
        string buff = "http://api.kivaws.org/v1/loans/newest.json?per_page=" + per_page + "&page=" + i;
        allData = w.DownloadString(buff);
        JavaScriptSerializer oJS = new JavaScriptSerializer();
        RootObject oRootObject = new RootObject();
        oRootObject = oJS.Deserialize<RootObject>(allData);
        p = oRootObject.paging;

        //código para processar os recursos

        i++;
} while (i <= p.pages);
```

Fonte: Elaborado pelo autor

Nesse exemplo, os recursos são recuperados página por página. O objeto WebClient é criado e seu método DownloadString é chamado para cada página que compõem o recurso requisitado. A *string* que contém a URI é modificada a cada repetição, começando no valor padrão para o parâmetro page=1 e terminando no valor contido no atributo pages do objeto Paging.

Armazenando os dados obtidos

Os dados obtidos com as técnicas citadas anteriormente podem ser armazenados de diferentes formas: como o próprio arquivo JSON recebido, armazenado em um banco de dados, convertido em um arquivo separado por virgulas etc.

A armazenagem dos dados em um banco de dados pode ser feita após o processamento dos recursos obtidos. Os atributos da classe que representa o recurso podem ser relacionados a campos em uma tabela de banco de dados que possui a mesma estrutura dos recursos recuperados. O código mostrado a seguir exemplifica uma forma de gravar em uma tabela os dados do recurso *loans* recuperados através de uma requisição à API do Kiva:

```
private void getLoan(int per_page) {
WebClient w = new WebClient();
string allData = string.Empty;
Paging p = new Paging();
int i = 1;
SqlConnection conn = new SqlConnection();
conn.ConnectionString                                                    =
ConfigurationManager.ConnectionStrings["kivaTest.Properties.Settings.KivaData1
ConnectionString"].ConnectionString;
progressBar1.Minimum = 1;
progressBar1.Step = 1;

try
{
        conn.Open();
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = conn;

        do
        {
                string buff = "http://api.kivaws.org/v1/loans/newest.json?per_page=" +
per_page + "&page=" + i;
                allData = w.DownloadString(buff);
                JavaScriptSerializer oJS = new JavaScriptSerializer();
                RootObject oRootObject = new RootObject();
                oRootObject = oJS.Deserialize<RootObject>(allData);
                p = oRootObject.paging;
                progressBar1.Maximum = oRootObject.paging.total;
```

```
                foreach (Loan lo1 in oRootObject.loans)
                {
                        loanBindingSource.Add(lo1);
                        cmd.CommandText = "select count(id) from loan where id=" +
lo1.id;
                        if ((int)cmd.ExecuteScalar() == 0)
                        {
                                string[] coord = lo1.location.geo.pairs.Split(' ');
                                cmd.CommandText = "SET ANSI_WARNINGS OFF
insert into loan (id, name, status, funded_amount, basket_amount, activity, sector,
theme, use1, partner_id, posted_date, planned_expiration_date, loan_amount,
borrower_count, lender_count, bonus_credit_eligibility, city, country, latitude,
longitude) VALUES (@id, @name, @status, @funded_amount, @basket_amount,
@activity, @sector, @theme, @use1, @partner_id, @posted_date,
@planned_expiration_date, @loan_amount, @borrower_count, @lender_count,
@bonus_credit_eligibility, @city, @country, @latitude, @longitude);";
                                SqlParameter param = new SqlParameter();
                                param.ParameterName = "@id";
                                param.Value = lo1.id;
                                cmd.Parameters.Add(param);
                                param = new SqlParameter();
                                param.ParameterName = "@name";
                                param.Value = lo1.name.Replace("'","''");
                                cmd.Parameters.Add(param);
                                param = new SqlParameter();
                                param.ParameterName = "@status";
                                param.Value = lo1.status;
                                cmd.Parameters.Add(param);
                                param = new SqlParameter();
                                param.ParameterName = "@funded_amount";
                                param.Value = lo1.funded_amount;
                                cmd.Parameters.Add(param);
                                param = new SqlParameter();
                                param.ParameterName = "@loan_amount";
                                param.Value = lo1.loan_amount;
                                cmd.Parameters.Add(param);
                                cmd.ExecuteNonQuery();
                                cmd.Parameters.Clear();
                        }
                }
                i++;
        } while (i <= p.pages);
}
```

Fonte: Elaborado pelo autor

O método *getLoan* incorpora os exemplos anteriores, fazendo uso da paginação para dividir o recurso solicitado em fragmentos que são recebidos e processados. Em cada repetição, a URI usada na chamada à API é modificada pela página que ser quer obter e os dados são processados e enviados para uma tabela em um banco de dados. Essa tabela contém campos que correspondem aos dados obtidos através do arquivo JSON recuperado. Além dos objetos vistos anteriormente, nesse exemplo são criados também um objeto do tipo SqlConnection, que é responsável pela conexão com o banco de dados (SQL server no exemplo). Essa conexão é aberta utilizando-se a string de conexão apropriada e fica disponível para que possa ser usada pela aplicação. Os comandos para o banco de dados utilizam a linguagem SQL e são passados para o banco de dados através do atributo CommandText do objeto da classe SqlCommand criado.

Para a gravação no banco de dados é passado um comando SQL composto pelas instruções para a gravação na tabela e campos apropriados e os valores são passados em forma de parâmetros. Esses parâmetros são compostos pelos dados contidos nos atributos do objeto que representa a classe *loan*. O formato geral do comando SQL é o seguinte:

```
"insert into TABELA (campos da tabela) VALUES (parâmetros SQL);"
```

Os parâmetros são adicionados ao comando SQL e contém os atributos do objeto *loan* representado no momento. É criado um objeto do tipo SqlParameter para armazenar o parâmetro a ser adicionado. Esse parâmetro possui um atributo chamado ParameterName, que é usado para fornecer o nome do parâmetro a ser usado no comando SQL final. O valor do recurso atualmente processado é conferido ao atributo Value. Após essa operação, o parâmetro completo é adicionado aos parâmetros do comando SQL:

```
SqlParameter param = new SqlParameter();
param.ParameterName = "@nome do parâmetro";
param.Value = recursoprocessado.atributo;
cmd.Parameters.Add(param);
```

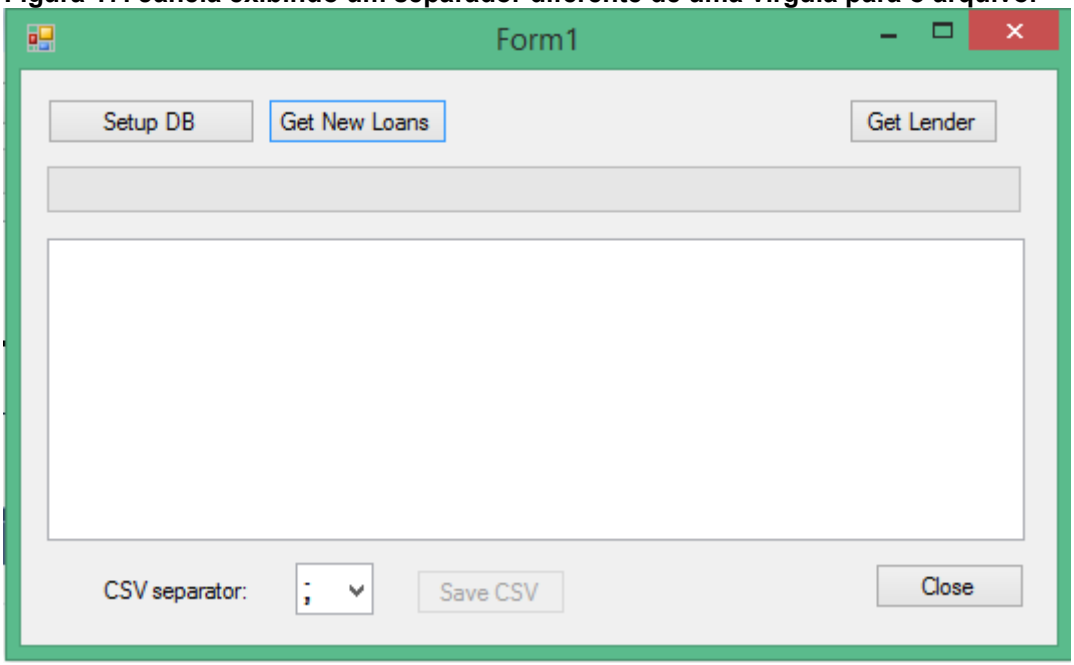
Uma outra forma de armazenar os dados obtidos é salvando-os como um arquivo separado por vírgula. O método exibido a seguir fornece uma técnica para isso:

```
private void saveToCSV() {
        string fileName = "";
        SaveFileDialog saveFile = new SaveFileDialog();
        saveFile.Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*";
        saveFile.FilterIndex = 1;
        saveFile.RestoreDirectory = true;

        if (saveFile.ShowDialog() == DialogResult.Cancel)
                return;

        fileName = saveFile.FileName;

        string separator = comboBox1.Text;

        using (StreamWriter theWriter = new StreamWriter(fileName == "" ? "c:\\test.csv" : fileName))
        {
                theWriter.WriteLine("ID" + separator + "Name" + separator + "Status" + separator + "Funded Amount" + separator + "Basket Amount" + separator + "Activity" + separator + "Sector" + separator + "Partner ID" + separator + "Posted Date" + separator + "Loan Amount" + separator + "Lender Count");
                foreach (Loan loan1 in loanBindingSource)
                {
                        theWriter.WriteLine(loan1.id + separator);
                        theWriter.WriteLine(loan1.name + separator);
                        theWriter.WriteLine(loan1.status + separator);
                        theWriter.WriteLine(loan1.funded_amount + separator);
                        theWriter.WriteLine(loan1.basket_amount + separator);
                        theWriter.WriteLine(loan1.activity + separator);
                        theWriter.WriteLine(loan1.sector + separator);
                        theWriter.WriteLine(loan1.partner_id + separator);
                        theWriter.WriteLine(loan1.posted_date + separator);
                        theWriter.WriteLine(loan1.loan_amount + separator);
                        theWriter.WriteLine(loan1.lender_count + separator);
                }
        }
}
```

Fonte: Elaborado pelo autor

Esse exemplo, criado para o sistema Windows, utiliza as caixas de diálogo padrão do sistema para que o nome do arquivo seja escolhido. O nome retornado pela caixa de diálogo é então usado para a criação do arquivo de texto separado por vírgula. Apesar do nome, esse tipo de arquivo pode usar qualquer caractere para separação. E essa escolha é feita através de uma comboBox no exemplo:

Figura 17: Janela exibindo um separador diferente de uma vírgula para o arquivo.



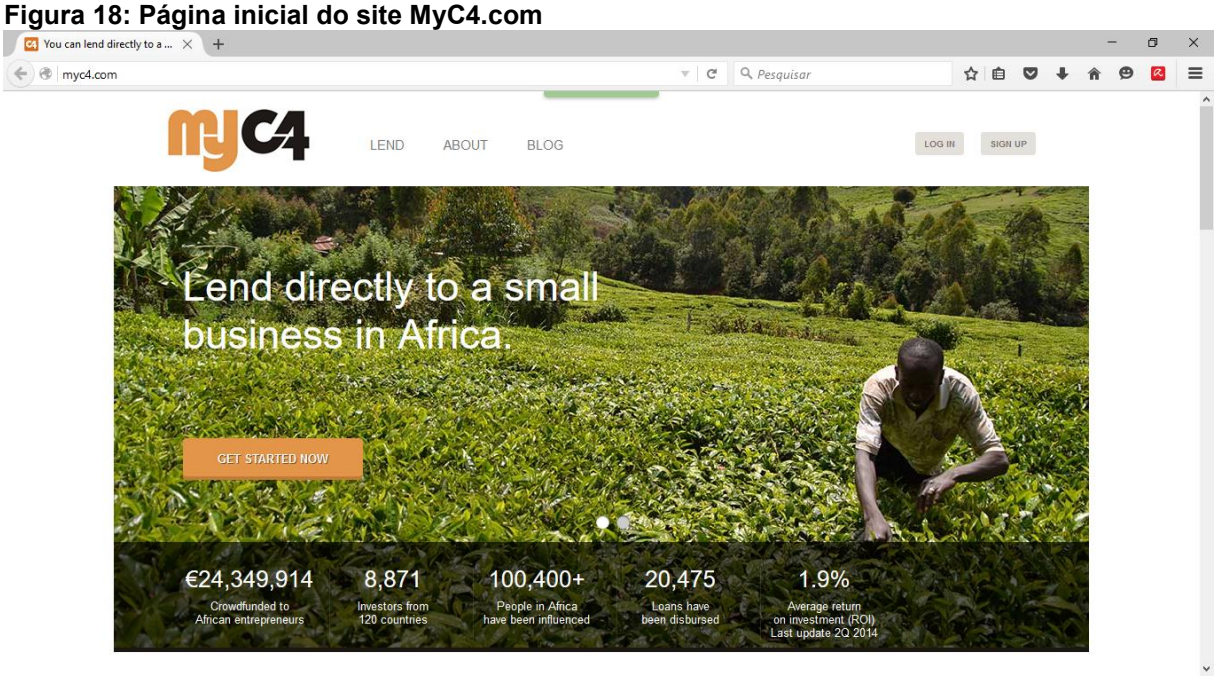
Fonte: Elaborado pelo autor

O arquivo é gerado usando uma instância da classe StreamWriter com o nome do arquivo como parâmetro. Esse objeto possui o método WriteLine que é responsável por escrever uma linha de texto no arquivo especificado. No exemplo acima os dados escritos no arquivo são obtidos do objeto loanBindingSource que fornece dados da tabela *loan* no banco de dados.

Obtendo dados de uma página HTML

Os dados do Kiva são obtidos através de uma requisição HTTP que responde com um arquivo JSON ou XML. Esses arquivos facilitam o trabalho de manipulação e armazenagem de dados. O processo de requisição de dados é feito através de uma API criada especificamente para isso. Outros sites não fornecem esse tipo de ferramenta. No exemplo a seguir é mostrado como são obtidos os dados do site myc4.com. Os dados são recuperados diretamente da página HTML através da busca de determinadas marcações no código da página.

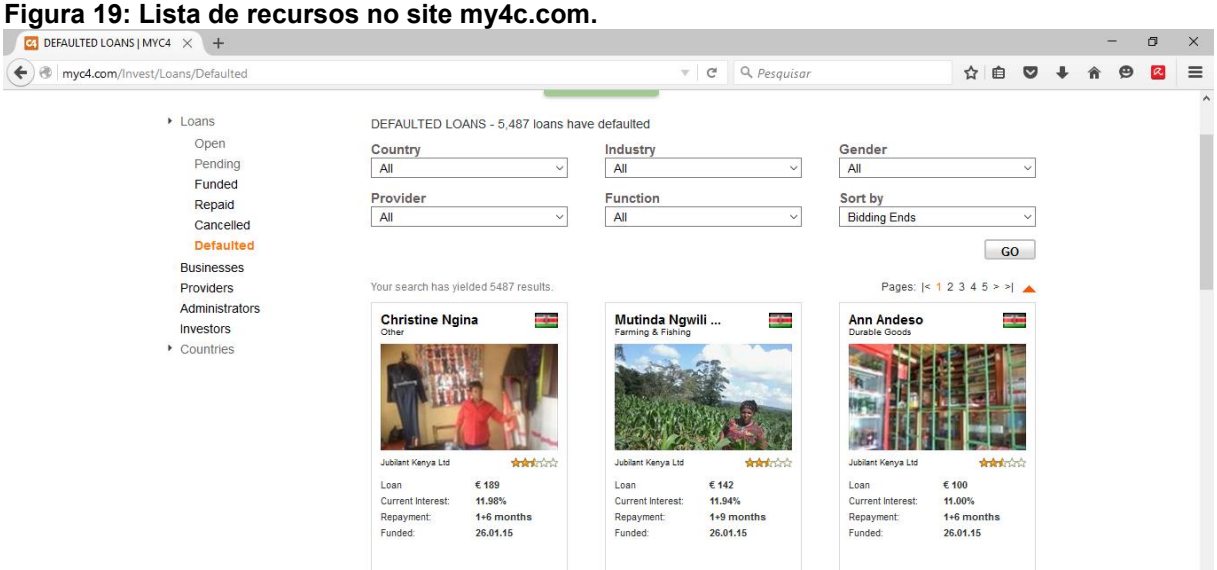
Figura 18: Página inicial do site MyC4.com



Fonte: myc4.com

O site mostrado na figura 18 é a única fonte para se obter os dados dessa organização. Por não possuir uma ferramenta específica para isso, é necessário que os dados sejam retirados de páginas HTML comuns. Essa extração de dados é feita percorrendo-se o arquivo em busca de marcações HTML que contém os dados desejados. Todos os dados são apresentados como uma página HTML.

Figura 19: Lista de recursos no site my4c.com.



Fonte: myc4.com

Figura 20: Fragmento de código da página anterior. Os dados relevantes estão exibidos em uma tabela

```
<!-- Avatar -->
<div id="ctl00_ctl00_Middle_Main_uiDataList_ctl09_uiAvatarImage" class="main-img" style="background-ima

</div>

<!-- Star rating -->
<div class="rating">
    <span class="company">
        <a id="ctl00_ctl00_Middle_Main_uiDataList_ctl09_uiProviderNameHyperLink" title="Jubilant Kenya
    <span id="ctl00_ctl00_Middle_Main_uiDataList_ctl09_uiRiskRating" title="Risk Rating" class="rate"><
</div>

<dl>
    <dt>Loan</dt>
    <dd>&#8364; 189</dd>



        <dt>Current Interest:</dt>
        <dd>
            12.00%</dd>

    <dt>Repayment:</dt>
    <dd>1+6 months</dd>
    <dt style="display: none;">Investors:</dt>
    <dd style="display: none;">
        0</dd>
    <dt>
        Funded:</dt>
    <dd>
        26.01.15

    </dd>
</dl>
```

Fonte: myc4.com

Para que os dados contidos na tabela da página HTML mostrada anteriormente possam ser usados, eles precisam ser extraídos e armazenados de forma apropriada. Uma forma de se fazer isso é analisar toda a estrutura da página, buscando as marcações HTML apropriadas onde os dados estão contidos.

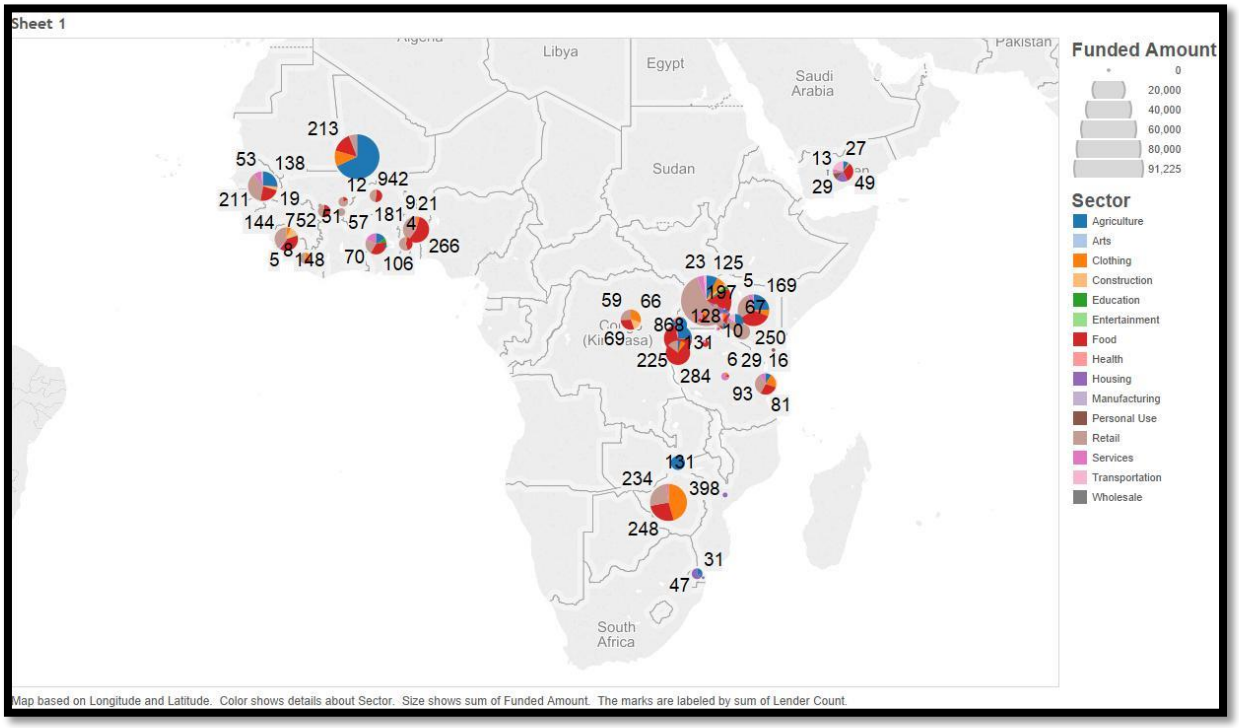
A página que será processada é obtida da mesma forma utilizada com a API do Kiva. Um objeto WebClient é criado usando seu método DownloadString para armazenar o resultado como texto. Esse texto é o código fonte da página HTML. Com esse código disponível, é possível percorrê-lo em busca dos padrões de marcação que se deseja. O fragmento de código que exemplifica esse processo se encontra no apêndice deste trabalho.

No exemplo citado é criada uma lista para armazenar os dados contidos na tabela da página HTML. Os dados estão contidos dentro de marcações específicas. O método percorre o arquivo em busca dessas marcações e extrai o seu conteúdo. Geralmente a quantidade de informações é muito grande para ser exibida em apenas uma página HTML, e também nesse caso é usada a paginação para dividir a informação em várias partes.

Com os dados extraídos da página, o processo de armazenamento é similar ao usado com a API do Kiva. Os dados podem ser armazenados em um banco de dados, salvos em um arquivo apropriado ou exibidos diretamente. A seguir alguns exemplos de visualizações dos dados obtidos utilizando o *software* Tableau, que é capaz de se conectar diretamente a um banco de dados ou processar arquivos com dados separados por vírgula:

A figura 21 apresenta um mapa onde aparecem alguns países que possuem empréstimos realizados. Os dados obtidos através das técnicas demonstradas acima foram utilizados para exibir a quantidade de empréstimos por país e os setores da economia a que esses empréstimos se destinam.

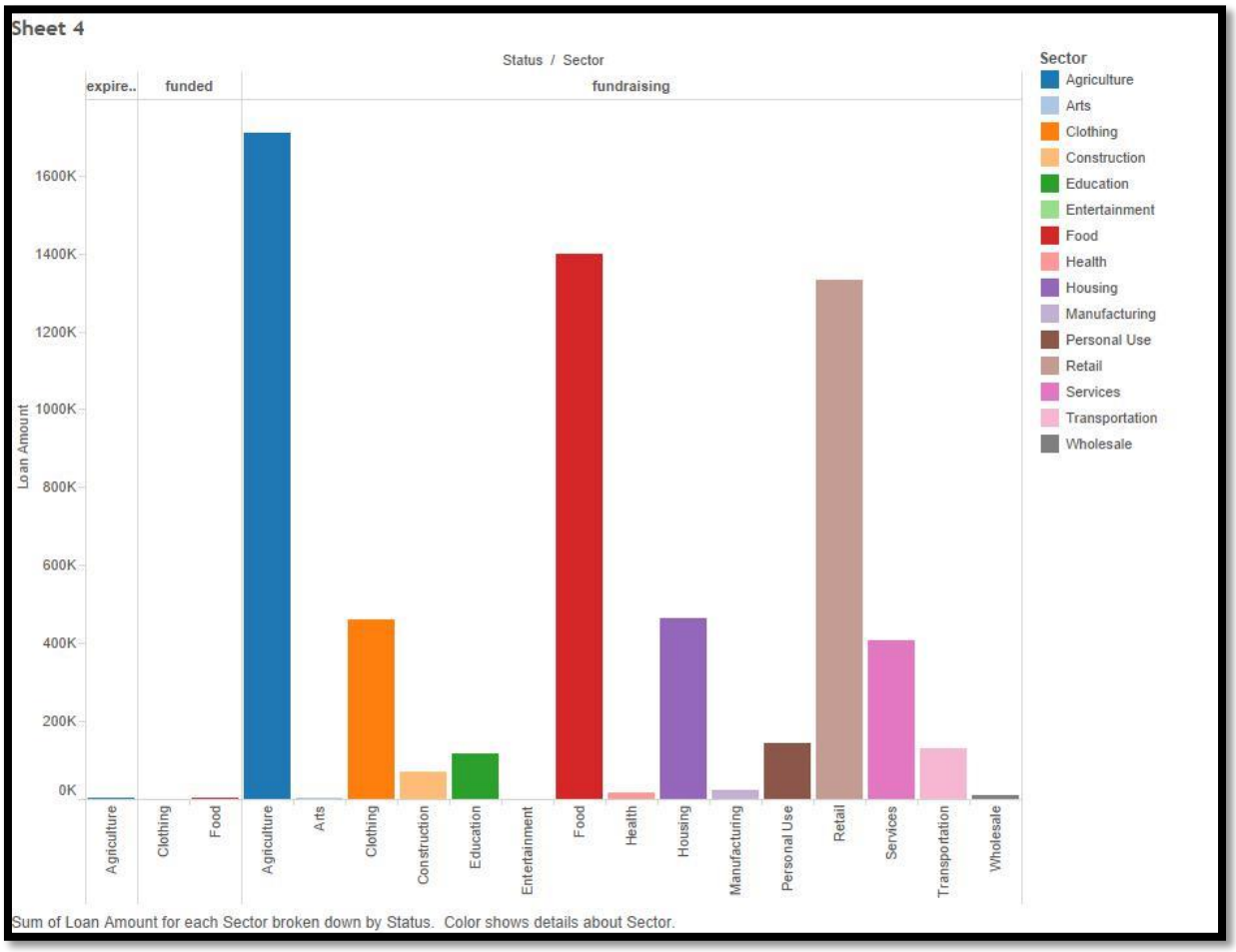
Figura 21: Visualização de dados por setor



Fonte: Elaborada pelo autor

Na figura 22 foram usados os mesmos dados para mostrar um gráfico diferente, que mostra o montante dos empréstimos por setor. As visualizações geradas são flexíveis e podem ser adaptadas para exibir apenas as informações que se julgar importantes. Nesse caso, a divisão por países foi omitida do gráfico e foi enfatizado apenas o montante dos empréstimos.

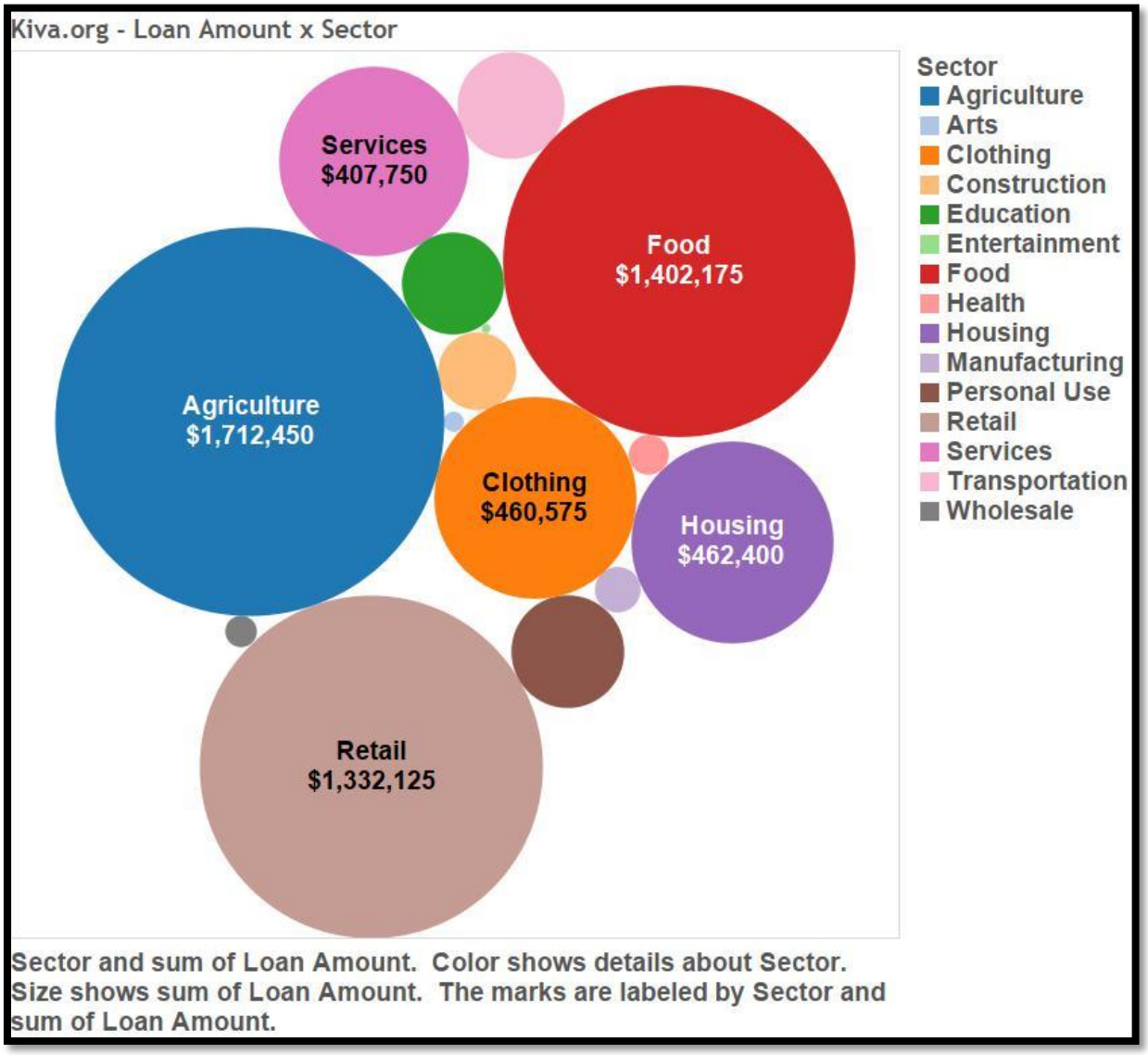
Figura 22: Quantidade de empréstimos por setor



Fonte: Elaborada pelo autor

Uma outra visualização dos dados obtidos através da API do Kiva. Na figura 23 o montante dos empréstimos determina o tamanho dos círculos que são codificados por cores que representam os diversos setores da economia cadastrados no sistema.

Figura 23: Uma outra forma de exibir os dados de empréstimos por setor



Fonte: Elaborada pelo autor

6. Considerações Finais

Conclui-se que o protocolo HTTP pode ser uma forma eficiente de se obter dados remotos. Utilizado normalmente por navegadores web para a recuperação e apresentação de páginas HTML, esse protocolo foi usado nesse trabalho para se conectar a servidores e se obter documentos JSON contendo dados. Com a ajuda de uma API o acesso aos dados foi feito de forma rápida e eficiente. Nesse trabalho ele também foi utilizado de forma mais convencional para se obter uma página HTML e os dados contidos nessa página foram extraídos através da análise de sua estrutura. Este é um método não muito eficiente, pois se baseia na estrutura da página. Se a empresa que disponibiliza essa página resolver mudar sua estrutura, todo o trabalho deve ser refeito.

Além da parte técnica da obtenção e manipulação dos dados, durante essa pesquisa foi demonstrado como funciona uma empresa de microfinanciamento. O serviço prestado por esse tipo de empresa é muito beneficiado pela Internet, tornando seu alcance e visibilidade muito maiores.

O uso de uma API específica possibilita ao Kiva fornecer acesso fácil e rápido aos dados referentes às suas atividades. Isso proporciona transparência, o que gera confiança dos interessados em investir nessa empresa. E para os desenvolvedores de aplicativos, a documentação fornecida juntamente com a API possibilita o desenvolvimento rápido e confiável de sistemas que podem acessar toda a base de dados da empresa.

O estudo da segunda empresa (myc4.com) pode ser usado em trabalhos futuros, pois a informação é disponibilizada diretamente em uma página HTML. Conhecendo a estrutura das páginas e a técnica para extrair os dados relevantes, pode-se ter acesso aos dados de qualquer empresa que disponibiliza esse tipo de visualização.

As técnicas apresentadas nesse trabalho possibilitam a qualquer pessoa interessada uma forma de acesso fácil a dados, sem a necessidade de aplicativos específicos ou sistemas de gerenciamento de banco de dados. Os dados são fornecidos através do protocolo HTTP, o que os torna acessíveis com poucas linhas de código e a resposta gerada é fornecida em um formato simples, como o JSON, XML ou HTML.

REFERÊNCIAS BIBLIOGRÁFICAS

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos:** Conceitos e projeto. Tradução de João Tortello. 4ª. ed. Porto Alegre: Bookman, 2007.

ECMA INTERNATIONAL. Standard ECMA-404. **Ecma International**, 2013. Disponivel em: <http://www.ecma-international.org/publications/standards/Ecma-404.htm>. Acesso em: 24 outubro 2015.

HAAS, H. Reconciling Web Services and REST Services. **w3.org**, 2005. Disponivel em: <http://www.w3.org/2005/Talks/1115-hh-k-ecows/#%281%29>. Acesso em: 15 outubro 2015.

JSON.ORG. Introdução ao JSON. **JSON.org**, 2015. Disponivel em: <http://json.org/json-pt.html>. Acesso em: 26 outubro 2015.

KIVA MICROFUNDS. About Us. **Kiva - Loans that change lives**, 2015. Disponivel em: <http://www.kiva.org/about>. Acesso em: 20 outubro 2015.

KIVA MICROFUNDS. build.kiva: The Home for Kiva Developers. **Kiva - Loans that change lives**, 2015. Disponivel em: <http://build.kiva.org/>. Acesso em: 18 outubro 2015.

W3C. Extensible Markup Language (XML). **w3.org**, 2015. Disponivel em: <http://www.w3.org/XML/>. Acesso em: 25 outubro 2015.

W3SCHOOLS. XML Tutorial. **w3schools**, 2015. Disponivel em: <http://www.w3schools.com/xml/default.asp>. Acesso em: 26 outubro 2015.

Apêndice

Código usado para recuperar dados contidos em um documento através da procura de padrões de marcações HTML:

```
        public List<string> getData(string type)
        {
         List<string> lsNames = new List<string>();
         List<string> lsCountry = new List<string>();

         List<string> lsLoans = new List<string>();
         List<string> ls = new List<string>();

         WebClient webClient = new WebClient();
         string page = webClient.DownloadString("http://myc4.com/Invest/Loans/" + type + "?page=1");

         HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();
         doc.LoadHtml(page);

         var coNameNode = doc.DocumentNode.SelectSingleNode("//ul");
         var cN = coNameNode.SelectNodes("//li//div//div//a");
         var cCountry = coNameNode.SelectNodes("//li//div//div//img");
         var node = doc.DocumentNode.SelectSingleNode("//dl");
         var nodec = node.SelectNodes("//dd");
         int x = (doc.DocumentNode.SelectNodes("//dl//dd").Count / doc.DocumentNode.SelectNodes("//dl").Count);
         int last = 1;
```

```
        string tmp = "";


        try
        {
                last = Convert.ToInt32(doc.DocumentNode.SelectSingleNode("//a[@title='Last']").Attributes["href"].Value.Split('=')[1]);


        }
        catch (Exception)
        {


                last = 1;
        }


        char separator = '\t';
        Console.WriteLine();
        Console.WriteLine("Processing {0} Loans...", type);


        for (int k = 1; k <= last; k++)
        {
                Console.WriteLine("Loading data from page {0} of {1} - {2} Loans", k, last, type);


        doc.LoadHtml(webClient.DownloadString("http://myc4.com/Invest/Loans/" + type + "?page=" + k));
                coNameNode = doc.DocumentNode.SelectSingleNode("//ul");
```

```
cN = coNameNode.SelectNodes("//li//div//div//a");
cCountry = coNameNode.SelectNodes("//li//div//img[@alt='Flag']");

node = doc.DocumentNode.SelectSingleNode("//dl");
nodec = node.SelectNodes("//dd");
int i = 1;

foreach (var item in cN)
{

        try
        {
                tmp += item.Attributes["title"].Value;
        }

        catch (Exception)
        {
                tmp += "none";
        }

        if (i % 2 == 0)
        {
                lsNames.Add(tmp);
                tmp = "";
        }
        else
```

```
                tmp += separator;
        i++;
}

tmp = "";
i = 1;

foreach (var item in cCountry)
{
        try
        {
                lsCountry.Add(item.Attributes["title"].Value);
        }
        catch (Exception)
        {
                lsCountry.Add("none");
        }

}

foreach (var item in nodec)
{
        tmp += item.InnerText.Trim().Replace("&#8364; ", "");
        if (i % x == 0)
        {
                lsLoans.Add(tmp);
```

```
                    tmp = "";
                }
                else
                    tmp += separator;
                i++;
            }
        }

        for (int j = 0; j < lsLoans.Count; j++)
        {
            ls.Add(lsNames[j] + "\t" + lsCountry[j] + "\t" + lsLoans[j]);
            Console.WriteLine("Added item {0} of {1} - {2} Loans", j + 1,
lsLoans.Count, type);
        }
        return ls;
}
```

Fonte: Elaborado pelo autor