UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

RODRIGO DE OLIVEIRA FERNANDES

DESENVOLVIMENTO DE UMA PLATAFORMA DE *BOOKMARKING* SOCIAL COM DJANGO E TDD

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2015

RODRIGO DE OLIVEIRA FERNANDES

DESENVOLVIMENTO DE UMA PLATAFORMA DE *BOOKMARKING* SOCIAL COM DJANGO E TDD

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – TADS – Departamento Acadêmico de Computação – Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. M.Sc. Ricardo Sobjak

MEDIANEIRA

2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Desenvolvimento de uma plataforma de *bookmarking* social com Django e TDD

Por

Rodrigo de Oliveira Fernandes

Este Trabalho de Diplomação (TD) foi apresentado às 08:20h do dia 12 de junho de 2015 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. O acadêmico foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof.M.Sc. Ricardo Sobjak
UTFPR – Câmpus Medianeira
(Orientador)

Prof.ª Alessandra Bortoletto Garbelotti Hoffmann
UTFPR – Câmpus Medianeira
(Convidado)

Prof. Valter Rodrigo Ekert
UTFPR – Câmpus Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb
UTFPR – Câmpus Medianeira
(Responsável pelas atividades de TD)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Meus sinceros agradecimentos a todos que de alguma forma colaboraram para a concretização de cada etapa rumo a esta grande conquista pessoal.

Aos meus pais, Vera e Joaquim, meu irmão Cristiano, e demais familiares, amigos próximos, pelo apoio constante nas mais variadas formas.

A minha esposa, Mariana, pela alegria de sua presença em minha vida, partilhando sonhos e realizações.

Aos professores com os quais aprendi a buscar o aprendizado constante a partir do amor dedicado naquilo que se faz.

Dedico especial agradecimento ao professor Ricardo Sobjak, orientador deste trabalho de diplomação, que me conduziu com disposição para a realização de um importante objetivo.

A todos os colegas de classe presentes na minha jornada educacional.

Agradeço profundamente àqueles que contribuíram diretamente ou indiretamente para que este momento fosse possível. Também a todos com quem posso compartilhar a alegria desta conquista.

Obrigado a todos.

RESUMO

FERNANDES, Rodrigo de Oliveira. Desenvolvimento de uma plataforma de *bookmarking* social com Django e TDD. 2014. 99f. Trabalho de diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas). Universidade Tecnológica Federal do Paraná. Medianeira 2014.

Com o surgimento da Web e o crescente volume de informações disponíveis, emerge a necessidade de armazenar hiperlinks, a fim de facilitar o acesso posterior. Os primeiros navegadores já ofereciam o recurso, que foi chamado de *bookmarks* ou favoritos. A partir da ascensão da Web 2.0, foram criados diversos serviços de *bookmarking* social, permitindo que comunidades de usuários guardassem links prediletos em plataformas coletivas. Alguns destes serviços ofereciam a possibilidade de organizar links por meio de etiquetas, que em ambiente social, resulta em uma nova forma de organizar informações, a folksonomia. Este trabalho visa à construção de uma plataforma de *bookmarking* social, utilizando o *framework* Django e desenvolvimento guiado por testes (TDD) para a construção do sistema.

Palavras-chave: Desenvolvimento web. Desenvolvimento guiado por testes. Folksonomia.

ABSTRACT

FERNANDES, Rodrigo de Oliveira. Developing a social bookmarking platform with Django and TDD. 2014. 99f. Trabalho de diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas). Universidade Tecnológica Federal do Paraná. Medianeira 2014.

With the appearance of the Web and the increasing volume of available information emerges the necessity of a way to store user hyperlinks in order to facilitate subsequent access. Early browsers already offer the feature, which was named bookmarks or favorites. Since the rise of the Web 2.0, various social bookmarking services appears, allowing user communities to store their favorite links in a collective environment. Some of these services offer the ability to organize the links through tags, which in a social environment results in a new way of organizing information: the folksonomy. This paper is seeking to build a social bookmarking platform, with Django Web framework and Test-driven Development (TDD) for the system construction.

Keywords: Web development. Test-driven development. Folksonomy.

LISTA DE SIGLAS E ABREVIATURAS

ACID	*Atomicity, Consistency, Isolation, Durability*
API	*Application Programming Interface*
APP	*Application*
BBS	*Bulletin Board System*
BSD	*Berkeley Software Distribution*
CSS	*Cascading Style Sheets*
CERN	European Organization for Nuclear Research
CRUD	*Create, Read, Update and Delete*
DRY	*Don´t Repeat Yourself*
DSF	Django Software Foundation
GIS	*Geographic Information System*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IOS	Iphone *Operating System*
MTV	*Model-Template-View*
MVC	*Model-View-Controler*
NCSA	National Center for Supercomputing Applications
ORM	*Object-Relational Mapping*
PHP	Hypertext Preprocessor
Pip	Pip Install (Python) Packages
PSF	Python Software Foundation
PyPI	Python Package Index
RSS	*Rich Site Sumary*

SQL	*Structured Query Language*
SMS	*Short Message Service*
TDD	*Test-Driven Development*
UGC	*User Generated Content*
UI	*User Interface*
URL	*Uniform Resource Locator*
XP	*Extreme Programming*
WWW	World Wide Web

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Na ocasião em que Tim Berners-Lee e Robert Cailliau concretizaram a ideia do hipertexto a partir do desenvolvimento da Web, surge a necessidade de organizar as informações da rede hipertextual. A experiência bem sucedida da Web e o iminente crescimento no volume de informações publicadas levam Tim Berners-Lee a iniciar o projeto chamado “The WWW Virtual Library”, como tentativa de indexar o conteúdo disponível na Internet, o qual se torna o primeiro catálogo de hiperlinks da Web.

O Mosaic, primeiro navegador que obteve destaque pelo uso popular, apresentava um recurso chamado “Hotlists” no qual os usuários poderiam guardar e organizar seus links prediletos, a fim de facilitar o acesso posterior. O Netscape (desenvolvido a partir do Mosaic) aparece mais tarde com um sistema semelhante chamado “Bookmarks”, fazendo referência a marcadores de livros, marca-páginas. A Microsoft com o Internet Explorer, também cria um sistema com o mesmo intuito e o chama de “Favoritos”. O sistema de armazenamento de links passa a ser comum e disponível na maioria dos navegadores. Dessa maneira, o usuário poderia organizar sua coleção de links da Web em seu próprio computador.

A popularização do acesso a Internet e o surgimento de ferramentas que facilitaram a contribuição dos usuários na geração de conteúdo, fizeram da Web um ambiente propício para a disseminação de informações, e neste mesmo contexto surge o *bookmarking* social, atividade na qual os usuários “salvam” seus links favoritos coletivamente em um sistema on-line.

Além de um meio de armazenar os hiperlinks, os serviços de *bookmarking* social normalmente oferecem formas de organizar as informações, como os sistemas de “etiquetagem”, que em ambiente social caracterizam um fenômeno que ficou conhecido como folksonomia.

A partir dos estudos deste trabalho, busca-se o embasamento para a elaboração de uma plataforma de *bookmarking* social utilizando o TDD – *Test-Driven Development* (desenvolvimento guiado por testes), com o Django, um *framework* para desenvolvimento Web escrito com a linguagem Python.

1.1 OBJETIVO GERAL

Elaborar uma plataforma de *bookmarking* social utilizando o *framework* Django e o desenvolvimento guiado por testes.

1.2 OBJETIVOS ESPECÍFICOS

- Realizar uma pesquisa sobre os conceitos associados ao *bookmarking* social;
- Apresentar os conceitos relacionados ao desenvolvimento guiado por testes;
- Apresentar o *framework* Django e demais tecnologias utilizadas no trabalho;
- Desenvolver uma aplicação Web baseando-se nas tecnologias, práticas e conceitos estudados;
- Apresentar a aplicação desenvolvida e os resultados obtidos.

1.3 JUSTIFICATIVA

A popularização de tecnologias que de algum modo promovem a participação do usuário, vêm favorecendo a produção crescente de conteúdo na Web. Dessa maneira, o volume de dados disponíveis cresce a cada dia. Associada a natureza desordenada, de fluxo livre, do hipertexto, observam-se novos desafios no processo de organização e busca da informação.

Neste contexto surgem os mecanismos de busca como ferramentas frequentemente utilizadas. No entanto a eficiência e legitimidade dos buscadores podem ser questionáveis, por utilizarem alguns critérios ocultos e controversos para relacionar os resultados, o que dá margem ao crescimento de uma opção de conceito transparente, guiada pela colaboração entre os próprios usuários.

Destaca-se o uso da folksonomia como uma alternativa na qual os usuários têm o poder de indexar o conteúdo que consideram relevantes na Web. Folksonomia é a

tradução do termo folksonomy que é um neologismo criado em 2004 por Thomas Vander Wal, a partir da junção dos termos *folk* (povo, pessoas) e taxonomia. Caracteriza-se pelo resultado da atribuição livre e pessoal de etiquetas a recursos da Web, realizada num ambiente social, compartilhado, e aberto a outros.

O *bookmarking* social e a folksonomia possibilitam que a movimentação dos indivíduos pela rede seja revertida em favor de todos. Os usuários podem organizar as informações colaborativamente e armazená-las coletivamente, ou seja, todos contribuem para o sistema, mas cada um tem a liberdade para organizar as informações de acordo com seus próprios critérios. Esta ideia contrasta com os processos de indexação "robotizados" dos mecanismos de busca. Com o *bookmarking* social e a folksonomia, além de encontrar conteúdo, pessoas podem encontrar pessoas, partilhar interesses e estabelecer relacionamentos.

O uso do *framework* Django é uma opção viável no desenvolvimento da plataforma proposta, pois oferece uma estrutura robusta para a criação de aplicações dinâmicas baseadas na Web. O TDD – *Test-Driven Development* (desenvolvimento guiado por testes), visa à melhoria na qualidade do código e, ao mesmo tempo, oferece agilidade ao processo de desenvolvimento de software.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho possui seis capítulos, sendo que o primeiro trata da contextualização do tema abordado, definindo os objetivos e a justificativa do projeto. O segundo busca fornecer referencial teórico sobre os conceitos que embasam o desenvolvimento da plataforma. O capítulo três contempla a apresentação dos materiais e métodos empregados no desenvolvimento do sistema, relacionando as características principais de cada uma delas. O capítulo quatro demonstra os resultados obtidos com o desenvolvimento do sistema, a partir dos conceitos, tecnologias e metodologias estudados. As considerações finais sobre o projeto e as sugestões para trabalhos futuros são abordadas no capítulo cinco. Por fim, no capítulo seis, estão as referências bibliográficas utilizadas para a elaboração do documento.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta conceitos relevantes para a compreensão dos objetivos propostos e que embasam o desenvolvimento do projeto.

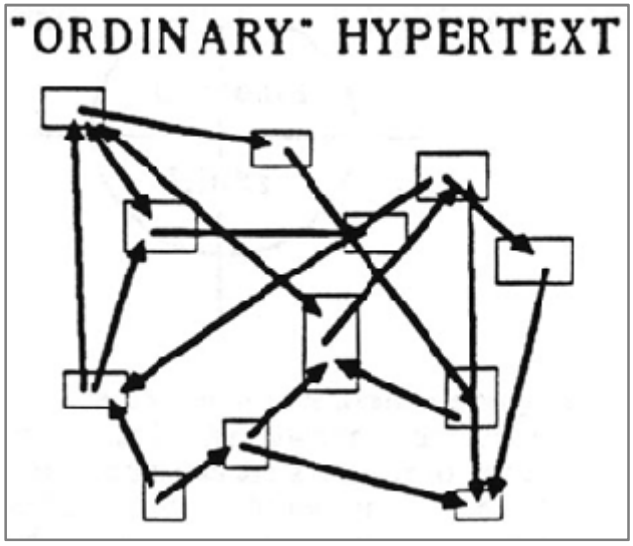
2.1 A WEB E O HIPERTEXTO

Na década de 80, Tim Berners-Lee desenvolveu um estudo denominado Enquire, cujo objetivo era descobrir uma maneira de interligar informações e documentos em seu próprio computador. Em essência, tratava-se de um sistema em que palavras de um arquivo digital pudessem oferecer acesso a outros arquivos (RIGBY, 2012).

O trabalho de Berners-Lee estava relacionado à ideias e projetos técnicos que, cerca de meio século antes, buscavam a possibilidade de associar fontes de informação por meio da computação interativa. Vannevar Bush no artigo “As We May Think” (BUSH, 1945), propôs o sistema Memex em 1945. Douglas Engelbart exibiu o On-Line *System* em 1968. Ted Nelson elaborou o conceito de “hipertexto” em seu manifesto de 1963, e trabalhou muitos anos na criação de um projeto utópico denominado Xanadu (CASTELLS, 2003).

O hipertexto, representado na figura 1, mostra-se como uma enciclopédia feita por diversos autores, sem um projeto anterior, sem sumário, índice ou numeração de páginas, com características de um “labirinto virtual”. A estrutura livre de organização formal e as múltiplas interconexões fazem com que seja pouco provável que se percorra um mesmo “caminho” ao navegar. Este aspecto, também pode representar um ambiente rico em variações e surpresas, propício a novas descobertas (FILHO, 2003).

Figura 1 - Diagrama de hipertexto de Ted Nelson.

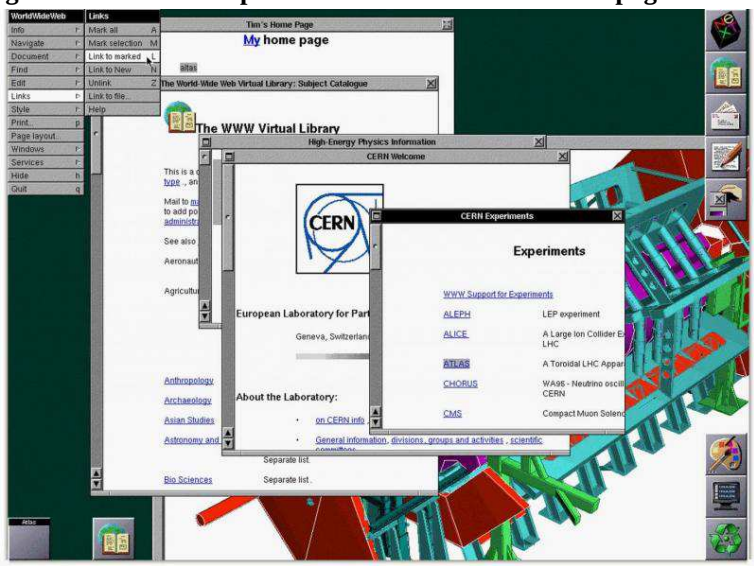


Fonte: W2VR (2000).

Em 1989, Tim Berners-Lee com o auxílio do engenheiro Robert Cailliau na Organização Européia para Pesquisa Nuclear (CERN), trabalharam para melhorar e compartilhar o sistema que Berners-Lee havia iniciado há alguns anos com o Enquire. A rede hipertextual foi ativada sobre a infraestrutura da Internet, dando origem a World Wide Web (WWW, ou simplesmente Web) em 1990 (AQUINO, 2006).

Na figura 2, a tela do computador de Tim Bernes-Lee em 1993 com algumas páginas Web:

Figura 2 - Tela do computador de Tim Bernes-Lee com páginas Web.



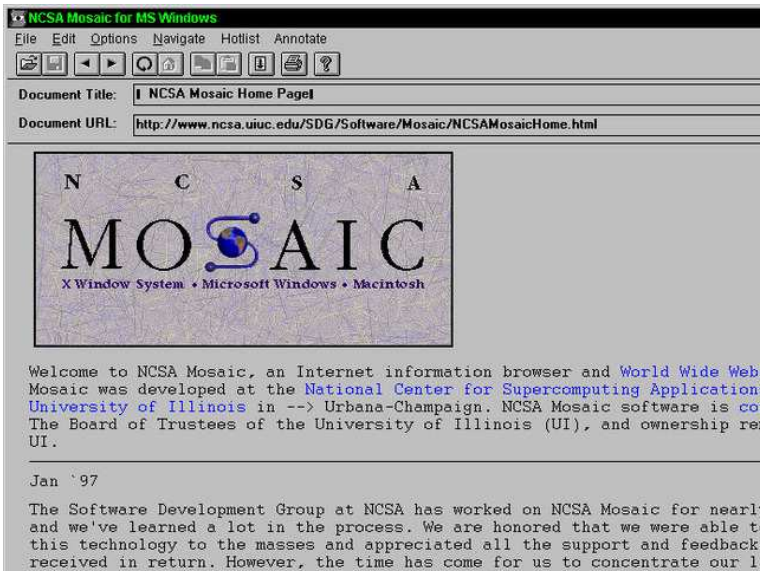
Fonte: CERN (2014).

O desenvolvimento da WWW foi realizado de acordo com um desenho composto por elementos fundamentais, dos quais tem destaque: o servidor Web, o "*browser*", os *Uniform Resource Locators* (URLs), a linguagem *HyperText Markup Language* (HTML) e o *HyperText Transfer Protocol* (HTTP).

A linguagem HTML previa uma forma de conexão entre documentos: o hiperlink (ou simplesmente link), ligação "clicável" que permitiria o acesso a outro recurso a partir de uma URL (*Uniform Resource Locator*), sequência de caracteres utilizados para localizar um recurso na Internet, também conhecido como "endereço Web" (NUNES, 2011).

Com a popularização e a consolidação da Web, os documentos e *links* começam se multiplicar. Diversos navegadores surgem, sendo que o primeiro a obter destaque foi o Mosaic (figura 3), lançado por Mark Andreessen e sua equipe na NCSA (National Center of Supercomputing Applications) em 1993. O navegador teve papel importante no avanço da Web nos primeiros anos, principalmente por apresentar recursos de interface gráfica aprimorados, permitindo que criadores de documentos HTML melhorassem a apresentação das páginas, utilizando quadros, acrescentando cores e inserindo imagens nas páginas publicadas. Muitos dos novos recursos do navegador foram incorporados às especificações seguintes da linguagem HTML (CASTELLS, 2003).

Figura 3 - Navegador Mosaic.

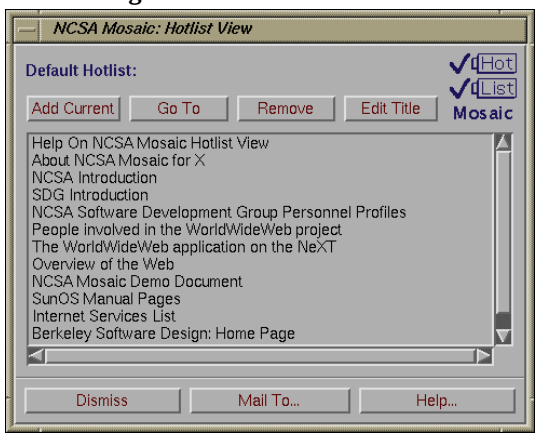


Fonte: Lasar (2011).

2.1.1 Bookmarking

Uma das funcionalidades introduzidas pelo Mosaic foi o “Hotlist” (figura 4), que oferecia um meio para que o usuário armazenasse links encontrados na Web. A inovação passa a fazer parte da maioria dos navegadores que surgiriam desde então.

Figura 4 - Hotlist do Mosaic.



Fonte: Desy (2014).

No Netscape, o recurso de armazenamento de links foi denominado “*Bookmark*”. No Internet Explorer, a funcionalidade foi nomeada como “Favoritos” (figura 5). Ambos os termos tornaram-se bastante utilizados e intercambiáveis, e quando os usuários desejavam salvar um link para acesso posterior, poderiam, desta forma, adicionar à sua listagem de favoritos ou *bookmarks* (FARKAS, 2007).

Figura 5 - Favoritos do Internet Explorer.

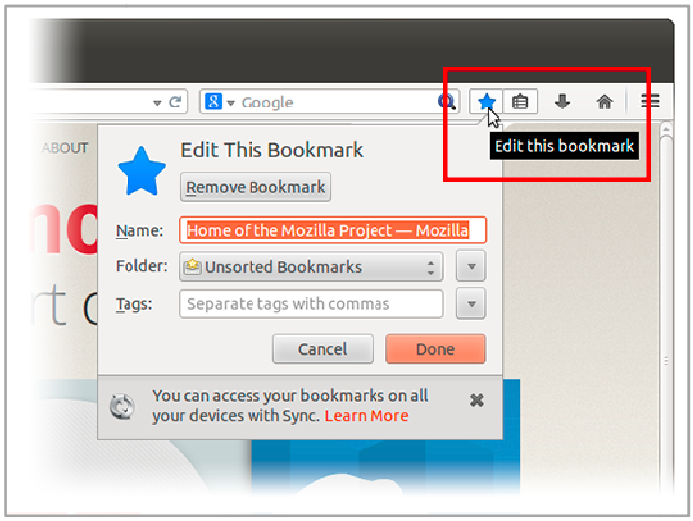
Fonte: Tymchuk (2014).

No novo contexto da Web, o termo em inglês *bookmark* faz referência ao uso original, como marcador de livro, ou marca-páginas, objeto utilizado para facilitar o acesso a uma determinada página de um livro.

O *bookmarking* representa o anseio de armazenar e organizar as novas descobertas ou informações relevantes que se encontram ocasionalmente pela Web, a fim de facilitar seu acesso posterior.

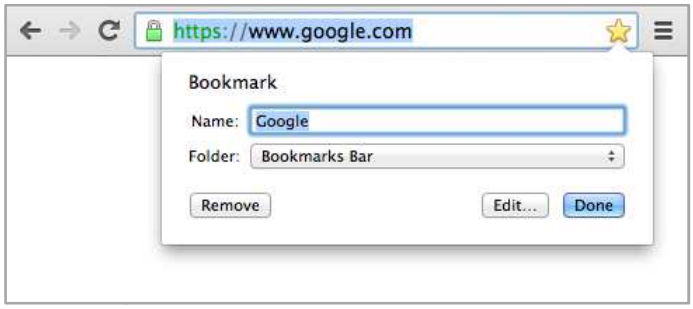
Os navegadores mais recentes, como o Firefox (figura 6) e o Chrome (figura 7), mantiveram a funcionalidade de armazenamento de links como um de seus principais recursos.

Figura 6 - Bookmarks do Firefox.



Fonte: Mozilla (2015).

Figura 7 - Bookmarks do Chrome.

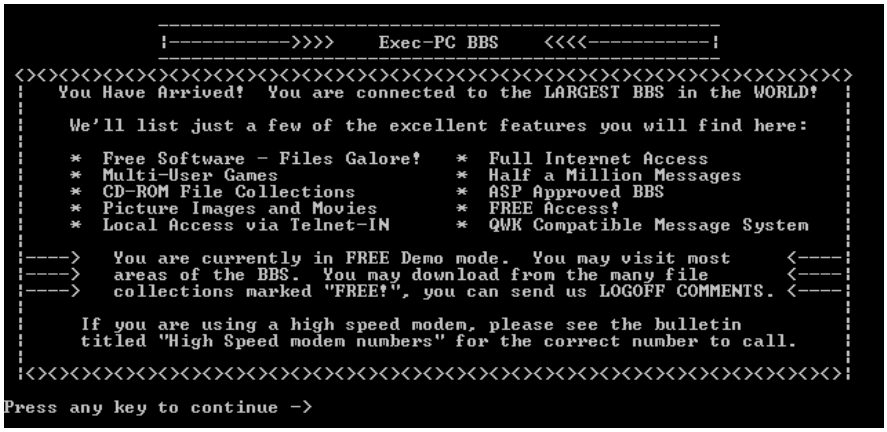


Fonte: Google (2015).

2.2 A WEB SOCIAL

Antes mesmo da criação da Web, a Internet já era utilizada como meio de interação social. Ao final da década de setenta, são criados os *Bulletin Board Systems* (BBS), ou sistema de quadro de avisos (figura 8). Estes permitiam a transferência de arquivos entre computadores pessoais, a partir de conexões realizadas com linhas telefônicas, com recursos de download, upload, leitura de notícias, participação em fóruns de discussão, chats e jogos on-line (FERREIRA, 2014).

Figura 8 - Tela de entrada de um BBS.



Fonte: Wikipedia (2014).

Durante a década seguinte, as conversas que ocorriam nos BBSs se expandiram para a Web. Os BBSs começaram a ter menos acesso, dando espaço para as salas de chat e fóruns. Ao final da década de 1990, o aumento expressivo de *websites* fortaleceu a ideia de que qualquer pessoa poderia criar e publicar páginas na Web. Empresas como a GeoCities e programas desktop como o FrontPage possibilitaram que pessoas com pouco conhecimento em desenvolvimento e linguagens de programação, construíssem *websites* sobre qualquer tipo de assunto. Neste contexto, surgem os blogs como diários pessoais on-line, e em 1999 aparecem os primeiros serviços que ofereciam uma plataforma para blogs, como o LiveJournal e o Blogger.

O aprimoramento da usabilidade das plataformas de blogs e a adoção do acesso doméstico a banda larga a custos acessíveis, contribuíram para a multiplicação dos blogs, antecipando um novo cenário, que passou a ser conhecido como a Web 2.0 (BAREFOOT e SZABO, 2010).

2.2.1 Web 2.0

O termo “Web 2.0” surgiu em 2004, quando representantes da editora O'Reilly e da promotora de eventos MediaLive realizaram uma sessão de “*brainstorming*” para conceber um congresso sobre Internet. Dale Dougherty, vice-presidente da O'Reilly, indica que a Web havia se tornado mais importante do que nunca, com novos aplicativos e sites sendo criados diariamente com uma regularidade e volume sem precedentes. Assim surge o termo “Web 2.0”, para referenciar a nova fase da Web, tema central da “Conferência Web 2.0”. Durante esta conferência, foram apresentadas as seguintes considerações centrais sobre o conceito de Web 2.0:

- A Web como plataforma;
- O usuário controla os dados;
- Serviços, não software “empacotado”;
- Arquitetura da participação;
- Eficiência visando a escalabilidade;
- Fonte e transformação de dados “remixáveis”;
- Software em mais de um dispositivo;
- Utilização da inteligência coletiva (O'REILLY, 2005).

Para O‘Reilly (2005), a Web 2.0 é uma atitude e não propriamente uma tecnologia. O principal recurso da plataforma é sua capacidade de atuar na circulação de dados oriundos das atividades dos usuários, em que a Web atua no todo como intermediário inteligente, ligando os extremos entre si e aproveitando as possibilidades que os próprios usuários oferecem, constituindo uma rede de colaboração entre os indivíduos, sustentada por uma Arquitetura da Participação, desenvolvida em torno das pessoas. Esta Arquitetura da Participação, sobre a qual a Web 2.0 é constituída, prevê novas ferramentas sociais para o intercâmbio do conhecimento, facilitadas a partir do desenvolvimento de novas tecnologias (ALVIM, 2011).

Neste contexto, a Web social emerge como um dos componentes mais relevantes da Web 2.0, fazendo da Internet um ambiente participativo, onde os indivíduos compartilham informações, sem a intervenção de uma autoridade central, com liberdade de utilizar, criar e reeditar o conteúdo. Conforme explicitado no diagrama da figura 9,

na Web 2.0 os usuários tornam-se também produtores de conteúdo (COUTINHO e JUNIOR, 2007):

Figura 9 - Comparativo entre Web 1.0 e Web 2.0.

Fonte: Ciência 2.0 (2014).

Os argumentos apresentados por Pierre Lévy sobre a Inteligência Coletiva, conforme observado de modo prático com a Web social, exprimem que as tecnologias na sociedade são mediadoras entre as inteligências individuais, e fortalecem as suas capacidades criativas. Os grupos de indivíduos que colaboram com conhecimento, também contribuem com a sociedade para que alcance um nível superior de inteligência, com o saber coletivo que transcende as inteligências individuais (ALVIM, 2011).

Observa-se então, que a Web desde o início teve caráter social, pois nela indivíduos interagem na troca de informações e na geração de conhecimento humano. A evolução das tecnologias possibilita que isso ocorra de maneira mais efetiva. As ideias relacionadas ao termo Web 2.0 representam a convergência de fatores, que se concretizam como ferramentas para a Web social. Tais ferramentas ficaram conhecidas como “mídias sociais”, ou “*social media*” em inglês.

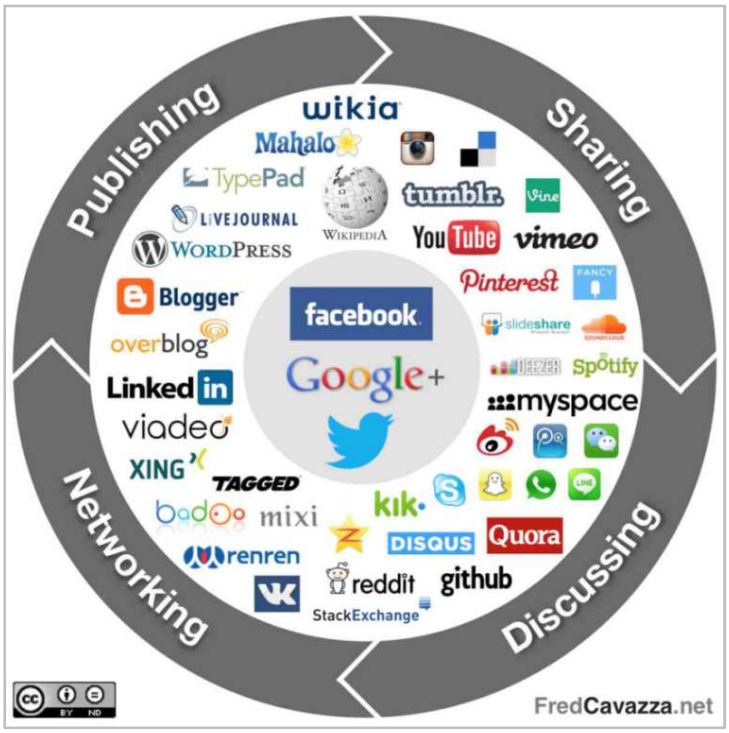
2.2.2 Mídias Sociais

A palavra mídia refere-se a qualquer instrumento ou meio de comunicação social, como o jornal impresso, revistas, TV, rádio. Esses meios de comunicação podem ser caracterizados como uma "via de mão única", na qual o indivíduo tem poucas possibilidades de participar, sendo designado como audiência, espectador ou leitor.

As mídias sociais, por outro lado, permitem a criação e o intercâmbio de conteúdos, muitos dos quais gerados pelos próprios usuários. Kaplan e Haenlein (2010) definem mídias sociais como "um grupo de aplicações baseadas na Web que representam as fundações tecnológicas e ideológicas da Web 2.0, permitindo a criação e compartilhamento de conteúdo gerado pelo usuário – *user generated content* (UGC)" (INTERNET INNOVATION, 2013).

A figura 10 é um panorama de mídias sociais popularmente conhecidas, distribuídas nas categorias: compartilhamento, discussão, relacionamento, publicação, ou ainda a combinação destas. Como exemplos de tipos de mídias sociais estão: redes de relacionamento, blogs, fóruns, microblogs, *wikis*, *bookmarks* sociais, sites de compartilhamento de mídia e *podcasts*:

Figura 10 - Social media landscape 2013.



Fonte: Cavazza (2014).

2.3 *BOOKMARKING* SOCIAL, *TAGGING* E FOLKSONOMIA

Os serviços de *bookmarking* social são um tipo de mídia social que permitem ao usuário guardar e organizar seus *bookmarks* coletivamente em uma plataforma social on-line.

O *bookmarking* social refere-se à atividade de armazenar links da Web em ambiente virtual, de modo que as informações inseridas no sistema possam ser compartilhadas entre os usuários. Cada usuário pode, além de manter uma coleção de *bookmarks*, visitar as coleções dos demais usuários. Outra vantagem deste tipo de serviço, é que os links ficam guardados na "nuvem", permitindo o acesso a partir de qualquer dispositivo ou navegador.

Nestes sistemas, a organização dos links é frequentemente realizada com o uso de *tags* (etiquetas), que são palavras-chave utilizadas com o intuito de facilitar a identificação do conteúdo do link. O processo de atribuição de *tags* resulta na "etiquetagem" (*tagging*), uma maneira não-hierárquica de classificação de informações. Na prática, cada *tag* apresenta-se como um hiperlink que conduz a uma página que engloba o conjunto dos recursos que recebem a respectiva *tag*. Sendo assim, ao se clicar em uma *tag* pode-se ter acesso a outros recursos aos quais foi atribuída esta mesma *tag* (LOMAS, 2005).

Entre *websites* populares que oferecem o serviço de *bookmarking* social pode-se citar: Delicious, Digg, Reddit, Pinterest, Pinboard e Diigo. Outros tipos de plataformas sociais, como as redes de relacionamento Facebook, Twitter e Google+, oferecem meios de compartilhar conteúdo entre os usuários que, na prática, acabam tendo função semelhante aos serviços de *bookmarking* social.

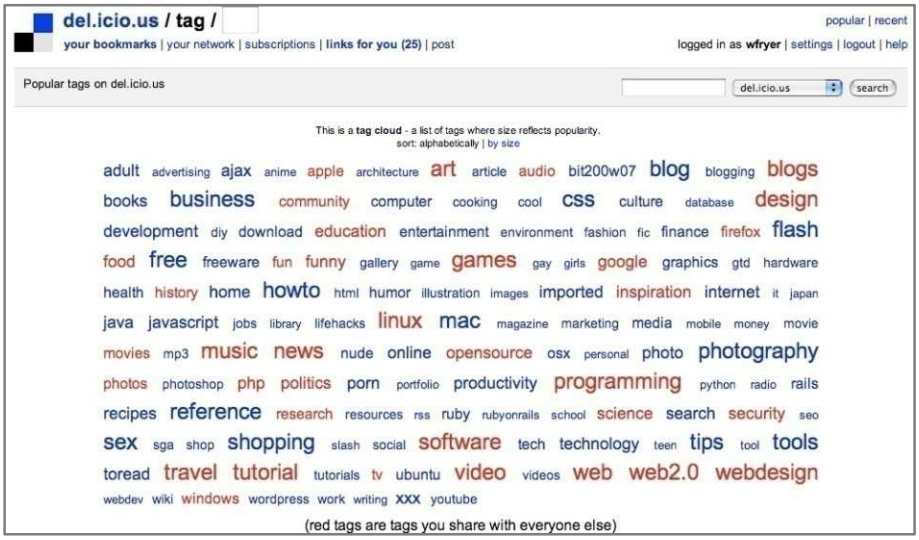
Entre características comuns observadas nestes serviços, destacam-se:

- Armazenamento de links on-line;
- Possibilidade de acesso a coleção de links de outros usuários;
- Filtragem de links por *tag*, popularidade, ou mais recentes;
- Atribuição livre de *tags* aos links pelos usuários para facilitar a identificação do conteúdo;
- Indicação de relevância de um link a partir da sua popularidade entre os usuários;

- Indicação de relevância de uma *tag* a partir da frequência em que é utilizada;
- Possibilidade de estabelecer uma rede de relacionamentos entre os usuários.

A classificação com *tags* oferece a possibilidade de novas formas de organização e categorização de recursos. A liberdade de cada usuário ao atribuir etiquetas virtuais a um recurso, resulta em um método de classificação das informações "amador", conduzido pela atividade dos usuários, sem formalidades explícitas. Nas plataformas de *bookmarking* social, a comunidade de usuários, ao longo do tempo, desenvolve uma base única de *tags* e recursos, resultando em uma estrutura conhecida como "folksonomia". Na figura 11, pode-se visualizar uma “nuvem de *tags*”, como representação da folksonomia do site Delicious:

Figura 11 - Nuvem de tags do site Delicious.



Fonte: Roettgers (2013).

O termo folksonomia, introduzido por Thomas Vander Wal em 2004, a partir da observação de uma prática crescente, refere-se à junção das palavras *folk* (do inglês, pessoas, povo) e taxonomia, e pode ser compreendido como “classificação popular”. Em contraste com a taxonomia, a folksonomia não possui hierarquias ou outro tipo de convenção técnica (BRANDT, 2009).

A folksonomia é observada como resultado da etiquetagem dos recursos da Web num ambiente social, pelos próprios usuários da informação visando à recuperação de recursos. Destacam-se, portanto três fatores essenciais:

- É resultado de uma indexação livre do próprio usuário do recurso;
- Objetiva a recuperação *a posteriori* da informação;
- É desenvolvida num ambiente aberto que possibilita o compartilhamento, e até em alguns casos, a sua construção conjunta (BAPTISTA e CATARINO, 2007).

O *bookmarking* social e a folksonomia dão aos usuários a oportunidade de compartilhar recursos e perspectivas sobre as informações, o que contribui para que indivíduos com mesmo interesse criem comunidades em torno de assuntos em comum. As preferências de outros usuários conectam informações relacionadas, mesmo em temas que não estão obviamente ligados. Se o usuário adiciona a um determinado recurso à *tag* "velejar", por exemplo, outros podem adicionar *tags* como "veleiros", "barcos" ou "mar". O recurso acumula cada uma das *tags* e favorece a pesquisa, podendo levá-la a novas direções, potencialmente valiosas (LOMAS, 2005).

A maioria dos autores que tratam sobre folksonomia identificam suas vantagens e desvantagens para a organização e recuperação da informação. A principal desvantagem citada é a falta de controle de vocabulário, ou seja, problemas como termos homônimos, sinônimos e uso de singular/plural, gerando inconsistência, redundância e falsas associações, o que prejudica a precisão da informação recuperada. Afirma-se também que a folksonomia é um estrutura sem controle: a mesma etiqueta pode ser usada para recursos diferentes, com sentidos semânticos diferentes, criando associações não relacionadas entre os recursos. Como vantagens os autores apontam o baixo custo de implementação, o vocabulário próprio do usuário e não o do sistema, a inserção em tempo real de temas emergentes, o cunho social e colaborativo, a formação de comunidades em torno de assuntos de interesse, a liberdade de expressão do usuário, a flexibilidade do sistema, a descoberta de informação nova, e, no caso de recursos visuais, a folksonomia possibilita a descrição do conteúdo visual e as etiquetas funcionam como uma espécie de metadado para a recuperação da informação (BRANDT, 2009).

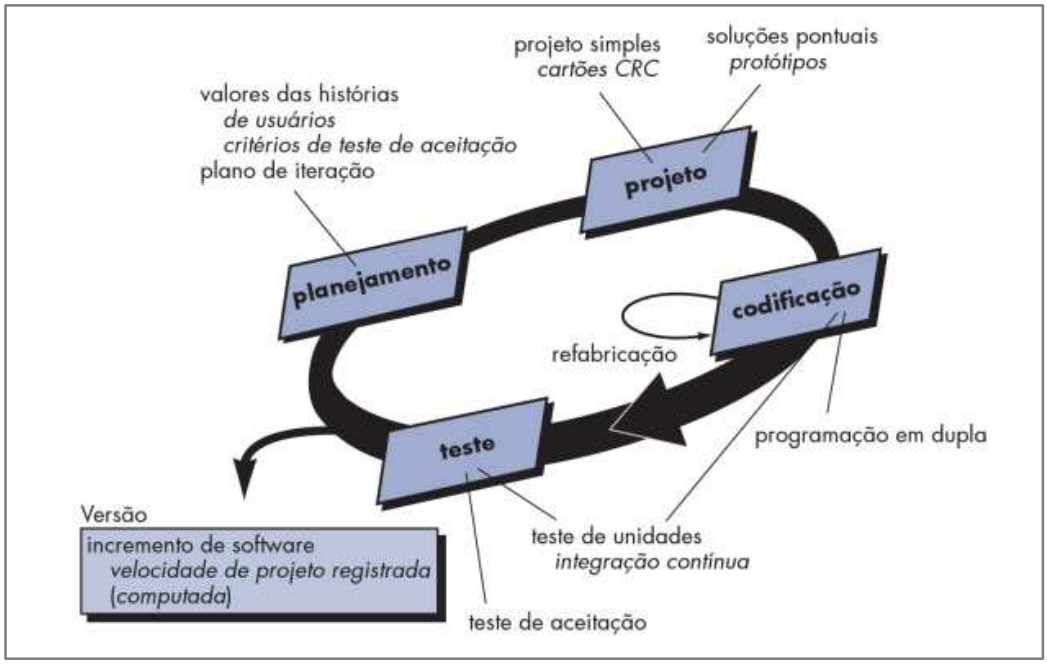
2.4 TEST-DRIVEN DEVELOPMENT

Test-driven Development (TDD), ou em português desenvolvimento guiado por testes, é uma pratica que foi proposta por Kent Beck como parte essencial da *eXtreme Programming* (XP). O TDD reflete os princípios da XP nos níveis mais internos e concretos das atividades de desenvolvimento, constituindo um dos pilares da metodologia (MILANI, PRIKLADNICKI e WILLI, 2014).

A XP, ou Programação Extrema, é uma das principais metologias para desenvolvimento ágil de software, destacando-se por definir em nível técnico um conjunto de práticas, dentre as quais o TDD e as histórias de usuários. Embora definidas originalmente com a XP, tanto o TDD como as histórias de usuários, são frequentemente utilizados em outras metodologias de desenvolvimento de software.

Na figura 12 é apresentado o ciclo de desenvolvimento da Programação Extrema:

Figura 12 - Visão geral do ciclo de desenvolvimento da XP



Fonte: Pressman (2011, p. 88).

O TDD é uma técnica de desenvolvimento de software que utiliza testes automatizados para assegurar que o código funcione (e continue funcionando) conforme se deseja. O TDD, também é refenciado como *"Test-driven Design"*, pois, além de

conduzir o desenvolvimento do código, a técnica resulta na construção satisfatória do design ou arquitetura do sistema (PERCIVAL, 2014).

Martin Fowler destaca o TDD como uma prática central na metodologia, sendo um dos fatores que a torna viável. Isso porque ajuda a manter controlado o crescimento do custo das mudanças ao longo do projeto – talvez a premissa maior por trás de XP. Em outras palavras, a prática do TDD (dentre outras) mantém a solução mais facilmente modificável durante o desenvolvimento, permitindo revisões constantes e viabilizando a estratégia adaptativa como um todo (MILANI, PRIKLADNICKI e WILLI, 2014).

2.4.1 Ciclo *Red-Green-Refactor*

No TDD cada teste deve ser escrito antes do código, permitindo um ciclo breve conhecido como *Red-Green-Refactor* (Vermelho-Verde-Refatore), que se repete indefinidamente e conduz todo o desenvolvimento do software (MARTIN, 2014).

O ciclo *Red-Green-Refactor*, que pode ser verificado na figura 13, corresponde às seguintes fases:

1. *Red*: Escreva um teste para um novo código e o veja falhar;
2. *Green*: Escreva o novo código fazendo somente o suficiente para o teste passar;
3. *Refactor*: Veja o teste passar e melhore a qualidade do código.

Figura 13 - Ciclo "Red-Green-Refactor"

Fonte: Adaptado de Martin (2014).

2.4.2 Testes Unitários

Um teste unitário verifica se uma parte do programa reproduz o resultado desejado. Cada teste unitário simula dados de entrada visando um comportamento definido de uma única parte de um programa. Geralmente testam o retorno dos métodos ou funções de modo isolado, a partir de uma situação exemplo. Em TDD, cada teste unitário deve ser escrito antes do programa, conforme especificado pelo ciclo *Red-Green-Refactor* (HAMILL, 2009).

Testes atualizados irão corresponder ao software desenvolvido, que além de monitorar constantemente a qualidade, servem também como uma espécie de especificação do sistema, visto que detalham como o software deve funcionar (PERCIVAL, 2014).

2.4.3 Testes Integrados

Um teste integrado verifica se componentes do sistema interagem corretamente quando combinados. Diferentemente dos testes unitários, não são isolados, ou seja, não testam apenas um comportamento esperado. O teste integrado caracteriza-se quando um método ou função depende de algum sistema externo que pode causar uma falha no teste. O sistema externo pode ser, por exemplo, um banco de dados ou uma função a qual não se tem controle. Desta forma o teste integrado testa necessariamente dois ou mais comportamentos, sendo que se tem controle parcial sobre o sistema em desenvolvimento (HAMILL, 2009).

2.4.4 Testes Funcionais

Testes funcionais também podem ser denominados testes de aceitação. Estes verificam se o software comporta-se adequadamente sob o ponto de vista do usuário. Geralmente, não especificam como os resultados são obtidos; ao invés disso, devem

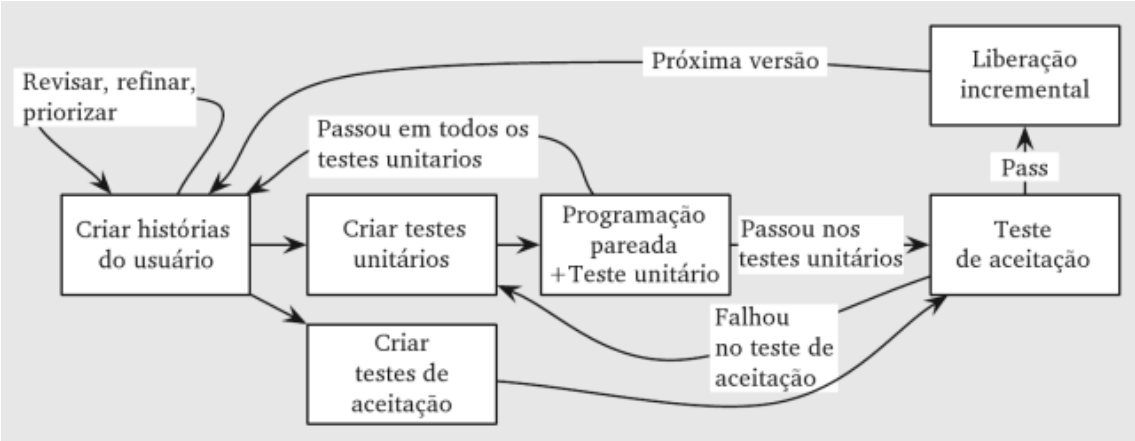
representar quais os resultados esperados. Não são necessariamente escritos por programadores, e devem ser acessíveis ao usuário. O programador, porém, pode tornar estes testes automatizados (HAMILL, 2009).

Os testes funcionais permitem que o programador verifique se está construindo a funcionalidade conforme as expectativas do usuário. Desta maneira, o programador trabalha com maior objetividade, construindo algo que irá trazer valor ao usuário.

Normalmente são definidos utilizando-se as “Histórias de Usuário”, prática muito utilizada no desenvolvimento ágil de software, como substituição às extensas especificações do desenvolvimento tradicional. As “Histórias de Usuário” devem ser “mapeadas” para testes funcionais, sendo que uma História de Usuário pode dar origem a um conjunto de testes funcionais que validam se a “história” pode ser realizada pelo usuário do sistema (COHN, 2009).

Se o teste funcional passa, significa que os testes unitários também passaram. Consequentemente a iteração pode ser dada como concluída. O ciclo do TDD pode combinar testes funcionais com testes unitários, conforme a figura 14, para conduzir o desenvolvimento do software que atenda as expectativas do usuário, mantendo em foco a qualidade técnica do código.

Figura 14 - Ciclo TDD combinado com teste funcional.



Fonte: Pezzé e Young (2008, p. 404).

2.4.5 Histórias de Usuário

O conceito de “Histórias de Usuário” está relacionado como uma alternativa ágil para o que é considerado requisito de software no desenvolvimento de software tradicional. É uma prática essencial para o desenvolvimento ágil, definida inicialmente pela XP, e é adotada de modo geral pelas metodologias ágeis (COHN, 2009).

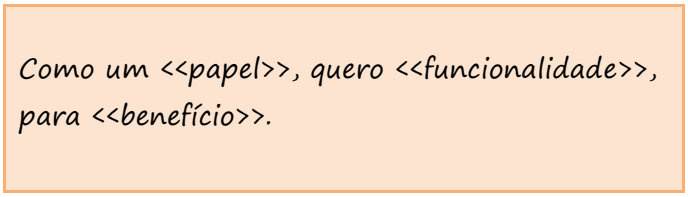
As Histórias de Usuário são uma técnica simplificada utilizada para capturar e comunicar as funcionalidades desejadas, descrevendo aquilo que deverá ter valor ao usuário do sistema. Uma História de Usuário é uma breve declaração de intenções que descreve algo que o sistema precisa fazer para o usuário. São compostas por três aspectos:

- Descrição por escrito da história utilizada como lembrete e planejamento;
- Conversações sobre a história, a fim de aprofundar os detalhes;
- Especificações que transmitem claramente quando uma “história” foi realizada.

São concebidas juntamente com o usuário (ou alguém que o represente), e servem para se comunicar de modo breve e claro cada funcionalidade a todos os envolvidos no desenvolvimento do software.

O modelo de história de usuário (Figura 15) tornou-se popular no desenvolvimento ágil de software, sendo considerado uma forma eficiente de identificar funcionalidades desejadas, bem como a relação de importância entre elas. Primeiro determina-se quem é o interessado na funcionalidade por meio de um “papel” desempenhado por um tipo de usuário, e em seguida a “funcionalidade” desejada, ou ação deste usuário no sistema. Opcionalmente pode-se destacar o “benefício”, ou o que a funcionalidade deverá trazer de valor para o usuário (LEFFINQWELL, 2010).

Figura 15 - Modelo de cartão de história de usuário.



Fonte: Adaptado de Leffinqwell (2010, p. 103).

2.5 LINGUAGEM PYTHON

Python é uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, de tipagem dinâmica e forte, lançada por Guido Van Rossum em 1991. A partir da versão 2.0, lançada em outubro de 2000, a linguagem passou a ter um modelo de desenvolvimento comunitário aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation (PSF) (PYTHON, 2015).

O Python foi projetado para otimizar a produtividade do desenvolvedor, a qualidade do software, a portabilidade do programa e a integração dos componentes. Os programas podem ser executados na maioria das plataformas de uso comum, incluindo *mainframes* e supercomputadores, Unix e Linux, Windows e Macintosh, Java e .NET, ou em qualquer outra que possua um interpretador Python disponível,.

Entre as características da linguagem Python pode-se destacar:

- É interpretada, o que significa que as instruções precisam ser reconhecidas pelo interpretador Python que executa o programa por meio de uma máquina virtual;
- Suporte à programação procedimental e orientada a objetos;
- Tipagem dinâmica, dispensando a necessidade da declaração de variáveis, detectando o seu tipo automaticamente;
- Tipos de alto nível: listas, tuplas, dicionários e conjuntos;
- Controle de blocos por endentação, o que favorece a legibilidade do código (LUTZ, 2010).

Na figura 16, um exemplo de código Python:

Figura 16 - Exemplo de código Python.

```
class Rectangle(Blob):

    def __init__(self, width, height,
                 color='black', emphasis=None, highlight=0):
        if width == 0 and height == 0 and \
           color == 'red' and emphasis == 'strong' or \
           highlight > 100:
            raise ValueError("sorry, you lose")
        if width == 0 and height == 0 and (color == 'red' or
                                           emphasis is None):
            raise ValueError("I don't think so")
        Blob.__init__(self, width, height,
                      color, emphasis, highlight)
```

Fonte: Pimentel (2009).

2.6 *FRAMEWORK* DJANGO

Um *framework* é uma estrutura de software pré-construída sobre a qual se desenvolve aplicações com um aspecto comum, conservando-se a adaptabilidade para diferentes situações. Normalmente conduz a uma prática padronizada no processo de desenvolvimento. Sendo assim, um *framework* Web é uma estrutura que objetiva acelerar e melhorar o desenvolvimento de aplicações Web, de maneira padronizada e adaptável, evitando-se o retrabalho.

Django é um *framework* Web escrito com a linguagem Python que foi criado em 2003, com os programadores Web do jornal Lawrence Journal-World, Adrian Holovaty e Simon Willison (DJANGO, 2014).

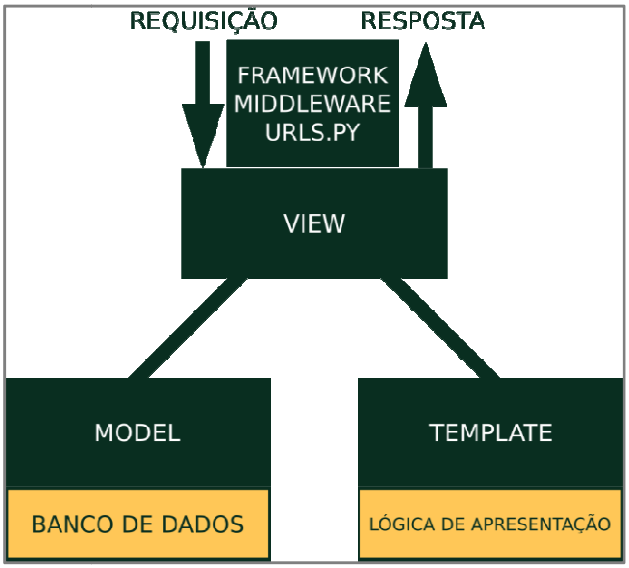
Sob o acelerado ritmo de um ambiente da redação de um jornal on-line, o *framework* Django foi concebido para tornar as tarefas comuns do desenvolvimento Web, mais rápidas e fáceis de serem realizadas. Foi lançado publicamente sob a licença livre Berkeley Software Distribution (BSD), em 2005 e nomeado em homenagem ao guitarrista Django Reinhardt (HOLOVATY e MOSS, 2009).

Em junho de 2008, foi estabelecida a Django Software Foundation (DSF), uma organização independente e sem fins lucrativos, que desde então, promove o desenvolvimento de código aberto do *framework* com a colaboração da comunidade de desenvolvedores voluntários.

2.6.1 Arquitetura do Django

A arquitetura do Django corresponde a uma variação do padrão *Model-View-Controller* (MVC), utilizado por outros *frameworks* Web desenvolvidos em diversas linguagens, como Ruby on Rails (Ruby), Spring MVC (Java), ASP.NET MVC (C#) e Zend Framework (PHP). O Django pode ser frequentemente considerado um *framework* MVC, porém algumas diferenças conceituais fazem com que a nomenclatura mais adequada, no caso do Django, seja “MTV” (*Model-Template-View*) conforme a figura 17:

Figura 17 - Arquitetura MTV do Django.

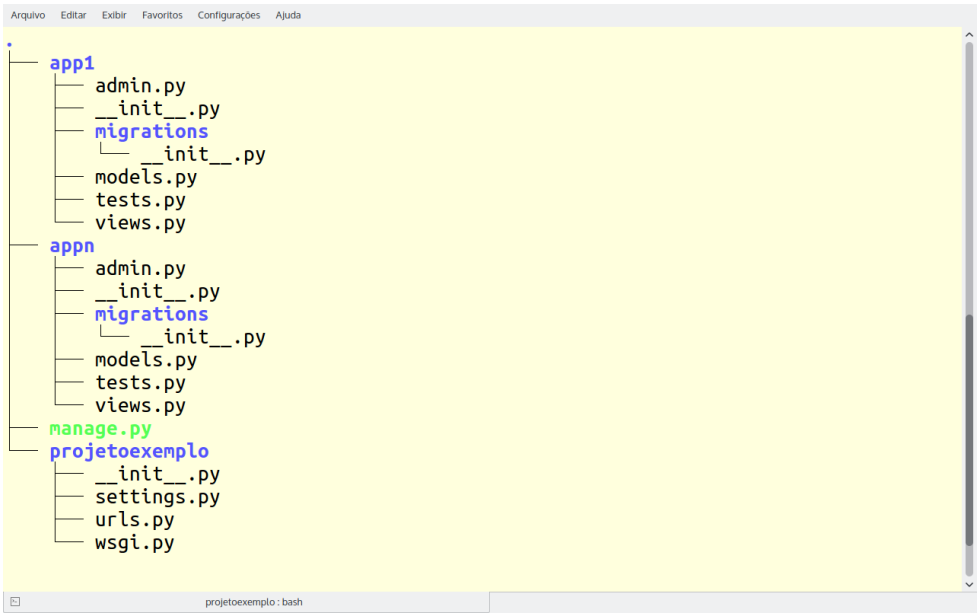


Fonte: Adaptado de Pirnat (2014).

Na arquitetura MTV do Django, a camada *model* fornece uma abstração dos modelos para estruturação e manipulação de dados do sistema em um banco de dados, por meio de um sistema ORM (*Object Relational Mapping*). A camada *template* oferece uma sintaxe para apresentar os dados ao usuário de maneira adequada. E a camada *view* encapsula a lógica responsável pelo processamento e resposta de uma solicitação do usuário (PIRNAT, 2014).

Na figura 18 pode-se observar um exemplo de estrutura de pastas de um projeto Django:

Figura 18 - Estrutura de diretórios de um projeto Django.



Fonte: Autoria própria.

2.6.2 Recursos do Django

O Django segue a filosofia “baterias incluídas”, que visa prover um conjunto de funcionalidades integradas, conforme apresentado a seguir:

- Mapeador-Objeto Relacional (ORM) com suporte a múltiplos bancos de dados;
- Aplicações “plugáveis”;
- Roteador de URLs;
- Linguagem de *template*;
- Utilitários de linha de comando;
- Gerador automático de interface de administração;
- Servidor de desenvolvimento;
- Suporte a migrações de banco de dados;
- *Application Programming Interface* (API) para gestão de Formulários;
- Módulo de configurações;
- Segurança;
- Autenticação, autorização e sessões;
- Internacionalização e localização;
- Medição e otimização de desempenho;
- Sistema de *caching*;
- Suporte ao desenvolvimento de aplicações geográficas utilizando o *Geographic Information System* (GIS);
- *Logging*;
- Envio de e-mails;
- *Feed Syndication* (*Rich Site Sumary* - RSS/Atom);
- Paginação;
- Gerenciamento de arquivos estáticos;
- Validação de dados;
- Módulo de testes;
- Integração com o “ecossistema” da linguagem Python, entre outros recursos (HOLOVATY e MOSS, 2009).

Django promove o desenvolvimento rápido de aplicações Web com a reutilização e "plugabilidade" de componentes, baseando-se no princípio *Don't Repeat Yourself* (DRY). Sugerido por Hunt e Thomas (2010), o princípio declara que cada parte do sistema deve ter uma representação única, não ambígua e definitiva, a fim de se obter um software flexível e fácil de manter (HUNT e THOMAS, 2010).

Os recursos oferecidos pelo framework associados ao desenvolvimento DRY visam à melhoria de qualidade do software, bem como maior produtividade do desenvolvedor.

2.6.2.1 Aplicações reutilizáveis do Django

Quando se desenvolve com Django, cria-se o que o *framework* chama de "projeto" que é composto por diversas "*apps*" ou "aplicações". As *apps* do Django são pacotes Python que englobam um conjunto de recursos com um fim específico, seguindo as prescrições da filosofia DRY. O *framework* oferece meios para que as aplicações sejam "plugáveis" e possam ser reutilizadas em diversos projetos diferentes. Isto evita o retrabalho e contribui para a rapidez no processo de desenvolvimento com o *framework*. No entanto, é necessário haver disciplina por parte do desenvolvedor, seguindo as especificações do *framework*, para que as aplicações sejam, de fato, reutilizáveis.

Outro efeito positivo desta característica, é que sutilmente incentiva os desenvolvedores a compartilhar as suas aplicações plugáveis como *opensource*, com a comunidade de desenvolvedores, para que outros também possam se beneficiar do software desenvolvido, assim como o próprio Django.

2.6.2.2 HTML e Linguagem de template do Django

HTML é uma linguagem de marcação que descreve o conteúdo de documentos da Web. Ela é composta por elementos que estruturam e identificam o conteúdo envolvido por *tags* HTML. Dessa maneira os navegadores podem interpretar

diferentemente as porções do documento, como sendo, por exemplo, títulos, parágrafos, imagens ou *links*.

A linguagem de *template* do Django foi criada para complemetar a linguagem HTML. Ela é compatível com a HTML e adiciona *tags* e *filters* específicos da linguagem de *template* do Django, oferecendo recursos simples que tornam possível a inserção de lógica e dinâmica na apresentação dos dados. Também permite a utilização de variáveis de contexto no *template* para que o *webserver* "sirva" a página HTML renderizada de modo adequado (HOLOVATY e MOSS, 2009).

2.6.2.3 ORM do Django e Queryset API

O ORM do Django permite definir a modelagem de dados através de classes em Python. Com isso, é possível gerar tabelas no banco de dados e acessá-las sem necessidade da utilização da linguagem SQL (*Structured Query Language*) (RAMALHO, 2009).

O ORM do Django oferece:

- Suporte a diversos Sistemas Gerenciadores de Bancos de Dados;
- Criação de tabelas, campos e relacionamentos de acordo com dados especificados por meio de classes;
- Implementação conveniente e flexível de operações CRUD;
- Validação de campos;
- Transações ACID.
- Métodos de consulta por meio da QuerySet API

Os métodos de QuerySet são utilizados para adicionar regras de consulta ao banco de dados. Podem oferecer acesso a outras QuerySets, o que possibilita o encadeamento de consultas a fim de combinar regras para a obtenção de dados (BRANDÃO, 2009).

2.6.2.4 Módulo de testes do Django

Os testes de Django utilizam o modulo da biblioteca padrão do Python chamado unittest.

O unittest é um *framework* de testes unitários que é caracterizado como um *framework* xUnit, nome dado para representar um tipo de *framework* de teste que seguem um determinado padrão, porém em implementações para diversas linguagens de programação. O nome xUnit é uma derivação de JUnit (*framework* de testes do Java). O unittest por sua vez foi originalmente desenvolvido a partir da implementação do JUnit.

Ele suporta a automação de teste, compartilhamento de configuração e desativação de código para testes, a agregação de testes em coleções e a independência dos testes do *framework* de relatórios.

Para realizar isto, o unittest oferece suporte aos seguintes conceitos:

- *Test fixture*: um *test fixture* representa a preparação necessária para realizar um ou mais testes, e quaisquer ações de limpeza. Isto pode envolver, por exemplo, a criação temporária ou a representação de bancos de dados, diretórios, ou a inicialização de um processo de servidor;
- *Test case*: um *test case* é uma unidade de teste. Ele checa por uma resposta específica a partir de um determinado conjunto de estímulos. O Unittest oferece uma classe chamada TestCase que deve ser utilizada para criar novos casos de testes;
- *Test suite*: é uma coleção de *test cases*, ou ainda de outros *test suites*. Serve para agregar testes que devem ser executados juntos;
- *Test runner*: o *test runner* é um componente que organiza a execução dos testes e produz relatórios ao usuário. O *runner* deve utilizar uma interface gráfica, uma interface textual ou retornar um valor especial para indicar os resultados da execução dos testes (THE PYTHON SOFTWARE FOUNDATION, 2015).

A figura 19 é um exemplo de *test case* em Python:

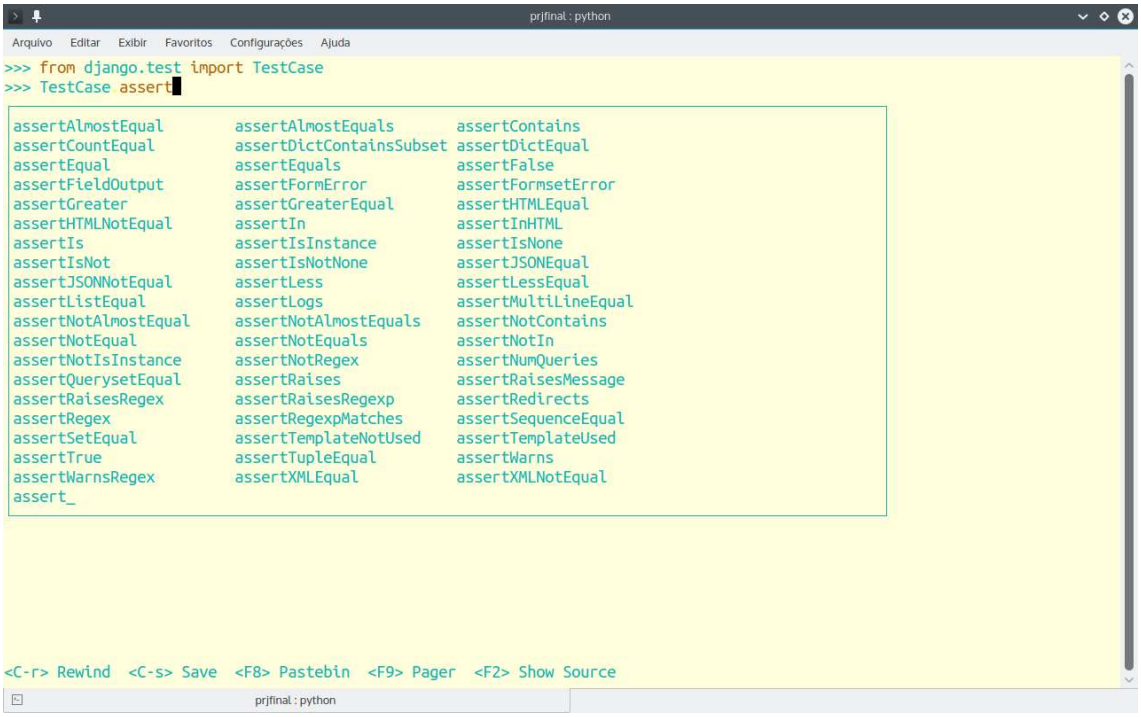
Figura 19 - Exemplo de um test case.

```
import unittest

class TestStringMethods(unittest.TestCase):

  def test_upper(self):
      self.assertEqual('foo'.upper(), 'FOO')

  def test_isupper(self):
      self.assertTrue('FOO'.isupper())
      self.assertFalse('Foo'.isupper())

  def test_split(self):
      s = 'hello world'
      self.assertEqual(s.split(), ['hello', 'world'])
      # check that s.split fails when the separator is not a string
      with self.assertRaises(TypeError):
          s.split(2)

if __name__ == '__main__':
    unittest.main()
```

Fonte: Python (2015).

A figura 20 relaciona os métodos das assertivas disponibilizadas pelo *framework* de testes do Django:

Figura 20 - Assertivas disponibilizadas pelo módulo de teste do Django.



Fonte: Autoria própria.

3 MATERIAL E MÉTODOS

Este capítulo apresentará as ferramentas e tecnologias utilizadas no desenvolvimento da aplicação

3.1 FERRAMENTAS UTILIZADAS

Para o desenvolvimento da plataforma foram utilizadas as seguintes ferramentas e tecnologias:

- Linguagem Python 3;
- *Framework* Django 1.8;
- Sistema operacional Ubuntu;
- Terminal;
- Editor Sublime Text;
- *Plug-ins* para Sublime Text: Anaconda, Djaneiro, CSS Extended Completions, Emmet;
- Gerenciador de pacotes Python Pip;
- Gerenciador de ambientes Python Virtualenv;
- Interpretador BPython;
- Sistema Gerenciador de Banco de Dados SQLite;
- Gerenciador de pacotes Bower;
- *Framework* de interface de usuário Semantic UI.

O sistema é desenvolvido com o *framework* Django versão 1.8.2 e a linguagem Python 3.4 instalados no sistema operacional Ubuntu versão 15.04.

Diversas tarefas foram realizadas a partir do terminal, especialmente os comandos disponibilizados pelo Django. O editor de textos utilizado foi o Sublime Text versão 3, com os *plugins* Anaconda, que oferece recursos que auxiliam no desenvolvimento com a linguagem Python, o Djaneiro, utilizado principalmente na

codificação de *templates*, e o Css Extended Completions e Emmet, com o objetivo de acelerar o desenvolvimento com as linguagens HTML e CSS.

Na criação de ambientes isolados Python foi utilizado o Virtualenv, e para a instalação de pacote nestes ambientes, o Pip.

O interpretador BPython foi instalado no ambiente de desenvolvimento por oferecer recursos adicionais ao interpretador padrão.

Uma das configurações iniciais do Django define o SQLite como banco de dados. A configuração foi mantida.

Os pacotes necessários para o desenvolvimento com HTML e CSS foram atendidos pelo gerenciador de pacotes Bower, como o caso do Semantic UI.

3.2 PROCESSO PARA DESENVOLVIMENTO DA APLICAÇÃO

Para a criação da plataforma foi seguido o seguinte processo de desenvolvimento:

1. Definição de requisitos utilizando histórias de usuários;
2. Prototipagem de UI com Semantic UI;
3. Preparação de ambiente Python;
4. Criação de projeto Django;
5. Criação de *app* Django;
6. Início do ciclo de desenvolvimento guiado por testes;
7. Inserção de dados no *admin* do Django;
8. Verificação das especificações definidas nas histórias de usuários no software final.

3.3 HISTÓRIAS DE USUÁRIO

No quadro 1 estão as Histórias de Usuário utilizadas para a determinação das funcionalidades desejadas no sistema:

Quadro 1 - Histórias de usuário para o desenvolvimento do sistema.

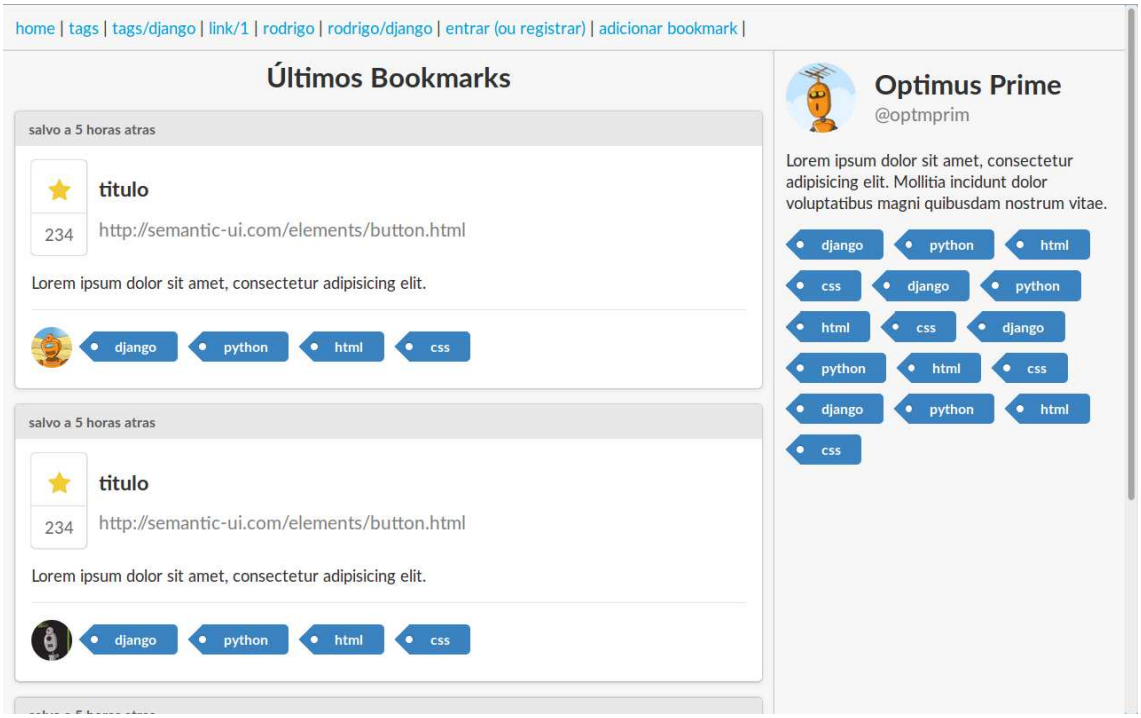
Como um usuário, quero visualizar os últimos bookmarks da comunidade, para descobrir novos links, usuários e tags.
Como um usuário, quero visualizar todas as tags da comunidade, para descobrir novos links.
Como um usuário, quero visualizar os bookmarks que salvei e tags que utilizei.
Como um usuário, quero guardar meus links favoritos, para acessá-los posteriormente.
Como um usuário, adicionar tags aos meus bookmarks, para facilitar a identificação e pesquisa.
Como um usuário, quero ter um perfil de usuário com foto, para ser reconhecido.
Como um usuário, quero visualizar os bookmarks e tags de outros usuários.
Como um usuário, quero filtrar os bookmarks por tags, para encontrá-los com mais facilidade.

Fonte: Autoria própria.

3.4 PROTIPAGEM DE INTERFACE DE USUÁRIO

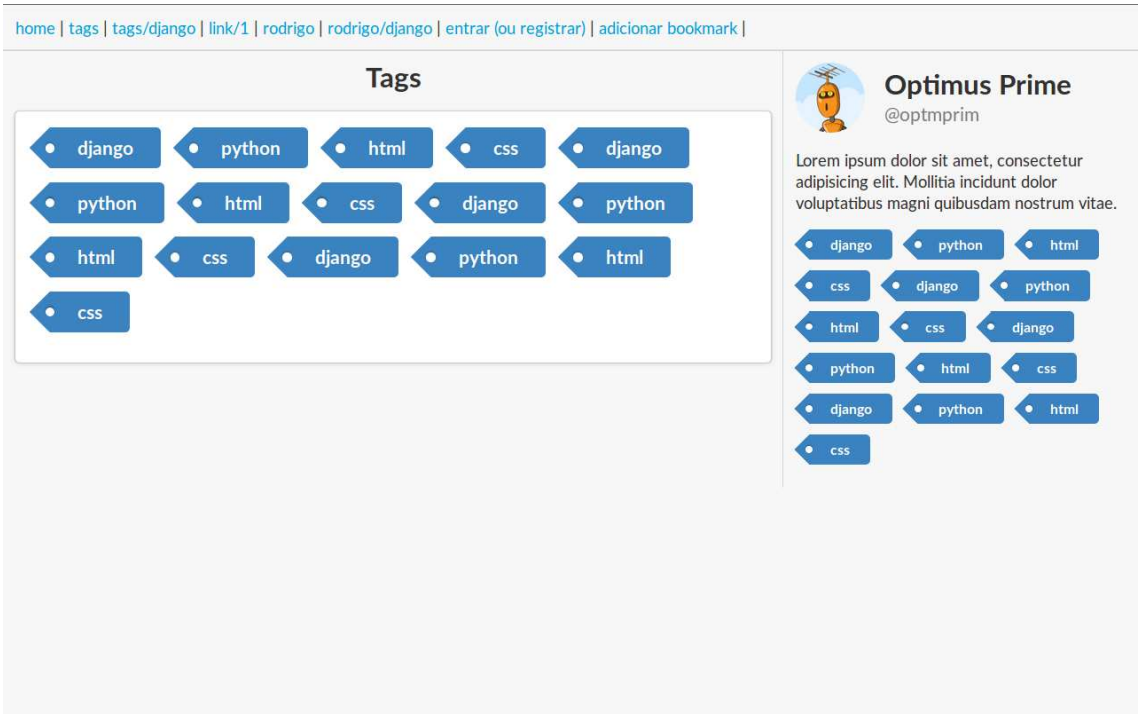
O protótipo do sistema foi construído com páginas HTML utilizando o framework Semantic UI, que dispõe de uma coleção de componentes de interface de usuário. O protótipo teve como propósito realizar uma pré-avaliação de como as histórias de usuários poderiam ser aplicadas a partir da suposta interação do usuário com a interface visual do sistema. Após diversos testes de layout, o resultado obtido e julgado como satisfatório está exposto nas figuras de 21 a 29. O protótipo serviu como base para o posterior desenvolvimento dos *templates* do sistema:

Figura 21 - Protótipo da página "home".



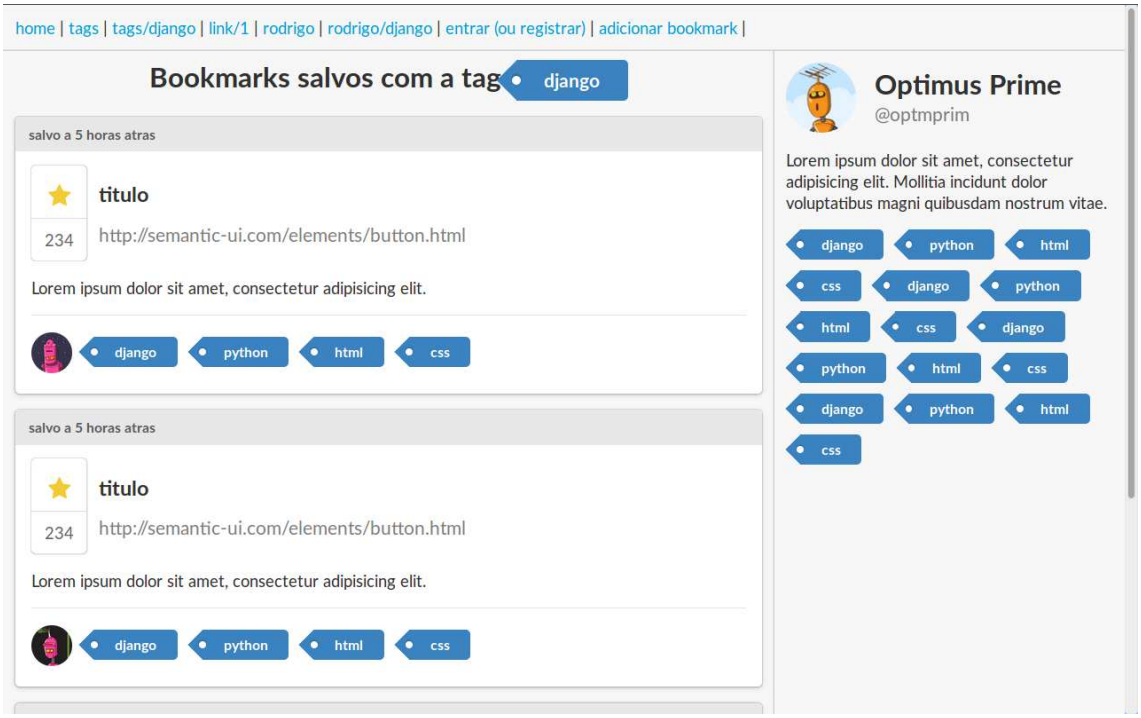
Fonte: Autoria própria.

Figura 22 - Protótipo da página “tags”.



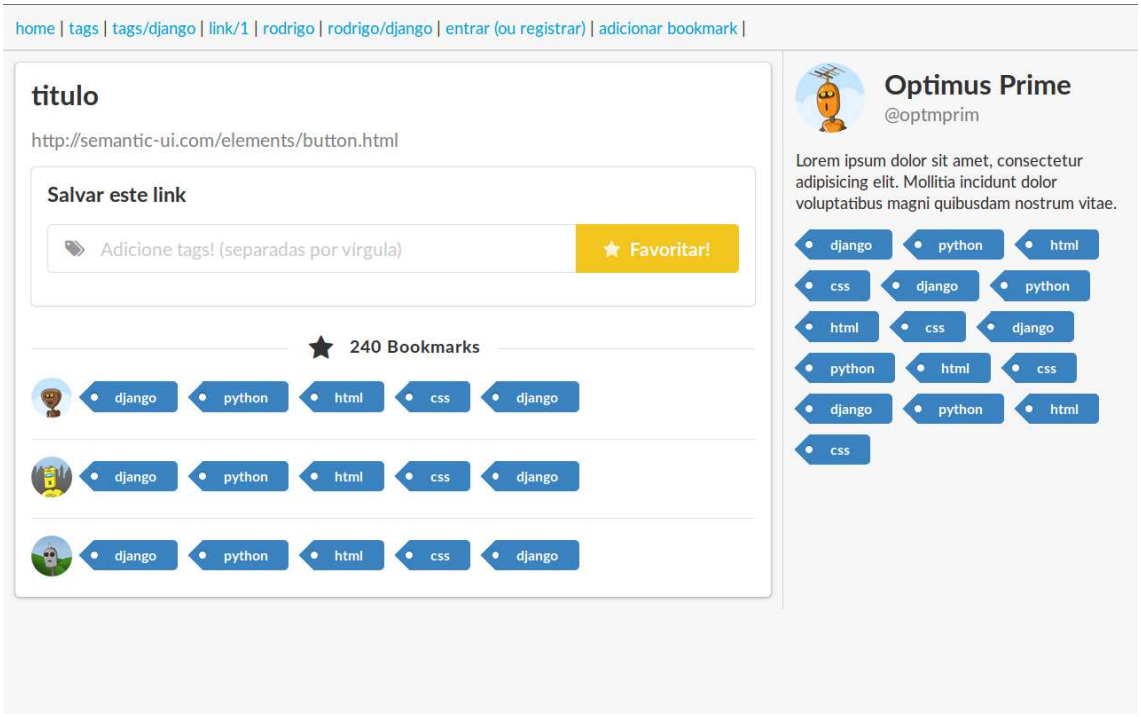
Fonte: Autoria própria.

Figura 23 - Protótipo da página "tag”.



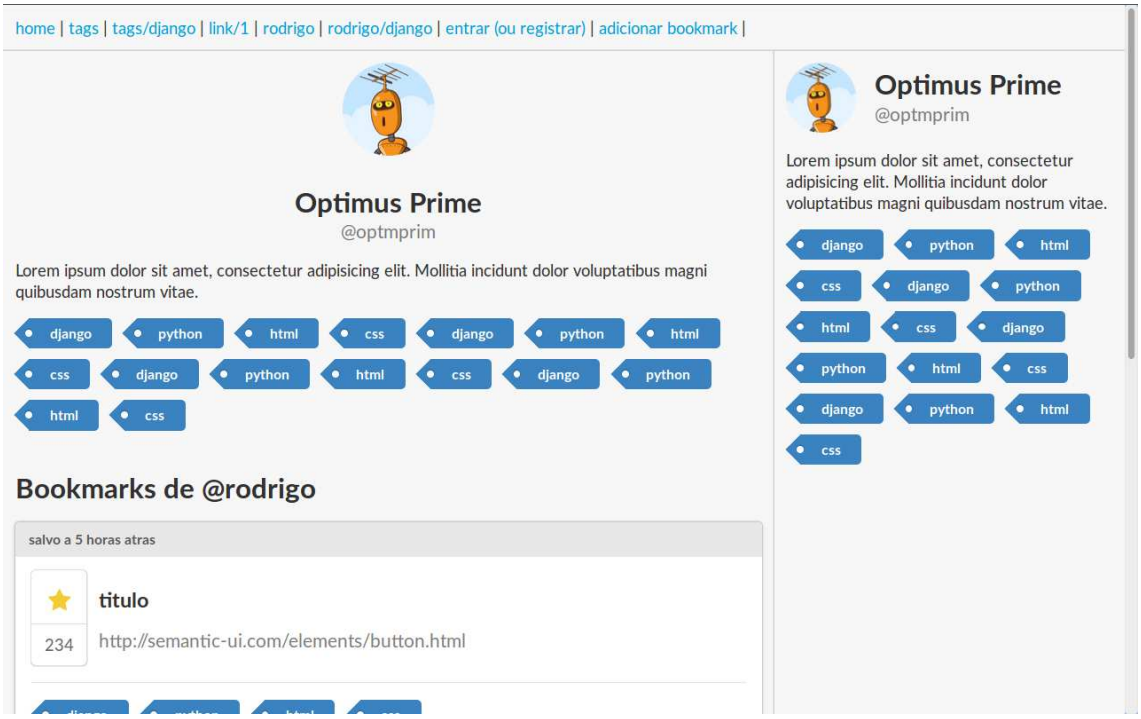
Fonte: Autoria própria.

Figura 24 - Protótipo da página “link”.



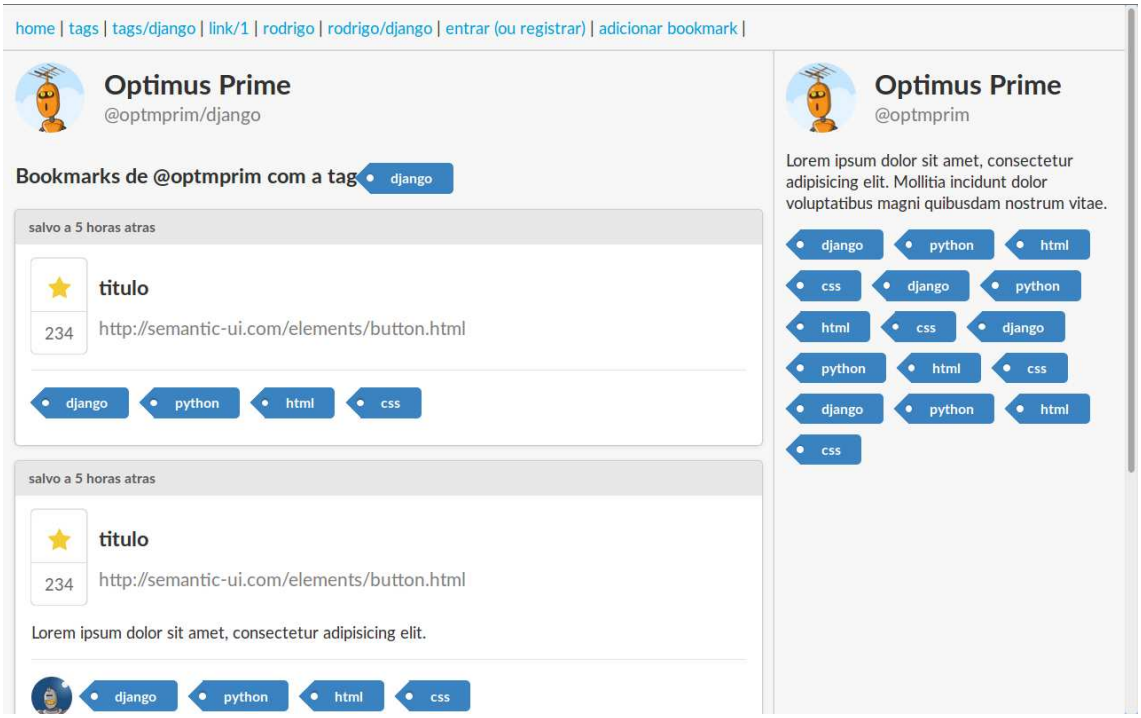
Fonte: Autoria própria.

Figura 25 - Protótipo da página "usuário".



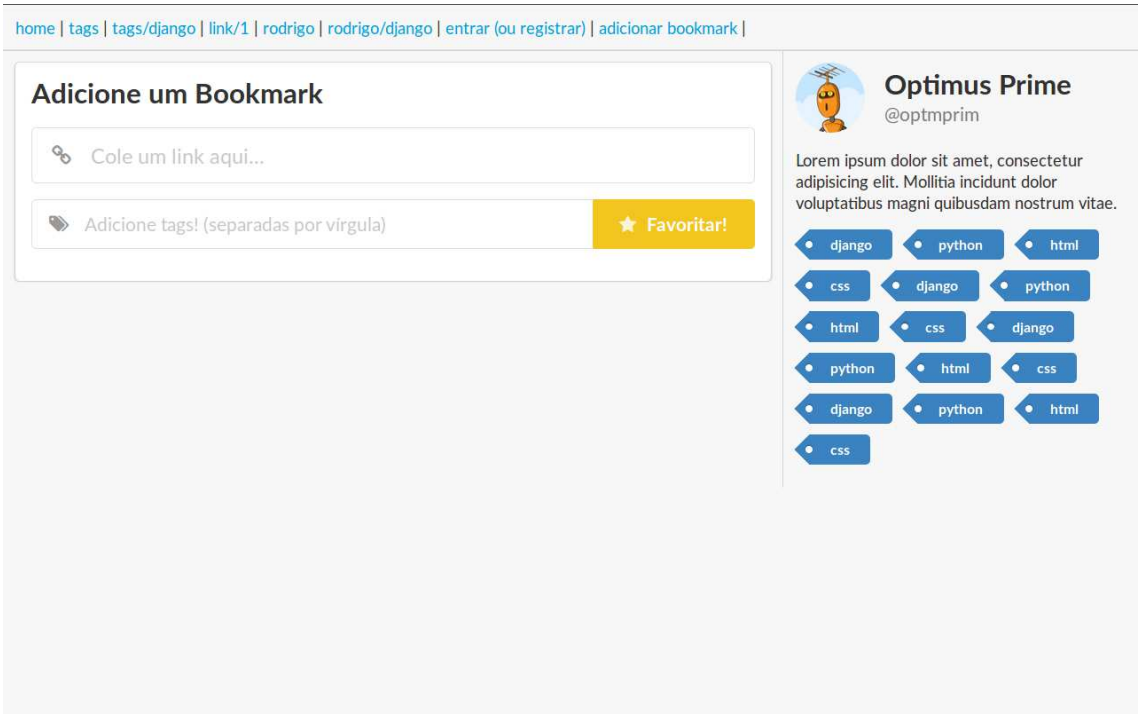
Fonte: Autoria própria.

Figura 26 - Protótipo da página "usuário-tag".



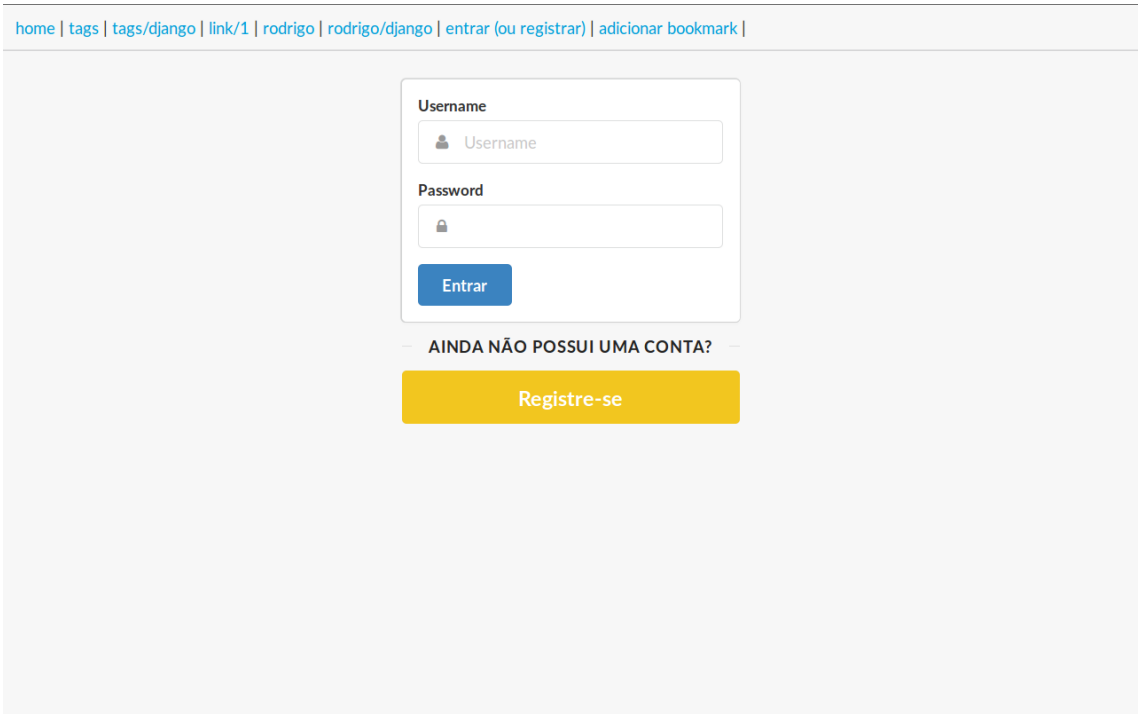
Fonte: Autoria própria.

Figura 27 - Protótipo da página "bookmark".



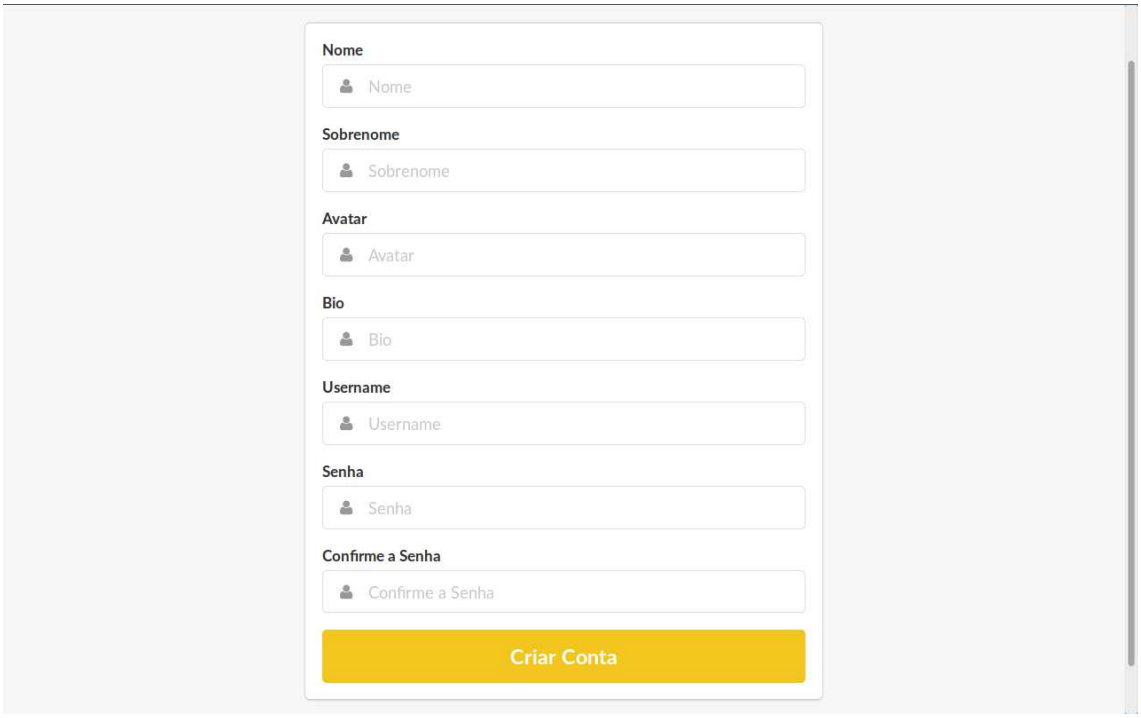
Fonte: Autoria própria.

Figura 28 - Protótipo da página "entrar".



Fonte: Autoria própria.

Figura 29 - Protótipo da página "registrar".



Fonte: Autoria própria.

4 RESULTADOS E DISCUSSÃO

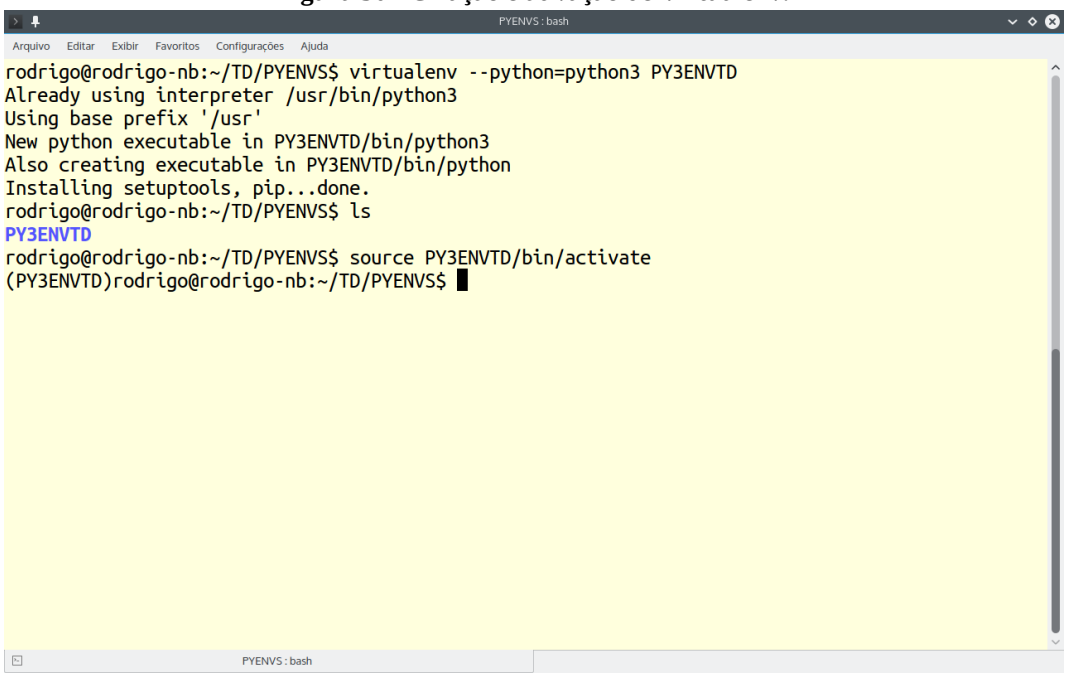
4.1 INICIALIZAÇÃO DO PROJETO

Neste capítulo são apresentadas as ações necessárias ao início do desenvolvimento de um projeto Django.

4.1.1 Preparação do ambiente Python e instalação do Django

O projeto demanda a criação de um ambiente Python exclusivo e isolado do sistema operacional. Para isto foi criado o ambiente virtual Python com a ferramenta Virtualenv. Em seguida o ambiente foi ativado (figura 30):

Figura 30 - Criação e ativação de Virtualenv.



Fonte: Autoria própria.

Após a ativação do ambiente, é necessário instalar Django no ambiente. Para isso foi utilizada a ferramenta Pip, que é instalada automaticamente junto com o Virtualenv (figura 31):

Figura 31 - Instalação do Django.

Fonte: Autoria própria.

4.1.2 Inicialização do projeto Django e criação da *app*

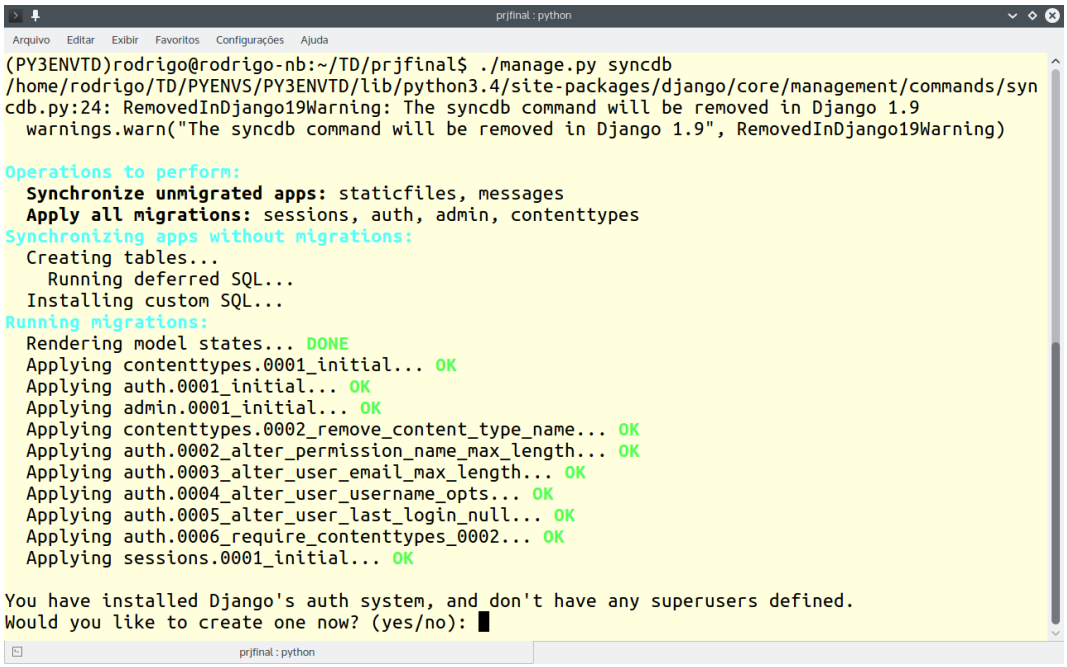
Para iniciar o projeto é necessário invocar o comando “django-admin startproject prjfinal” no terminal. Como resultado, o Django gera uma estrutura de diretórios contendo os arquivos iniciais do projeto, conforme a figura 32:

Figura 32 - Inicialização do projeto Django.

Fonte: Autoria própria.

Em seguida é executado o comando "./manage.py syncdb" para inicializar o banco de dados (figura 33):

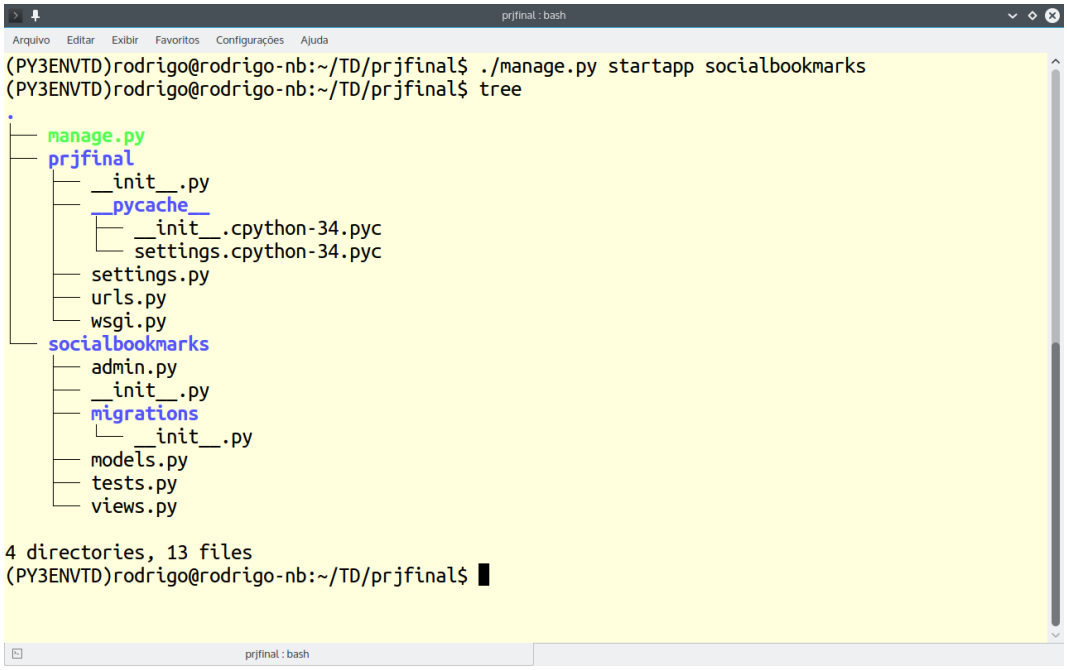
Figura 33 - Inicialização do banco de dados como comando "syncdb".



Fonte: Autoria própria.

Na figura 34 é demonstrada a criação de uma *app* Django com o comando “manage.py startapp socialbookmarks”. O Django gera a pasta com o mesmo nome da *app* contendo os arquivos que serão utilizados no desenvolvimento da aplicação:

Figura 34 - Inicialização de *app* Django.



Fonte: Autoria própria.

4.2 DESENVOLVIMENTO DA APLICAÇÃO COM TDD

Após a preparação do ambiente e criação do projeto, os arquivos gerados pelo Django podem ser editados para o desenvolvimento do software. Conforme a proposta deste trabalho, segue a construção do sistema utilizando o TDD. Nesta etapa os testes são escritos.

4.2.1 Criação do primeiro teste para *views*

A fim de aproveitar o protótipo criado anteriormente, é criado um teste para verificar se as páginas do sistema estão presentes em uma URL adequada (quadro 2):

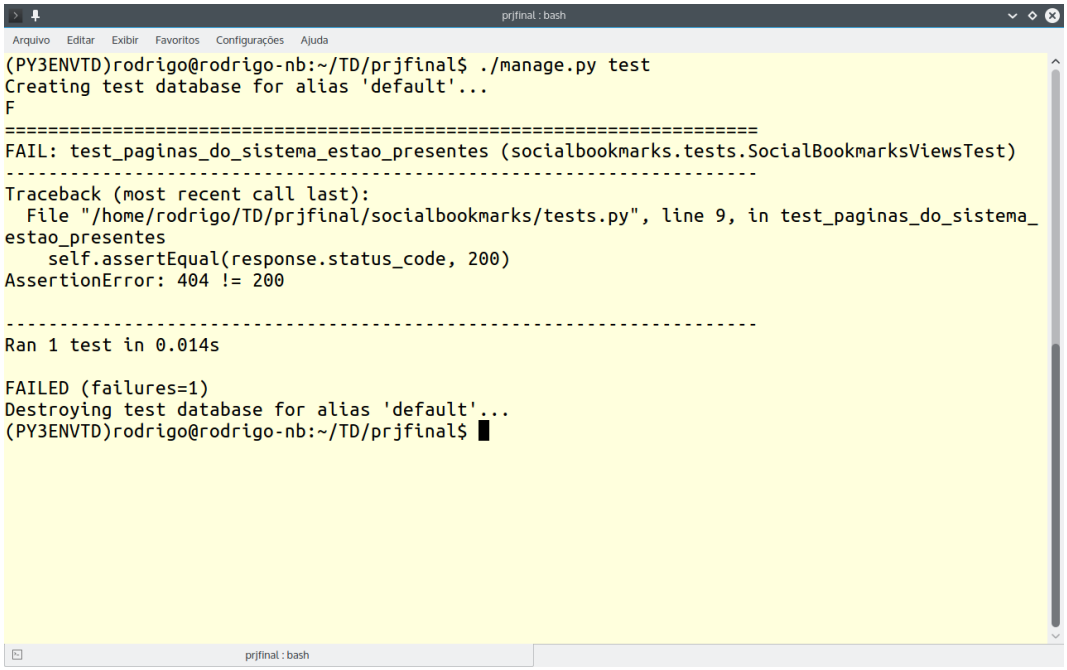
Quadro 2 - Testes para *views* falhando.

```
from django.test import TestCase


class SocialBookmarksViewsTest(TestCase):

    def test_paginas_do_sistema_estao_presentes(self):
        # página "home" está presente na url correta?
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'home.html')
        # página "tags" está presente na url correta?
        response = self.client.get('/tags')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'tags.html')
        # página "tag_bookmarks" está presente na url correta?
        response = self.client.get('/tag/tdd')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'tag_bookmarks.html')
        # página "link" está presente na url correta?
        response = self.client.get('/link/1')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'link.html')
        # página "usuário" está presente na url correta?
        response = self.client.get('/rodrigo')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'usuario.html')
        # página "usuário_tag" está presente na url correta?
        response = self.client.get('/rodrigo/tdd')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'usuario_tag.html')
        # página "adicionar_bookmark" está presente na url correta?
        response = self.client.get('/bookmark')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'adicionar_bookmark.html')
        # página "entrar" está presente na url correta?
        response = self.client.get('/entrar')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'entrar.html')
        # página "registro" está presente na url correta?
        response = self.client.get('/registro')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'registro.html')
```

Fonte: Autoria própria.

Neste estágio os testes são executados com o comando “./manage.py test” no terminal. O resultado é apresentado na figura 35:

Figura 35 - Teste para *views* falhando.



Fonte: Autoria própria.

A falha do teste indica que não há página sob a requisição. Isto ocorre porque as configurações de URL e template não foram definidas, e porque nenhuma função no arquivo “view.py” foi escrita para atender a requisição.

Django usa expressões regulares configuradas no módulo urls.py para analisar as URLs das requisições e invocar a *view* apropriada para cada padrão de URL (quadro 3):

Quadro 3 - Arquivo “urls.py”.

```
from django.conf import settings
from django.conf.urls import patterns, include, url
from django.conf.urls.static import static
from django.contrib import admin

urlpatterns = patterns(
    '',
    # fornece acesso a interface de administração fornecida pelo django
    url(r'^admin/', include(admin.site.urls)),
    # "/" chama função "home" em views.py
    url(r'^$', 'socialbookmarks.views.home', name='home'),
    # "/entrar" chama função "entrar" em views.py
    url(r'^entrar$', 'socialbookmarks.views.entrar', name='entrar'),
    # "/registro" chama função "registro" em views.py
    url(r'^registro$', 'socialbookmarks.views.registro', name='registro'),
    # "/bookmark" chama função "adicionar_bookmark" em views.py
    url(r'^bookmark$',
        'socialbookmarks.views.adicionar_bookmark', name='adicionar_bookmark'),
    # "/tags" chama função "tags" em views.py
    url(r'^tags$', 'socialbookmarks.views.tags', name='tags'),
    # "/tag/parâmetro" chama função "tag" fornecendo parâmetro "tag" em views.py
    url(r'^tag/(?P<tag>[\w_]+)$',
        'socialbookmarks.views.bookmarks_tag', name='bookmarks_tag'),
    # "/link/parâmetro" chama função "link" fornecendo parâmetro "link_id"
    # em views.py
    url(r'^link/(?P<link_id>\d+)$', 'socialbookmarks.views.link', name='link'),
    # "/parâmetro" chama função "usuário" fornecendo parâmetro "username"
    # em views.py
    url(r'^(?P<username>\w+)$',
        'socialbookmarks.views.usuario', name='usuario'),
    # "/parâmetro1/parâmetro2" chama função "usuário_tag" fornecendo parâmetros
    # "username" e "tag" em views.py
    url(r'^(?P<username>\w+)/(?P<tag>[\w_]+)$',
        'socialbookmarks.views.usuario_tag', name='usuario_tag'),
)
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Fonte: Autoria própria.

No arquivo “settings.py” foram definidas as configurações de *template* e arquivos estáticos (quadro 4). Também foram criados os diretórios “static” e “templates” no diretório base do projeto.

Quadro 4 - Arquivo “settings.py”.

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            # insert your TEMPLATE_DIRS here
            os.path.join(BASE_DIR,  'templates'),
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                # Insert your TEMPLATE_CONTEXT_PROCESSORS here or use this
                # list if you haven't customized them:
                'django.contrib.auth.context_processors.auth',
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.template.context_processors.i18n',
                'django.template.context_processors.media',
                'django.template.context_processors.static',
                'django.template.context_processors.tz',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Fonte: Autoria própria.

No quadro 5 pode-se visualizar o arquivo “views.py” somente com o código necessário para o teste passar:

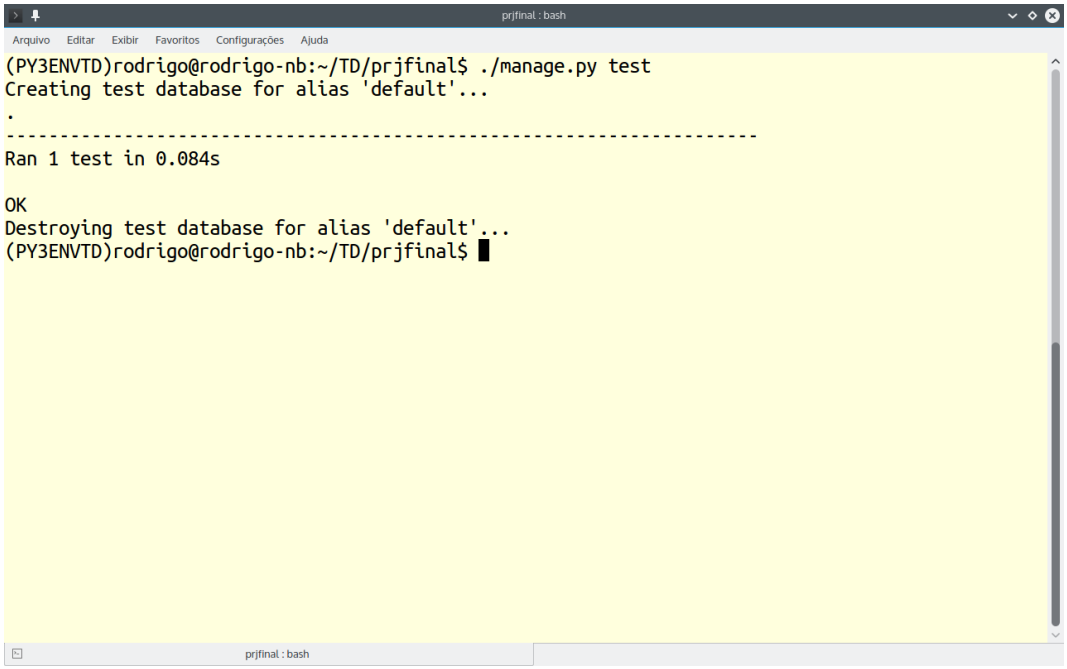
Quadro 5 - Código *views* para o teste passar.

```
from django.shortcuts import render


def home(request):
    template = 'home.html'
    return render(request, template)


def entrar(request):
    template = 'entrar.html'
    return render(request, template)


def registro(request):
    template = 'registro.html'
    return render(request, template)


def adicionar_bookmark(request):
    template = 'adicionar_bookmark.html'
    return render(request, template)


def tags(request):
    template = 'tags.html'
    return render(request, template)


def bookmarks_tag(request, tag):
    template = 'tag_bookmarks.html'
    return render(request, template)


def link(request, link_id):
    template = 'link.html'
    return render(request, template)


def usuario(request, username):
    template = 'usuario.html'
    return render(request, template)


def usuario_tag(request, username, tag):
    template = 'usuario_tag.html'
    return render(request, template)
```

Fonte: Autoria própria.

Ao repetir o mesmo teste, a saída do terminal indica que não houve erros (figura 36). Mais adiante este código será refatorado para atender aos testes funcionais.

Figura 36 - Teste para *views* passando.



Fonte: Autoria própria.

4.2.2 Criação dos testes para os modelos da *app*

Nesta etapa, é escrito um teste para os supostos modelos necessários para a realização das funcionalidades desejadas no sistema (quadro 6):

Quadro 6 - Teste para modelos da *app* Django.

```
from django.test import TestCase
from socialbookmarks.models import User, Usuario, Link, Bookmark, Tag


class SocialBookmarksModelsTest(TestCase):

    def setUp(self):
        # prepara os testes registrando um usuário
        user_teste = User(username='rodrigo', password='1234',
                          email='rodrigodeoliveira.ti@gmail.com')
        user_teste.save()
        Usuario(user=user_teste, avatar_uploaded='img.jpg',
                bio='''Lorem ipsum dolor sit amet, consectetur
                       adipisicing elit.''').save()

    def test_usuario_adiciona_bookmark(self):
        # pega um usuário registrado e salva um bookmark
        usuario1 = Usuario.objects.get(id=1)
        link1 = Link(url='python.org', titulo='Welcome to Python.org')
        tag1 = Tag(tag='python')
        tag2 = Tag(tag='django')
        link1.save()
        tag1.save()
        tag2.save()
        bookmark1 = Bookmark(link=link1, usuario=usuario1)
        bookmark1.save()
        bookmark1.tags.add(tag1)
        bookmark1.tags.add(tag2)
        link2 = Link(url='www.djangoproject.com', titulo='Django Framework')
        link2.save()
        bookmark2 = Bookmark(link=link2, usuario=usuario1)
        bookmark2.save()
        bookmark2.tags.add(tag1)
        # verifica se os dados do bookmark salvo são iguais aos enviados
        self.assertEqual(Bookmark.objects.count(), 2)
```

Fonte: Autoria própria.

Ao executar o teste, foi verificada não uma falha, mas um erro, indicando que o teste não pode ser executado corretamente. Isso é útil para indicar outros tipos de problemas. Neste caso, não existe uma tabela correspondente aos modelos no banco de dados (quadro 7):

Quadro 7 - Teste para modelos falhando.

```
======================================================================

ERROR: test_usuario_adiciona_bookmark (socialbookmarks.tests.SocialBookmarksModelsTest)

----------------------------------------------------------------------

The above exception was the direct cause of the following exception:

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/db/backends/sqlite3/base.py", line 318, in execute

    return Database.Cursor.execute(self, query, params)

django.db.utils.OperationalError: no such table: socialbookmarks_usuario

----------------------------------------------------------------------

Ran 2 tests in 0.100s

FAILED (errors=1)
```

Fonte: Autoria própria.

Foram definidos no código referente ao arquivo “models.py” os campos de dados necessários para a aplicação (quadro 8):

Quadro 8 - Arquivo “models.py”.

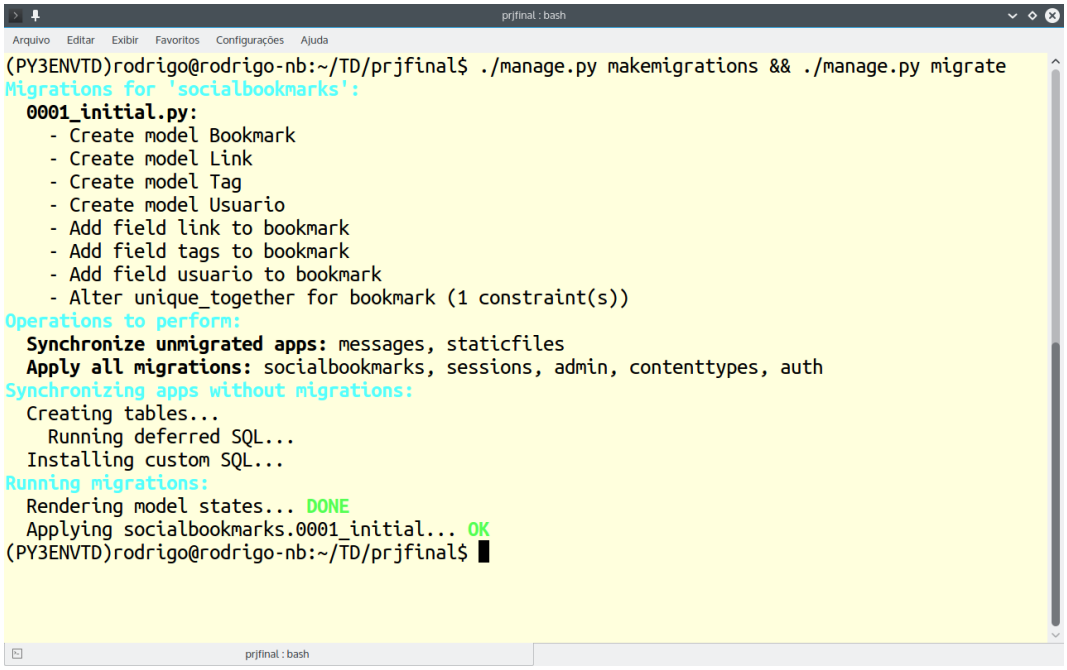
```
from django.db import models
from django.contrib.auth.models import User
import re


class Tag(models.Model):
    tag = models.CharField(unique=True, max_length=30)
    tag_slug = models.CharField(editable=False, max_length=30)

    def save(self, *args, **kwargs):
        # formata tag para ser ser amigável a url
        self.tag = re.sub('\s+', ' ', self.tag)
        self.tag_slug = re.sub('\s+', '_', self.tag)
        super(Tag, self).save(*args, **kwargs)

    def __str__(self):
        return self.tag


class Link(models.Model):
    url = models.URLField(unique=True)
    titulo = models.CharField(max_length=120)

    def __str__(self):
        return self.url


class Usuario(models.Model):
    user = models.ForeignKey(User)
    avatar_uploaded = models.ImageField(upload_to='uploads/avatars')
    bio = models.TextField(max_length=120)

    def __str__(self):
        return str(self.user)


class Bookmark(models.Model):
    usuario = models.ForeignKey(Usuario)
    link = models.ForeignKey('Link')
    tags = models.ManyToManyField('Tag')
    adicionado_em = models.DateTimeField(auto_now_add=True, editable=True)
    descricao = models.TextField(max_length=120)

    class Meta:
        unique_together = ("usuario", "link")

    def __str__(self):
        return str('%s: %s' % (self.usuario, self.link))
```

Fonte: Autoria própria.

Após a definição dos modelos, são realizadas as migrações do banco de dados, através dos comandos “makemigrations”, e em seguida “migrate”. Estas operações alteram o banco de dados adicionando as tabelas conforme os modelos (figura 37).

Para que o Django identifique as tarefas que devem ser realizadas no banco de dados, é preciso declarar a *app* Django como instalada no arquivo “settings.py”.

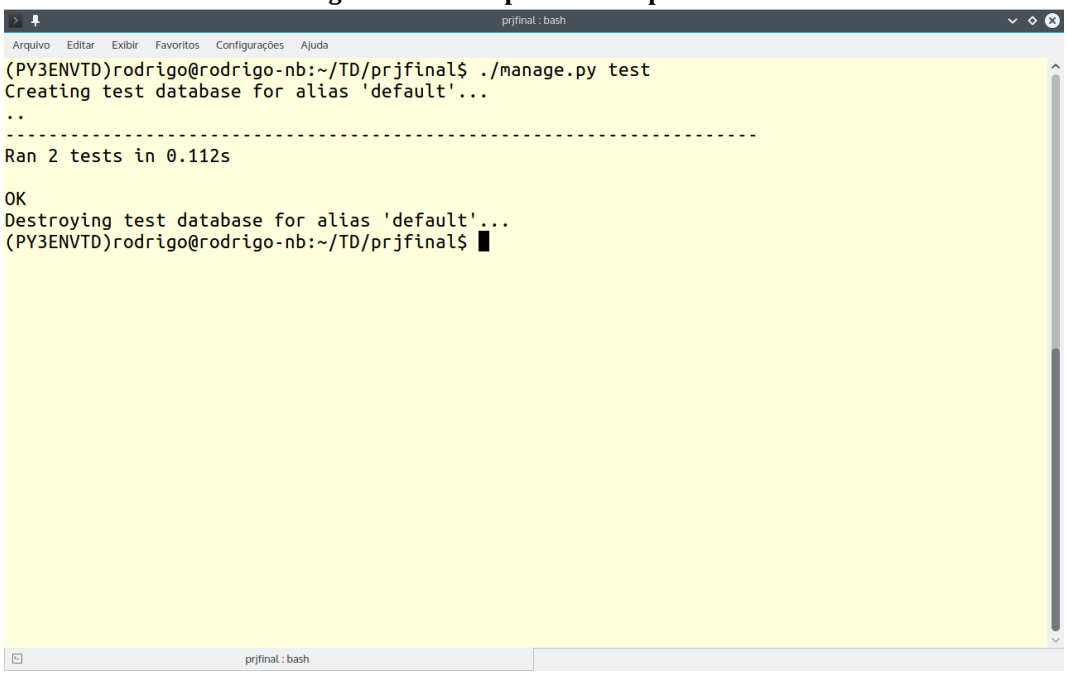
Figura 37 - Migração da *app* “socialbookmarks”.



Fonte: Autoria própria.

Realizando os testes novamente, verifica-se a validação (figura 38):

Figura 38 - Teste para *models* passando.



Fonte: Autoria própria.

4.2.3 Testes de validação de dados das funções do arquivo “views.py”

Nesta etapa é feita a verificação da existência dos dados na resposta de uma requisição, de acordo com a demanda do usuário.

A função “setUp” prepara os testes que seguem, simulando o registro de um usuário que salva 2 *bookmarks*. Em seguida os demais testes verificam se a contagem dos dados confere com o que foi definido anteriormente (quadro 9):

Quadro 9 - Teste para verificar dados de contexto.

```
class SocialBookmarksViewsTest(TestCase):

    # preparação dos dados para os testes

    def setUp(self):
        # registrando um usuário que salva 2 bookmarks
        user_teste = User(username='rodrigo', password='1234',
                          email='rodrigodeoliveira.ti@gmail.com')
        user_teste.save()
        usuario_teste = Usuario(user=user_teste, avatar_uploaded='img.jpg',
                                bio='''Lorem ipsumdolor sit amet, consectetur
                                       adipisicing elit.''')
        usuario_teste.save()
        link1 = Link(url='python.org', titulo='Welcome to Python.org')
        tag1 = Tag(tag='python')
        tag2 = Tag(tag='django')
        link1.save()
        tag1.save()
        tag2.save()
        bookmark1 = Bookmark(link=link1, usuario=usuario_teste)
        bookmark1.save()
        bookmark1.tags.add(tag1)
        bookmark1.tags.add(tag2)
        link2 = Link(url='www.djangoproject.com', titulo='Django Framework')
        link2.save()
        bookmark2 = Bookmark(link=link2, usuario=usuario_teste)
        bookmark2.save()
        bookmark2.tags.add(tag1)

    # teste de dados do contexto

    def test_home_retorna_ultimos_bookmarks(self):
        response = self.client.get('/')
        # verifica se o número de bookmarks corresponde
        self.assertEqual(response.context['ultimos_bookmarks'].count(), 2)

    def test_tags_retorna_todas_as_tags(self):
        response = self.client.get('/tags')
        # verifica se o número de tags corresponde
        self.assertEqual(response.context['tags'].count(), 2)

    def test_tag_retorna_bookmarks_com_a_tag(self):
        # pega uma tag qualquer para utilizar como parâmetro da url
        tag_teste = Tag.objects.get(id=1)
        response = self.client.get('/tag/%s' % (tag_teste.tag_slug, ))
        # verifica se o número de bookmarks com a tag corresponde
        self.assertEqual(response.context['bookmarks_tag'].count(), 2)

    def test_link_retorna_links_e_bookmarks(self):
        # pega um link qualquer para utilizar como parâmetro da url
        link_teste = Link.objects.get(id=1)
        response = self.client.get('/link/%s' % (link_teste.id, ))
        '''verifica se o link corresponde ao solicitado e se o número de
           Bookmarks está correto
        '''
        self.assertEqual(response.context['link'], link_teste)
        self.assertEqual(response.context['link_bookmarks'].count(), 1)
```

```
    def test_usuario_retorna_bookmarks_do_usuario(self):
        # pega um usuário para utilizar como parâmetro da url
        usuario_teste = Usuario.objects.get(id=1)
        response = self.client.get('/%s' % (usuario_teste.user.username, ))
        # verifica se o número de bookmarks corresponde
        self.assertEqual(response.context['u_bookmarks'].count(), 2)

    def test_usuario_tag_retorna_bookmarks_do_usuario_com_a_tag(self):
        # pega um usuário e uma tag para utilizar como parâmetro da url
        usuario_teste = Usuario.objects.get(id=1)
        tag_teste = Tag.objects.get(id=1)
        response = self.client.get('/%s/%s'
                                   % (usuario_teste.user.username, tag_teste))
        # verifica se corresponde o número de bookmarks do usuário com a tag
        self.assertEqual(response.context['u_bookmarks_tag'].count(), 2)
```

Fonte: Autoria própria.

Ao executar o teste, observam-se erros decorrentes da inexistência dos dados de contexto. Deste modo, as assertivas não podem ser verificadas (quadro 10):

Quadro 10 - Erro nos testes.

```
.EE.EEEE

======================================================================

ERROR: test_home_retorna_ultimos_bookmarks
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 112, in
test_home_retorna_ultimos_bookmarks

    self.assertEqual(response.context['ultimos_bookmarks'].count(), 2)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/template/context.py", line 71, in __getitem__

    raise KeyError(key)

KeyError: 'ultimos_bookmarks'


======================================================================

ERROR: test_link_retorna_links_e_bookmarks
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------
```

```
Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 133, in
test_link_retorna_links_e_bookmarks

    self.assertEqual(response.context['link'], link_teste)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/template/context.py", line 71, in __getitem__

    raise KeyError(key)

KeyError: 'link'


======================================================================

ERROR: test_tag_retorna_bookmarks_com_a_tag
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 124, in
test_tag_retorna_bookmarks_com_a_tag

    self.assertEqual(response.context['bookmarks_tag'].count(), 2)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/template/context.py", line 71, in __getitem__

    raise KeyError(key)

KeyError: 'bookmarks_tag'


======================================================================

ERROR: test_tags_retorna_todas_as_tags (socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 117, in
test_tags_retorna_todas_as_tags

    self.assertEqual(response.context['tags'].count(), 2)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/template/context.py", line 71, in __getitem__

    raise KeyError(key)

KeyError: 'tags'
```

```
======================================================================

ERROR: test_usuario_retorna_bookmarks_do_usuario
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 141, in
test_usuario_retorna_bookmarks_do_usuario

    self.assertEqual(response.context['u_bookmarks'].count(), 2)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/template/context.py", line 71, in __getitem__

    raise KeyError(key)

KeyError: 'u_bookmarks'


======================================================================

ERROR: test_usuario_tag_retorna_bookmarks_do_usuario_com_a_tag
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 150, in
test_usuario_tag_retorna_bookmarks_do_usuario_com_a_tag

    self.assertEqual(response.context['u_bookmarks_tag'].count(), 2)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/template/context.py", line 71, in __getitem__

    raise KeyError(key)

KeyError: 'u_bookmarks_tag'


----------------------------------------------------------------------

Ran 8 tests in 0.333s

 FAILED (errors=6)
```

Fonte: Autoria própria.

O código da “views” foi refatorado para incluir os dados necessários para os testes, porém foram propositalmente deixados como vazios, a fim de tornar possível a observação da falha nas assertivas (quadro 11):

Quadro 11 - Arquivo *views* com *queries* vazias.

```
from django.shortcuts import render
from socialbookmarks.models import Tag, Link, Bookmark, Usuario


def home(request):
    template = 'home.html'
    ultimos_bookmarks = Bookmark.objects.none()  # TODO: query objects

    context = {'ultimos_bookmarks': ultimos_bookmarks}
    return render(request, template, context)


def entrar(request):
    template = 'entrar.html'
    return render(request, template)


def registro(request):
    template = 'registro.html'
    return render(request, template)


def adicionar_bookmark(request):
    template = 'adicionar_bookmark.html'
    return render(request, template)


def tags(request):
    template = 'tags.html'
    tags = Tag.objects.none()  # TODO: query objects
    context = {'tags': tags}
    return render(request, template, context)


def bookmarks_tag(request, tag):
    template = 'tag_bookmarks.html'
    bookmarks_tag = Bookmark.objects.none()  # TODO: query objects
    context = {'bookmarks_tag': bookmarks_tag}
    return render(request, template, context)


def link(request, link_id):
    template = 'link.html'
    link = Link.objects.none()  # TODO: query objects
    link_bookmarks = Bookmark.objects.none()  # TODO: query objects
    context = {'link': link,
               'link_bookmarks': link_bookmarks}
    return render(request, template, context)


def usuario(request, username):
    template = 'usuario.html'
    user = Usuario.objects.none()  # TODO: query objects
    tags_usuario = Tag.objects.none()  # TODO: query objects
    u_bookmarks = Bookmark.objects.none()  # TODO: query objects
    context = {'usuario': user,
```

```
                'u_bookmarks': u_bookmarks,
                'tags_usuario': tags_usuario}
    return render(request, template, context)


def usuario_tag(request, username, tag):
    template = 'usuario_tag.html'
    usuario = Usuario.objects.none()  # TODO: query objects
    tags_usuario = Tag.objects.none()  # TODO: query objects
    u_bookmarks_tag = Bookmark.objects.none()  # TODO: query objects
    context = {'usuario': usuario,
                'tag': tag,
                'u_bookmarks_tag': u_bookmarks_tag,
                'tags_usuario': tags_usuario}
    return render(request, template, context)
```

Fonte: Autoria própria.

Ao executar novamente os testes, observa-se que eles apresentam falhas decorrentes de verificação de assertivas (quadro 12):

Quadro 12 - Testes para *views* falhando.

```
.FF.FFFF

======================================================================

FAIL: test_home_retorna_ultimos_bookmarks
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 112, in
test_home_retorna_ultimos_bookmarks

    self.assertEqual(response.context['ultimos_bookmarks'].count(), 2)

AssertionError: 0 != 2


======================================================================

FAIL: test_link_retorna_links_e_bookmarks
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 133, in
test_link_retorna_links_e_bookmarks

    self.assertEqual(response.context['link'], link_teste)
```

```
AssertionError: [] != <Link: python.org>


======================================================================

FAIL: test_tag_retorna_bookmarks_com_a_tag
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 124, in
test_tag_retorna_bookmarks_com_a_tag

    self.assertEqual(response.context['bookmarks_tag'].count(), 2)

AssertionError: 0 != 2


======================================================================

FAIL: test_tags_retorna_todas_as_tags (socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 117, in
test_tags_retorna_todas_as_tags

    self.assertEqual(response.context['tags'].count(), 2)

AssertionError: 0 != 2


======================================================================

FAIL: test_usuario_retorna_bookmarks_do_usuario
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 141, in
test_usuario_retorna_bookmarks_do_usuario

    self.assertEqual(response.context['u_bookmarks'].count(), 2)

AssertionError: 0 != 2


======================================================================

FAIL: test_usuario_tag_retorna_bookmarks_do_usuario_com_a_tag
(socialbookmarks.tests.SocialBookmarksViewsTest)
```

```
----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 150, in
test_usuario_tag_retorna_bookmarks_do_usuario_com_a_tag

    self.assertEqual(response.context['u_bookmarks_tag'].count(), 2)

AssertionError: 0 != 2


----------------------------------------------------------------------

Ran 8 tests in 0.333s

 FAILED (failures=6)
```

Fonte: Autoria própria.

Com o intuito de atender as assertivas dos testes, foi desenvolvido o código para que as funções do arquivo “views.py” retornem no contexto os dados de acordo com o esperado (quadro 13).

Quadro 13 - Código do arquivo views.py refatorado.

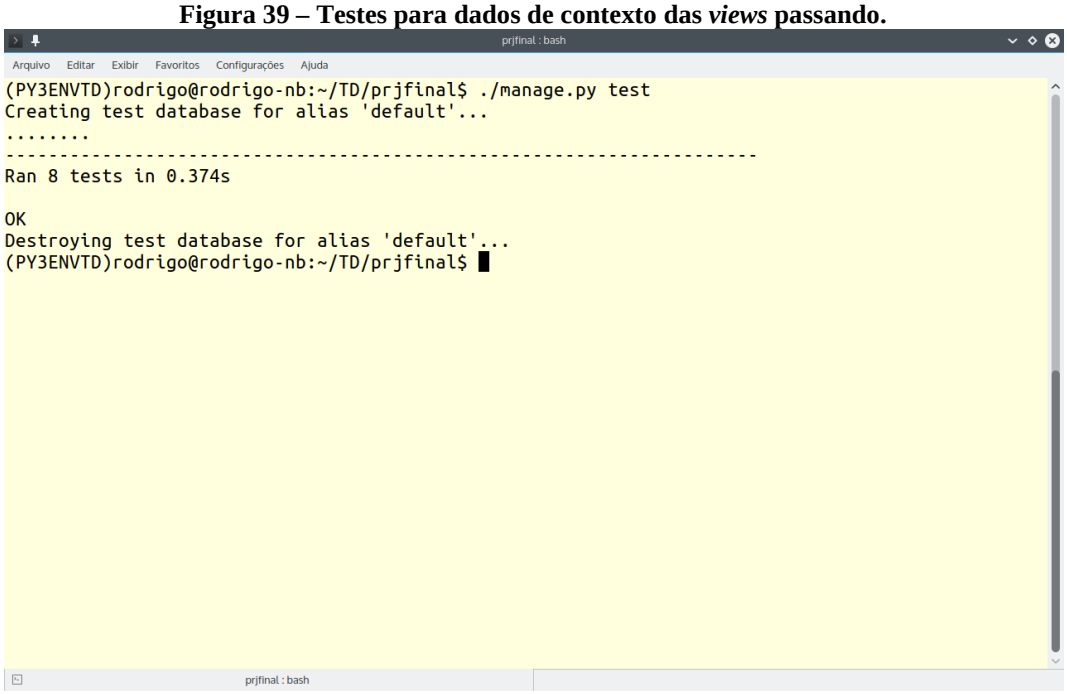
```
from django.shortcuts import render
from django.db.models import Count
from socialbookmarks.models import Tag, Link, Bookmark, Usuario


def home(request):
    template = 'home.html'
    ultimos_bookmarks = Bookmark.objects.annotate(
        saves_count=Count('link__bookmark'))
    context = {'ultimos_bookmarks': ultimos_bookmarks}
    return render(request, template, context)


def entrar(request):
    template = 'entrar.html'
    return render(request, template)


def registro(request):
    template = 'registro.html'
    return render(request, template)


def adicionar_bookmark(request):
    template = 'adicionar_bookmark.html'
    return render(request, template)


def tags(request):
    template = 'tags.html'
    tags = Tag.objects.all()
    context = {'tags': tags}
    return render(request, template, context)


def bookmarks_tag(request, tag):
    template = 'tag_bookmarks.html'
    bookmarks_tag = Bookmark.objects.filter(tags__tag_slug=tag).annotate(
        saves_count=Count('link__bookmark'))
    context = {'bookmarks_tag': bookmarks_tag}
    return render(request, template, context)


def link(request, link_id):
    template = 'link.html'
    link = Link.objects.get(id=link_id)
    link_bookmarks = Bookmark.objects.filter(link_id=link_id)
    context = {'link': link,
               'link_bookmarks': link_bookmarks}
    return render(request, template, context)


def usuario(request, username):
    template = 'usuario.html'
    user = Usuario.objects.get(user__username=username)
    tags_usuario = Tag.objects.filter(bookmark__usuario=user)
    u_bookmarks = Bookmark.objects.filter(usuario=user).annotate(
```

```
        saves_count=Count('link__bookmark'))
    context = {'usuario': user,
               'u_bookmarks': u_bookmarks,
               'tags_usuario': tags_usuario}
    return render(request, template, context)


def usuario_tag(request, username, tag):
    template = 'usuario_tag.html'
    usuario = Usuario.objects.get(user__username=username)
    tags_usuario = Tag.objects.filter(bookmark__usuario=usuario)
    u_bookmarks_tag = Bookmark.objects.filter(usuario=usuario).filter(
        tags__tag_slug=tag).annotate(
        saves_count=Count('link__bookmark'))
    context = {'usuario': usuario,
               'tag': tag,
               'u_bookmarks_tag': u_bookmarks_tag,
               'tags_usuario': tags_usuario}
    return render(request, template, context)
```

Fonte: Autoria própria.

Execução dos testes após refatoração do código demonstram que todos os testes passam (figura 39):

Figura 39 – Testes para dados de contexto das *views* passando.



Fonte: Autoria própria.

4.2.4 Teste para dados nos *templates*

Após a verificação dos testes anteriores, é necessário observar se os dados oferecidos na resposta de uma requisição são utilizados na renderização dos *templates*. Os testes verificam se a resposta, que utiliza determinado *template*, contem um dos dados de contexto (quadro 14):

Quadro 14 - Testes para dados nos *templates*.

```
    # teste templates utilizam dados do contexto

    def test_template_home_utiliza_dados_do_contexto(self):
        response = self.client.get('/')
        self.assertContains(response,
                            response.context['ultimos_bookmarks'][0].link)
        self.assertContains(response, response.context['usuario_logado'])

    def test_template_tag_utiliza_dados_do_contexto(self):
        response = self.client.get('/tags')
        self.assertContains(response, response.context['tags'][0])
        self.assertContains(response, response.context['usuario_logado'])

    def test_template_tags_utiliza_dados_do_contexto(self):
        response = self.client.get('/tag/django')
        self.assertContains(response,
                            response.context['bookmarks_tag'][0].link)
        self.assertContains(response, response.context['usuario_logado'])

    def test_template_link_utiliza_dados_do_contexto(self):
        response = self.client.get('/link/1')
        self.assertContains(response, response.context['link'])
        self.assertContains(response,
                            response.context['link_bookmarks'][0].usuario)
        self.assertContains(response, response.context['usuario_logado'])

    def test_template_usuario_utiliza_dados_do_contexto(self):
        response = self.client.get('/rodrigo')
        self.assertContains(response, response.context['u_bookmarks'][0].link)
        self.assertContains(response, response.context['usuario_logado'])

    def test_template_usuario_tag_utiliza_dados_do_contexto(self):
        response = self.client.get('/rodrigo/python')
        self.assertContains(response,
                            response.context['u_bookmarks_tag'][0].link)
        self.assertContains(response, response.context['usuario_logado'])

    def test_template_bookmark_utiliza_dados_do_contexto(self):
        response = self.client.get('/bookmark')
        self.assertContains(response, response.context['usuario_logado'])
```

Fonte: Autoria própria.

No quadro 15 são demonstradas as falhas dos testes, considerando que os *templates* utilizados ainda possuem o mesmo código do protótipo HTML. Desta forma, são necessárias adaptações para que se tornem *templates* Django funcionais.

Quadro 15 - Teste para *templates* falhando.

```
.......FF.FFF..
======================================================================

FAIL: test_template_home_utiliza_dados_do_contexto
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 120, in
test_template_home_utiliza_dados_do_contexto

    response.context['ultimos_bookmarks'][0].link)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/test/testcases.py", line 358, in assertContains

    msg_prefix + "Couldn't find %s in response" % text_repr)

AssertionError: False is not true : Couldn't find 'python.org' in response


======================================================================

FAIL: test_template_link_utiliza_dados_do_contexto
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 136, in
test_template_link_utiliza_dados_do_contexto

    self.assertContains(response, response.context['link'])

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/test/testcases.py", line 358, in assertContains

    msg_prefix + "Couldn't find %s in response" % text_repr)

AssertionError: False is not true : Couldn't find 'python.org' in response


======================================================================

FAIL: test_template_tags_utiliza_dados_do_contexto
(socialbookmarks.tests.SocialBookmarksViewsTest)
```

```
----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 131, in
test_template_tags_utiliza_dados_do_contexto

    response.context['bookmarks_tag'][0].link)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/test/testcases.py", line 358, in assertContains

    msg_prefix + "Couldn't find %s in response" % text_repr)

AssertionError: False is not true : Couldn't find 'python.org' in response


======================================================================

FAIL: test_template_usuario_tag_utiliza_dados_do_contexto
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 149, in
test_template_usuario_tag_utiliza_dados_do_contexto

    response.context['u_bookmarks_tag'][0].link)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/test/testcases.py", line 358, in assertContains

    msg_prefix + "Couldn't find %s in response" % text_repr)

AssertionError: False is not true : Couldn't find 'python.org' in response


======================================================================

FAIL: test_template_usuario_utiliza_dados_do_contexto
(socialbookmarks.tests.SocialBookmarksViewsTest)

----------------------------------------------------------------------

Traceback (most recent call last):

  File "/home/rodrigo/TD/prjfinal/socialbookmarks/tests.py", line 143, in
test_template_usuario_utiliza_dados_do_contexto

    self.assertContains(response, response.context['u_bookmarks'][0].link)

  File "/home/rodrigo/TD/PYENVS/PY3ENVTD/lib/python3.4/site-
packages/django/test/testcases.py", line 358, in assertContains

    msg_prefix + "Couldn't find %s in response" % text_repr)
```

```
AssertionError: False is not true : Couldn't find 'python.org' in response

----------------------------------------------------------------------

Ran 15 tests in 3.724s

 FAILED (failures=5)
```

Fonte: Autoria própria.

Refatoração dos *templates* para que utilizem os dados de contexto de modo adequado (quadros 16 a 22):

Quadro 16 - Refatoração do *template* "home".

```
                <h2 class="ui header centered">Últimos Bookmarks</h2>
                 {% for bookmark in ultimos_bookmarks %}
                 <div class="ui raised segment">
                     <div class="ui top attached label">{{ bookmark.adicionado_em
}}</div>
                     <div class="ui header">
                         <div class="ui vertical basic buttons icon">
                             <a href="/link/{{ bookmark.link.id }}" class="ui button"><i
class="icon favorite large yellow"></i>
                                 <p><strong>{{ bookmark.saves_count }}</strong></p>
                             </a>
                         </div>
                         <div class="content">
                             <h3>
                                 <a href="{{ bookmark.link.url }}" target="_blank">{{
bookmark.link.titulo }}</a>
                             </h3>
                             <div class="sub header">{{ bookmark.link.url }}</div>
                         </div>
                     </div>
                     <p>{{ bookmark.descricao }}</p>
                     <div class="ui divider">
                     </div>
                     <div class="ui tag blue labels">
                         <a href="/{{ bookmark.usuario.user.username }}"><img class="ui
avatar image middle aligned" src="{{ MEDIA_URL }}{{ bookmark.usuario.avatar_uploaded
}}"></a>
                         {% for tag in bookmark.tags.all %}
                         <a href="/{{ bookmark.usuario.user.username }}/{{ tag.tag_slug
}}" class="ui label">{{ tag }}</a>
                         {% endfor %}
                     </div>
                 </div>
                 {% endfor %}
```

Fonte: Autoria própria.

Quadro 17 - Refatoração do *template* “sidebar”.

```
            <div class="five wide column">
                <h2 class="ui header">
                    <img class="ui circular image" src="{{ MEDIA_URL }}{{
usuario_logado.avatar_uploaded }}">
                    <div class="content">
                        <a href="/{{ usuario_logado.user.username }}">{{ usuario_logado
}}</a>
                        <div class="sub header">
                            <a href="/{{ usuario_logado.user.username }}">@{{
usuario_logado.user.username }}
                            </a>
                        </div>
                    </div>
                </h2>
                <p>{{ usuario_logado.bio }}</p>
                <div class="ui tag blue labels">

                    {% for tag in usuario_logado.tags.all %}

                    <a href="/{{ usuario_logado.user.username }}/{{ tag.tag_slug }}"
class="ui label">{{ tag }}</a>

                    {% endfor %}

                </div>
            </div>
```

Fonte: Autoria própria.

Quadro 18 - Refatoração do *template* “tags”.

```
                <h2 class="ui header centered">Tags</h2>
                <div class="ui raised segment">
                    <div class="ui tag blue labels big">

                        {% for tag in tags %}

                        <a href="/tag/{{ tag }}" class="ui label">{{ tag }}</a>

                        {% endfor %}

                    </div>
                </div>
```

Fonte: Autoria própria.

Quadro 19 - Refatoração do *template* "tag".

```
                <h2 class="ui header centered">Bookmarks salvos com a tag
                    <span class="ui tag label big blue">{{ tag }}</span>
                </h2>

                {% for bookmark in bookmarks_tag %}

                <div class="ui raised segment">
                    <div class="ui top attached label">{{ bookmark.adicionado_em
}}</div>
                    <div class="ui header">
                        <div class="ui vertical basic buttons icon">
                            <a href="/link/{{ bookmark.link.id }}" class="ui button">
                                <i class="icon favorite large yellow"></i>
                                <p>
                                    <strong>{{ bookmark.saves_count }}</strong>
                                </p>
                            </a>
                        </div>
                        <div class="content">
                            <h3>
                                <a href="{{ bookmark.link.url }}" target="_blank">{{
bookmark.link.titulo }}</a>
                            </h3>
                            <div class="sub header">{{ bookmark.link.url }}</div>
                        </div>
                    </div>
                    <p>{{ bookmark.descricao }}</p>
                    <div class="ui divider">
                    </div>
                    <div class="ui tag blue labels">
                        <a href="/{{ bookmark.usuario.user.username }}">
                            <img class="ui avatar image middle aligned" src="{{
MEDIA_URL }}{{ bookmark.usuario.avatar_uploaded }}">
                        </a>

                        {% for tag in bookmark.tags.all %}

                        <a href="/{{ bookmark.usuario.user.username }}/{{ tag.tag_slug
}}" class="ui label">{{ tag }}</a>

                        {% endfor %}

                    </div>
                </div>

                {% endfor %}
```

Fonte: Autoria própria.

Quadro 20 - Refatoração do *template* “link”.

```
        <div class="ui raised segment">
                <div class="ui header">
                        <div class="content">
                            <h2>
                                <a href="{{ link.url }}" target="_blank">{{ link.titulo
}}</a>
                            </h2>
                            <div class="sub header">{{ link.url }}</div>
                        </div>
                    </div>
                    <div class="ui form segment">
                        <h3 class="ui header">Salvar este link</h3>
                        <div class="field">
                            <div class="ui left icon action input large">
                                <i class="tags icon"></i>
                                <input placeholder="Adicione tags! (separadas por vÃ-
rgula)" type="text">
                                <div class="ui submit large yellow button">
                                    <i class="favorite icon"></i>Favoritar!</div>
                            </div>
                        </div>
                    </div>
                    <h4 class="ui horizontal header divider">
                        <i class="favorite icon"></i>{{ link_bookmarks.count }}
Bookmarks
                    </h4>

                    {% for bookmark in link_bookmarks %}

                    <div class="ui tag blue labels">
                        <img class="ui avatar image middle aligned" src="{{ MEDIA_URL
}}{{ bookmark.usuario.avatar_uploaded }}">

                        {% for tag in bookmark.tags.all %}

                        <a href="/{{ bookmark.usuario.user.username }}/{{ tag.tag_slug
}}" class="ui label">{{ tag }}</a>

                        {% endfor %}

                    </div>
                    <div class="ui divider"></div>

                    {% endfor %}

                </div>
```

Fonte: Autoria própria.

Quadro 21 - Refatoração do *template* “usuário”.

```
                <img class="ui tiny circular image centered" src="{{ MEDIA_URL }}{{
usuario.avatar_uploaded }}">
                <h2 class="ui header centered">
                    <div class="content">
                        {{ usuario }}
                        <div class="sub header">@{{ usuario }}</div>
                    </div>
                </h2>
                <p>{{ usuario.bio }}</p>
                <div class="ui tag blue labels">
                    {% for tag in tags_usuario.all %}

                    <a href="/{{ usuario.user.username }}/{{ tag.tag_slug }}" class="ui
label">{{ tag }}</a>
                    {% endfor %}

                </div>
                <h2 class="ui header">Bookmarks de @{{ usuario }}</h2>

                {% for bookmark in u_bookmarks %}

                <div class="ui raised segment">
                    <div class="ui top attached label">{{ bookmark.adicionado_em
}}</div>
                    <div class="ui header">
                        <div class="ui vertical basic buttons icon">
                            <a href="/link/{{ bookmark.link.id }}" class="ui button">
                                <i class="icon favorite large yellow"></i>
                                <p>
                                    <strong>{{ bookmark.saves_count }}</strong>
                                </p>
                            </a>
                        </div>
                        <div class="content">
                            <h3>
                                <a href="{{ bookmark.link.url }}" target="_blank">{{
bookmark.link.titulo }}</a>
                            </h3>
                            <div class="sub header">{{ bookmark.link.url }}</div>
                        </div>
                    </div>
                    <p>{{ bookmark.descricao }}</p>
                    <div class="ui divider">
                    </div>
                    <div class="ui tag blue labels">

                        {% for tag in bookmark.tags.all %}

                        <a href="/{{ bookmark.usuario.user.username }}/{{ tag.tag_slug
}}" class="ui label">{{ tag }}</a>
                        {% endfor %}
                    </div>
                </div>
                {% endfor %}
```

Fonte: Autoria própria.

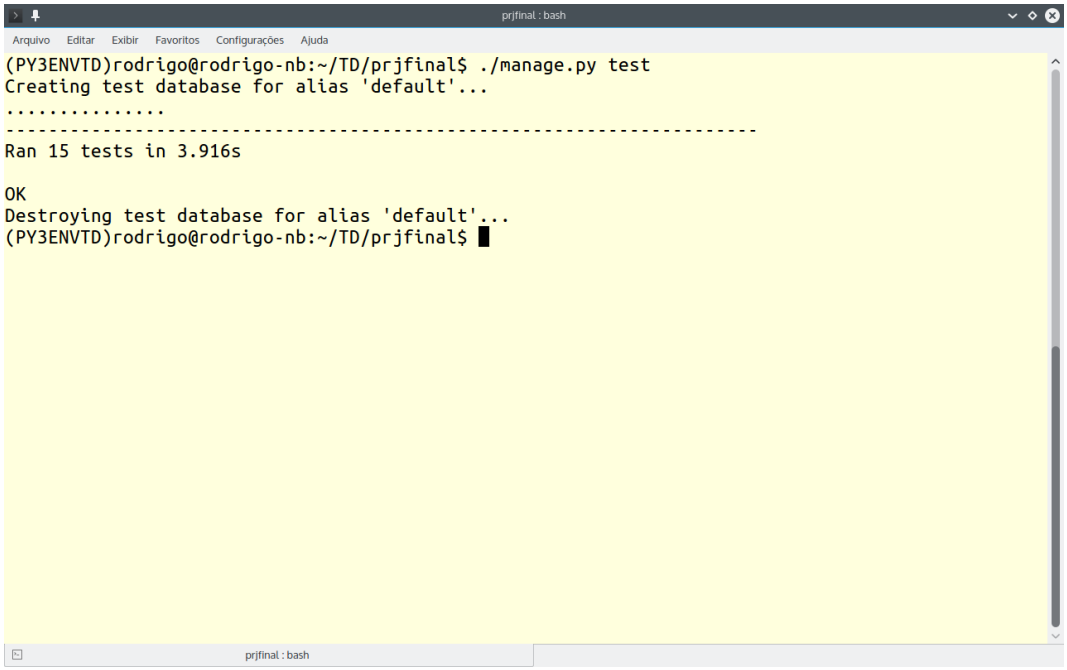
Quadro 22 - Refatoração do *template* “usuário-tag”.

```
                    <h2 class="ui header">
                    <img class="ui circular image" src="{{ MEDIA_URL }}{{
usuario.avatar_uploaded }}">
                    <div class="content">
                        <a href="/{{ usuario.user.username }}">{{ usuario }}</a>
                        <div class="sub header">
                            <a href="/{{ usuario.user.username }}">@{{
usuario.user.username }}</a>/
                            <a href="/{{ usuario.user.username }}/{{ tag.tag_slug
}}"></a>{{ tag }}</div>
                        </div>
                </h2>
                <h3 class="ui header">Bookmarks de @{{ usuario.user.username }} com a
tag
                    <a href="/tag/{{ tag }}" class="ui tag label blue">{{ tag }}</a>
                </h3>
                {% for bookmark in u_bookmarks_tag %}

                <div class="ui raised segment">
                    <div class="ui top attached label">{{ bookmark.adicionado_em
}}</div>
                    <div class="ui header">
                        <div class="ui vertical basic buttons icon">
                            <a href="/link/{{ bookmark.link.id }}" class="ui button">
                                <i class="icon favorite large yellow"></i>
                                <p>
                                    <strong>{{ bookmark.saves_count }}</strong>
                                </p>
                            </a>
                        </div>
                        <div class="content">
                            <h3>
                                <a href="{{ bookmark.link.url }}" target="_blank">{{
bookmark.link.titulo }}</a>
                            </h3>
                            <div class="sub header">{{ bookmark.link.url }}</div>
                        </div>
                    </div>
                    <p>{{ bookmark.descricao }}</p>
                    <div class="ui divider">
                    </div>
                    <div class="ui tag blue labels">

                        {% for tag in bookmark.tags.all %}

                        <a href="/{{ bookmark.usuario.user.username }}/{{ tag.tag_slug
}}" class="ui label">{{ tag }}</a>

                        {% endfor %}

                    </div>
                </div>
                {% endfor %}
```

Fonte: Autoria própria.

Após a alteração nos *templates*, os testes foram rodados novamente e todos foram validados, conforme a figura 40:

Figura 40 - Todos os testes passando.



Fonte: Autoria própria.

4.3 INSERÇÃO DE DADOS NA INTERFACE DE ADMINISTRAÇÃO

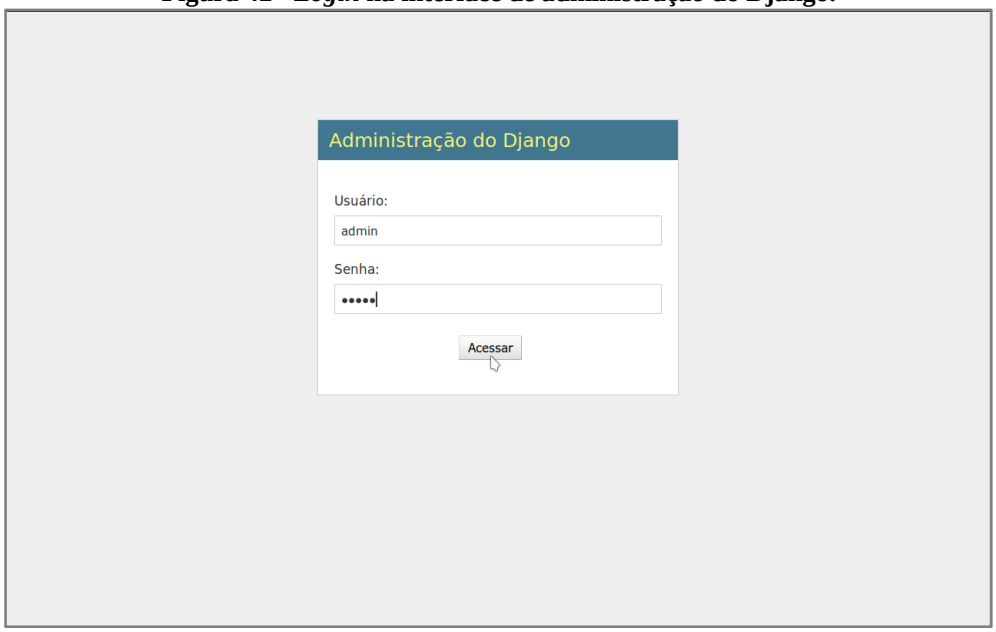
O Django oferece uma interface de administração com recursos de autenticação e gerenciamento de usuários. Além disso, é possível gerenciar os dados das aplicações adequadamente ativadas. Desta forma, a interface de administração do Django oferece acesso a operações CRUD nos modelos definidos nas apps do projeto. Para inicializar a interface de administração do Django é preciso configurar um *superuser* (usuário com privilégios administrativos) com o comando “createsuperuser”. (figura 41).

Figura 41 - Criação de super usuário.

Fonte: Autoria própria.

O usuário criado anteriormente pode ter acesso à interface de administração do Django (figura 42):

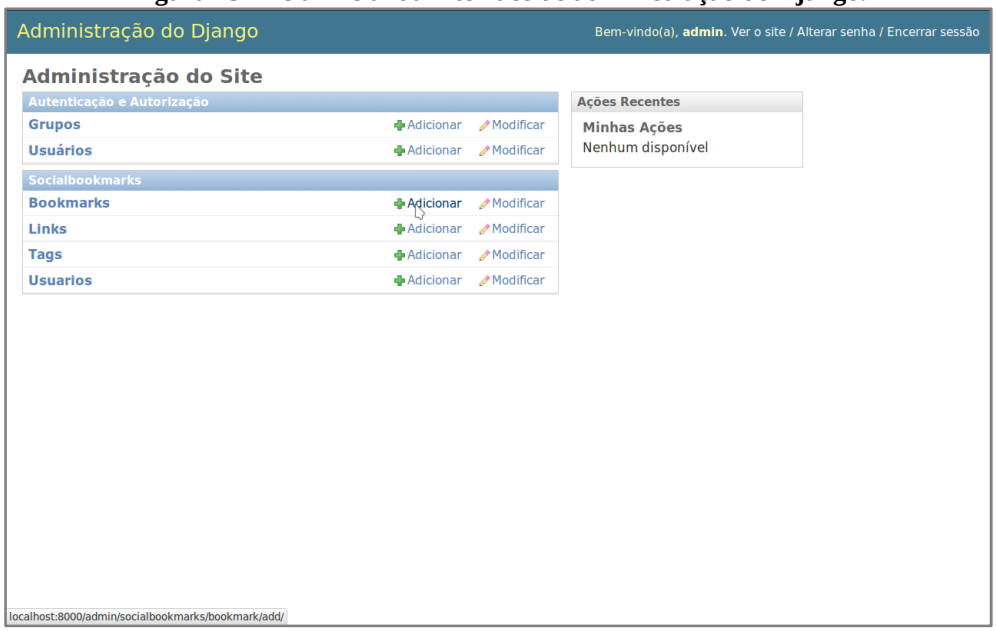
Figura 42 - *Login* na interface de administração do Django.



Fonte: Autoria própria.

Interface de administração do Django com as aplicações ativadas e acesso a operações CRUD aos respectivos modelos (figura 43):

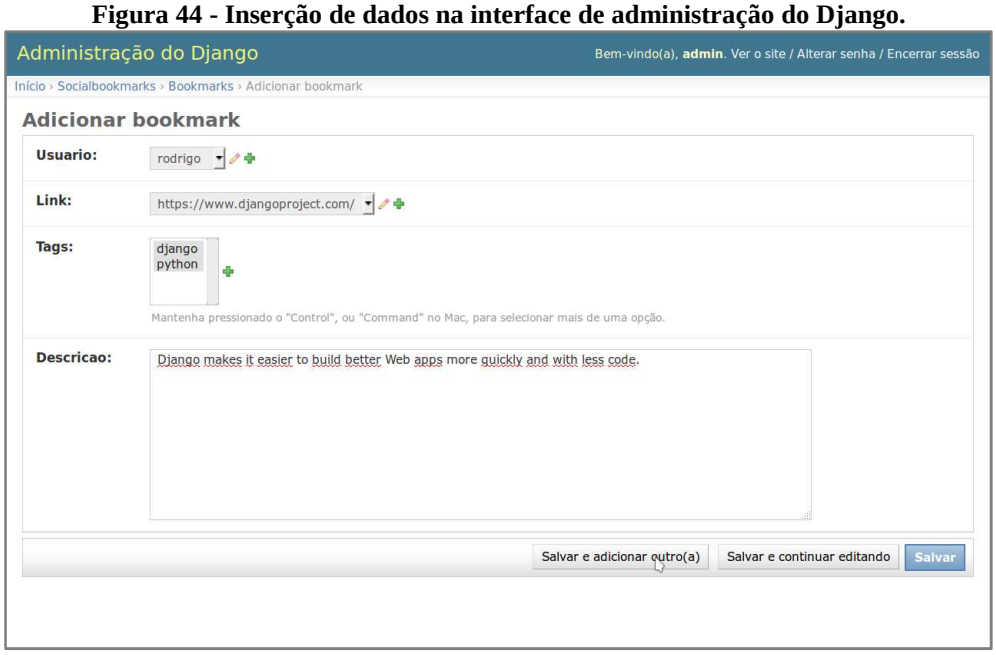
Figura 43 - Tela inicial da interface de administração do Django.



Fonte: Autoria própria.

Adição de alguns dados por meio da interface de administração do Django (figura 44):

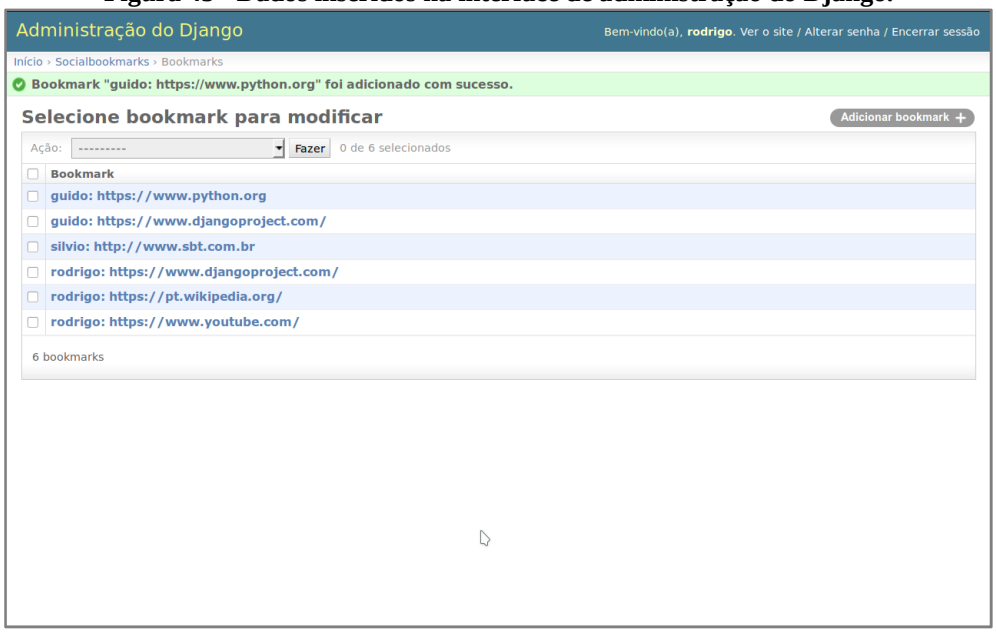
Figura 44 - Inserção de dados na interface de administração do Django.



Fonte: Autoria própria.

Dados inseridos (figura 45):

Figura 45 - Dados inseridos na interface de administração do Django.



Fonte: Autoria própria.

4.4 VERIFICAÇÃO DAS FUNCIONALIDADES DO SISTEMA

Nesta etapa é realizada a verificação dos cartões de histórias de usuário, confrontando-os com o sistema em funcionamento, a fim de verificar se as funcionalidades do sistema desenvolvido estão de acordo com as especificações propostas.

O primeiro cartão (figura 46) pode ser atendido em diversas partes da interação do usuário com o sistema. Estas informações são oferecidas em grande parte dos templates, por meio de uma barra lateral, e nas páginas dos usuários.

Figura 46 - Cartão de história de usuário.

Como um usuário, quero ter um perfil de usuário com foto, para ser reconhecido.

Fonte: Autoria própria.

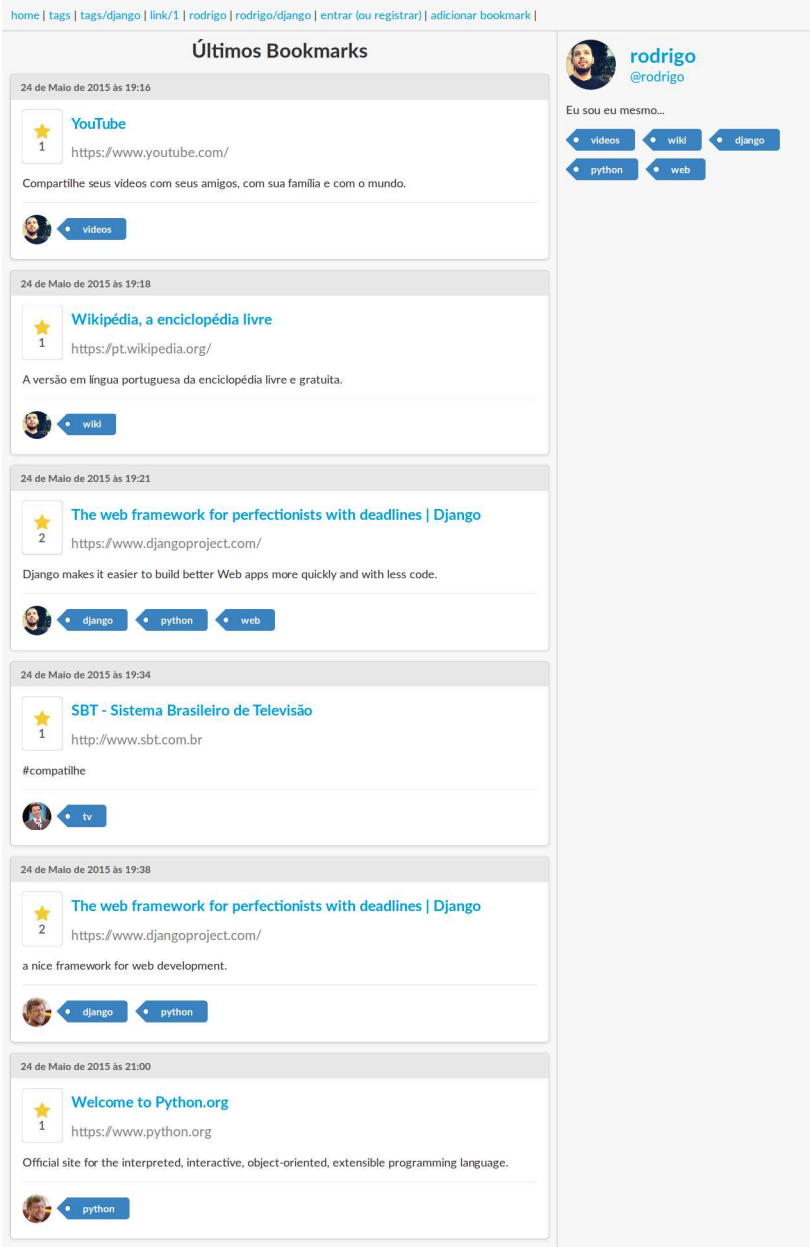
A listagem dos *bookmarks* oferece acesso aos links, aos usuários que os salvaram e as *tags* que utilizaram (figura 48), conforme definido com a história de usuário (figura 47):

Figura 47 - Cartão de história de usuário.

Como um usuário, quero visualizar os últimos bookmarks da comunidade, para descobrir novos links, usuários e tags.

Fonte: Autoria própria.

Figura 48 - Funcionalidade desenvolvida de acordo com a história de usuário



Fonte: Autoria própria.

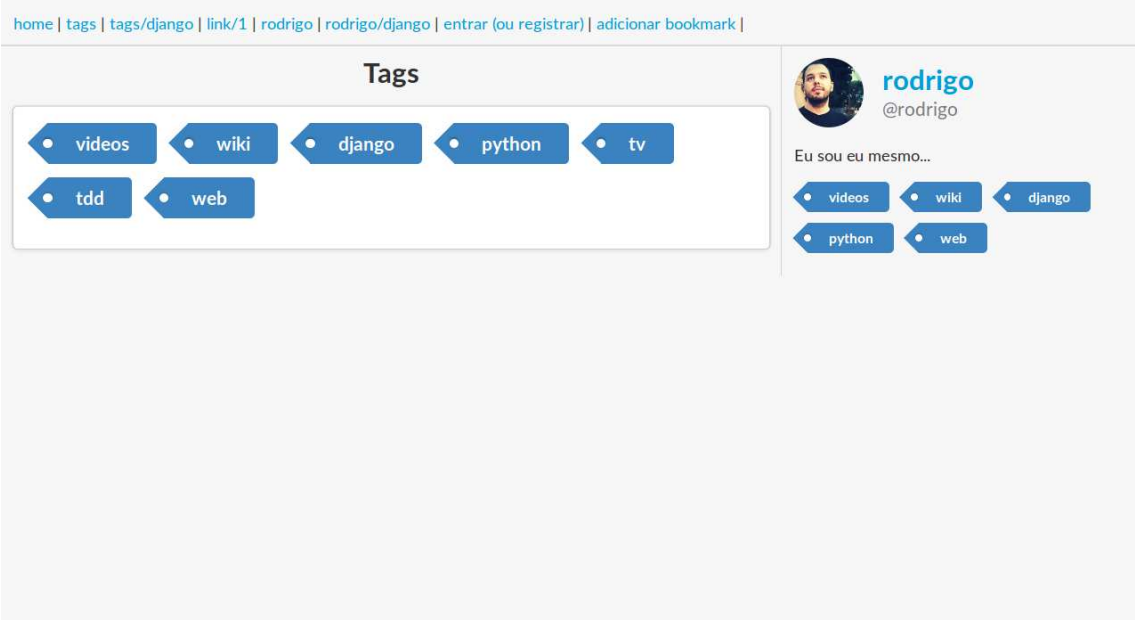
A página de *tags* do sistema (figura 50) oferece a funcionalidade especificada no cartão (figura 49), promovendo ao usuário o acesso à página de uma determinada *tag* que contém *bookmarks* salvos com esta *tag*:

Figura 49 - Cartão de história de usuário.

Como um usuário, quero visualizar todas as tags da comunidade, para descobrir novos links.

Fonte: Autoria própria.

Figura 50 - Funcionalidade desenvolvida de acordo com a história de usuário



Fonte: Autoria própria.

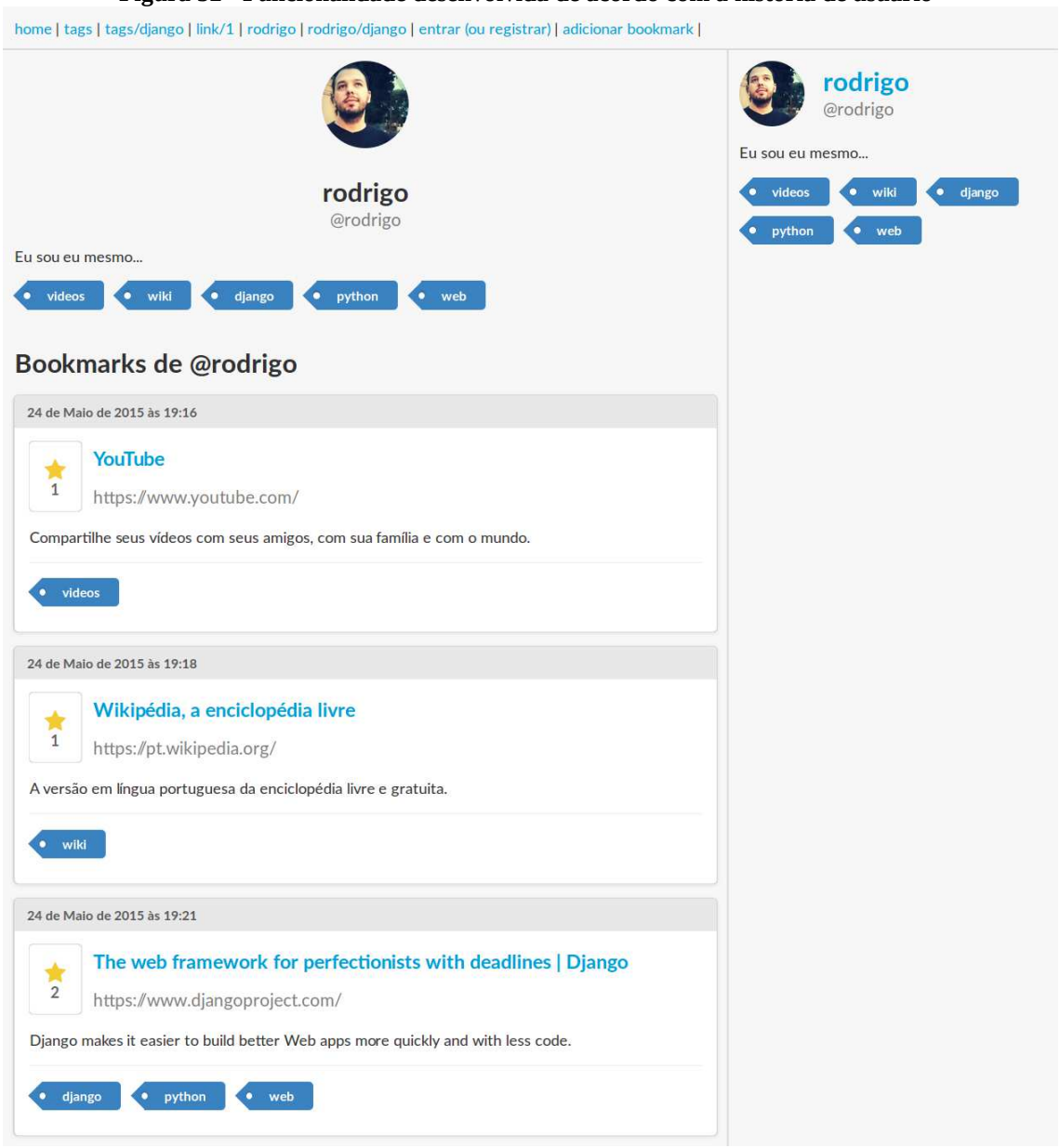
A página do usuário (figura 52) permite visualizar a coleção pessoal de links e *tags* do usuário, conforme definido com a história de usuário (figura 51):

Figura 51 - Cartão de história de usuário.

Como um usuário, quero visualizar os bookmarks que salvei e tags que utilizei.

Fonte: Autoria própria.

Figura 52 - Funcionalidade desenvolvida de acordo com a história de usuário



Fonte: Autoria própria.

O sistema possui a página (figura 55) que permite ao usuário armazenar um link favorito para acesso posterior e adicionar *tags* para facilitar sua recuperação, conforme definido com as histórias de usuário (figura 53 e 54):

Figura 53 - Cartão de história de usuário.

Como um usuário, quero guardar meus links favoritos, para acessá-los posteriormente.

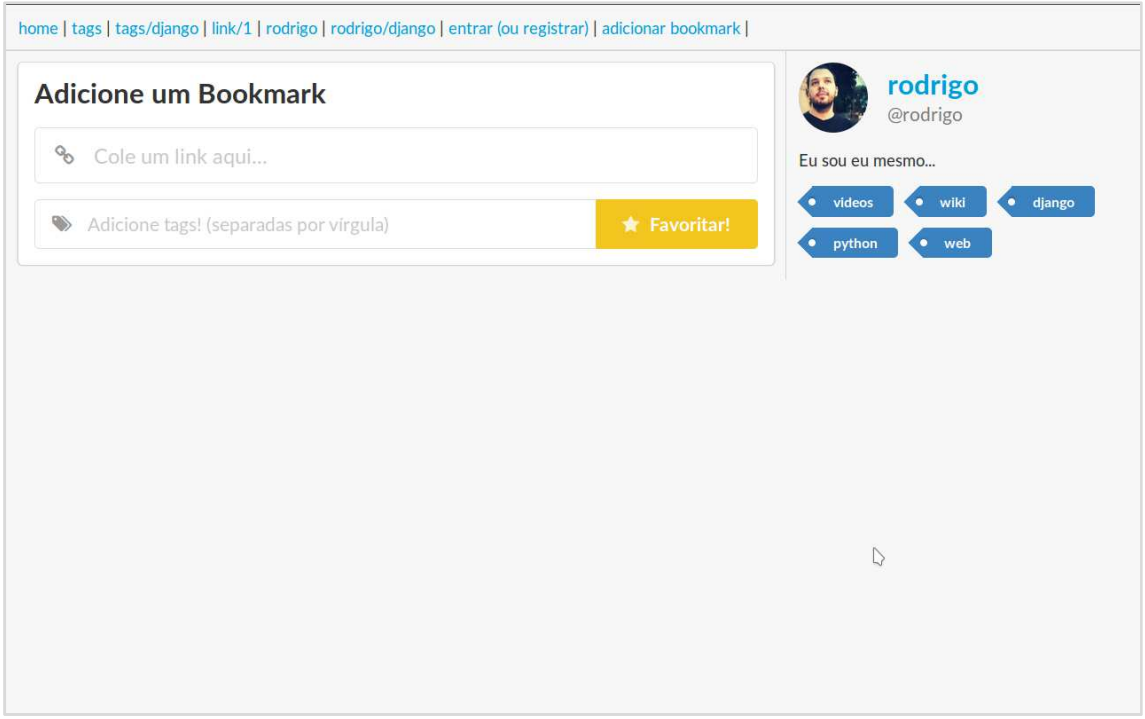
Fonte: Autoria própria.

Figura 54 - Cartão de história de usuário.

Como um usuário, adicionar tags aos meus bookmarks, para facilitar a identificação e pesquisa.

Fonte: Autoria própria.

Figura 55 - Funcionalidade desenvolvida de acordo com a história de usuário



Fonte: Autoria própria.

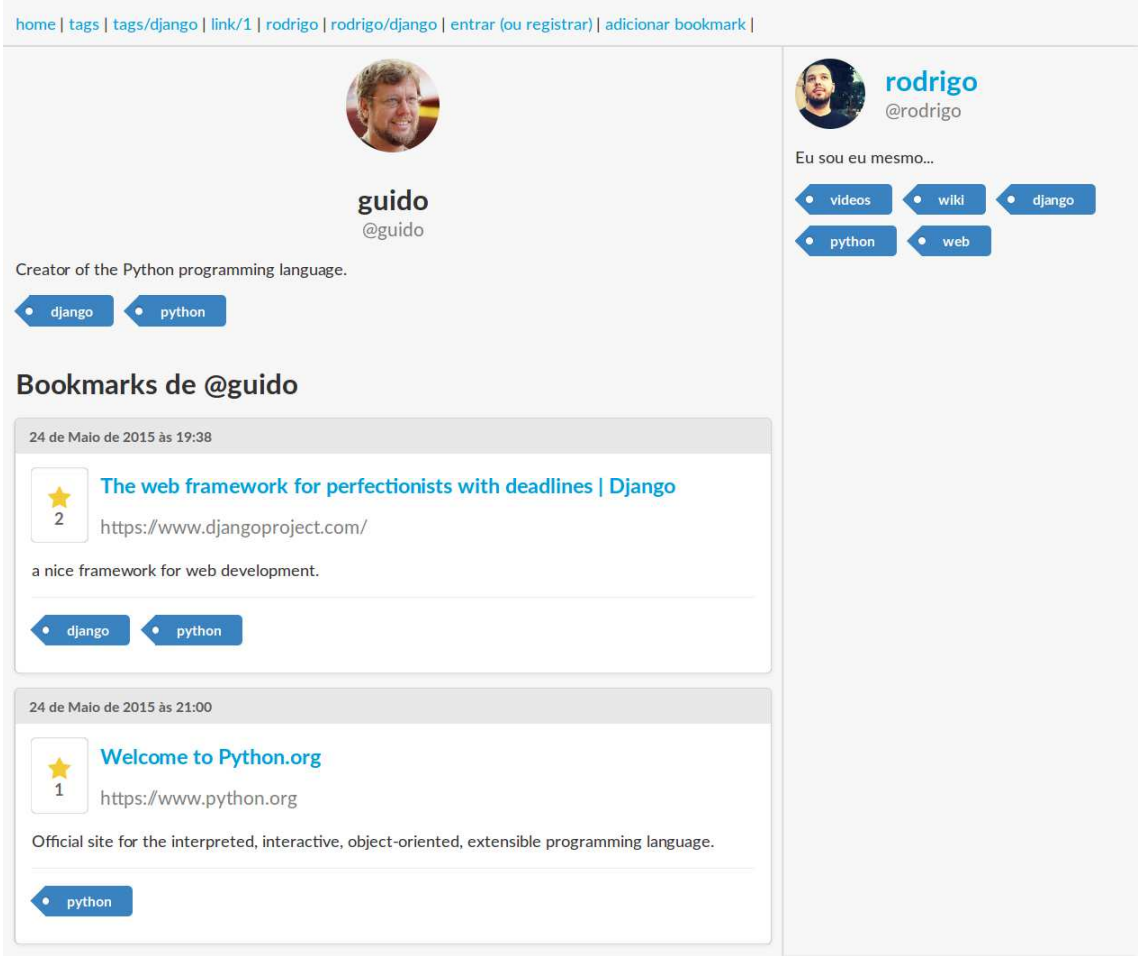
Do mesmo modo verificado na página pessoal do usuário, é possível acessar *bookmarks* de qualquer outro usuário e visualizar suas *tags* (figura 57), conforme definido com a história de usuário (figura 56):

Figura 56 - Cartão de história de usuário.

Como um usuário, quero visualizar os bookmarks e tags de outros usuários.

Fonte: Autoria própria.

Figura 57 - Funcionalidade desenvolvida de acordo com a história de usuário



Fonte: Autoria própria.

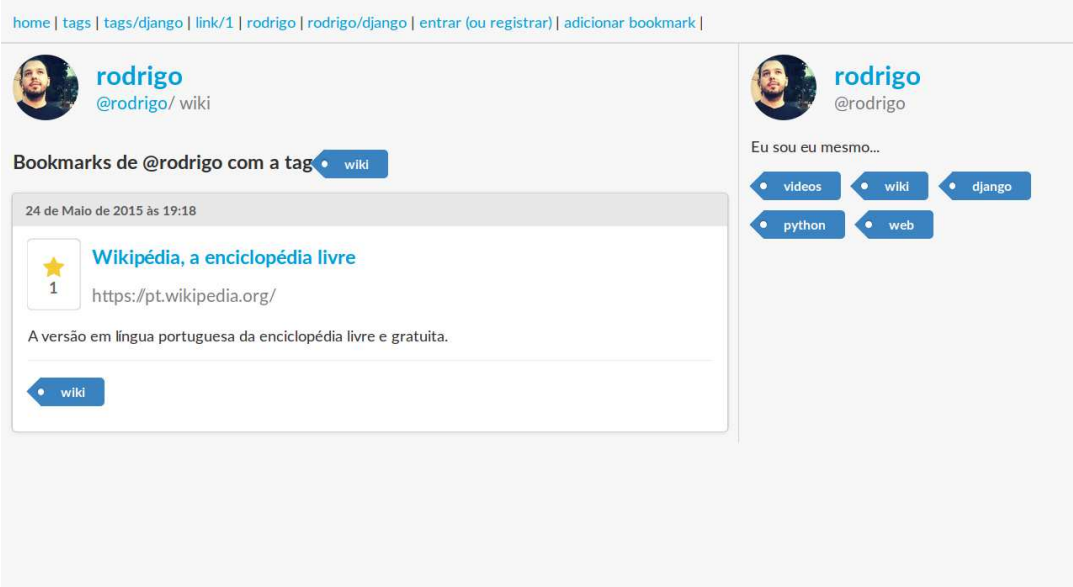
A história de usuário definida (figura 58) pode ser verificada filtrando os *bookmarks* por *tags* em dois templates distintos. O primeiro (figura 59) filtra os *bookmarks* de um determinado usuário; o segundo (figura 60) filtra os *bookmarks* de toda a comunidade.

Figura 58 - Cartão de história de usuário.

Como um usuário, quero filtrar os bookmarks por tags, para encontrá-los com mais facilidade.

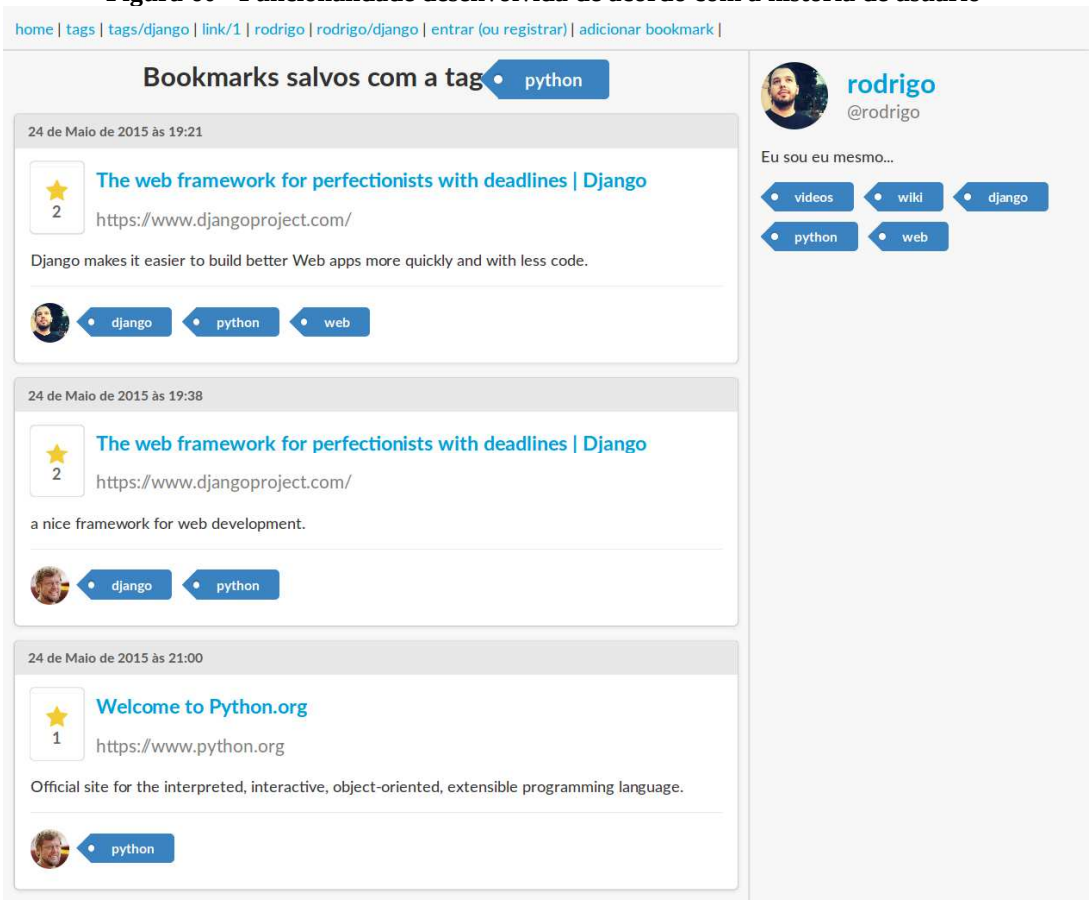
Fonte: Autoria própria.

Figura 59 - Funcionalidade desenvolvida de acordo com a história de usuário



Fonte: Autoria própria.

Figura 60 - Funcionalidade desenvolvida de acordo com a história de usuário



Fonte: Autoria própria.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

Por meio deste trabalho foi possível iniciar o desenvolvimento de uma plataforma de *bookmarking* social utilizando o framework Django em combinação com a técnica do desenvolvimento guiado por testes.

Os estudos realizados para a melhor compreensão do *bookmarking* social e a folksonomia contribuíram para a delimitação do funcionamento do software, contribuindo para a determinação das histórias de usuário que foram a base para a condução do desenvolvimento da plataforma.

A prototipagem de interface de usuário permitiu antever possíveis incoerências, favorecendo criação de um produto com maior possibilidade de acerto diante das funcionalidades requisitadas nas histórias de usuário.

No desenvolvimento com a linguagem Python e o *framework* Django, foi possível observar o potencial destas tecnologias no desenvolvimento de aplicações Web. O Django é um *framework* repleto de recursos que permite o desenvolvimento de aplicações Web com rapidez e confiabilidade, assim como a própria linguagem Python.

O módulo de testes do *framework* possibilitou a realização do desenvolvimento guiado por testes (TDD). A partir de então, verificou-se vantagens do uso de testes antes mesmo de se escrever o programa. Embora haja um trabalho adicional com os testes, escrevê-los antes, elimina consideravelmente o trabalho desnecessário ou prejudicial, garantindo melhor qualidade no código do sistema.

Além disso, o TDD tende a levar a um bom design do sistema, pois os testes unitários garantem que cada parte do software funcione de modo adequado e o mais independentemente possível, possibilitando futuras manutenções e melhorias.

O uso de TDD, além de colaborar na codificação, ajuda a compreender melhor o sistema e seus comportamentos, pois escrever ou pensar nos testes contribui para o entendimento do comportamento esperado do sistema.

5.2 CONTINUAÇÃO DO TRABALHO

A continuação deste trabalho sugere:

a) aprimoramento da plataforma desenvolvida;

b) publicação on-line do *website*, para que usuários utilizem a ferramenta e ofereçam feedback ao trabalho;

c) elaboração de um modelo de negócios para a plataforma desenvolvida;

d) um estudo mais aprofundado dos assuntos expostos neste trabalho;

e) expansão do trabalho para assuntos correlatos.

6 BIBLIOGRAFIA

ALVIM, L. Impossível não estar no Facebook: o nascimento das bibliotecas portuguesas na rede social. **Cadernos BAD**, 2011. Disponivel em: <http://www.bad.pt/publicacoes/index.php/cadernos/article/download/737/736>. Acesso em: 21 Jan. 2015.

ANDRADE, I. A. et al. Inteligência coletiva e ferramentas web 2.0: a busca da gestão da informação e do conhecimento em organizações. **Perspectivas em Gestão**, João Pessoa, n. Número Especial, p. 27-43, Out. 2011.

AQUINO, M. C. **Um resgate histórico do hipertexto:** O desvio da escrita hipertextual provocado pelo advento da Web e o retorno aos preceitos iniciais através de novos suportes, Julho 2006. Disponivel em: <http://www.moodle.ufba.br/file.php/10203/hipertextualidade/historico_hipertexto_Aquino.pdf>. Acesso em: 23 Outubro 2014.

AQUINO, M. C. Hipertexto 2.0, folksonomia e memória coletiva: um estudo das tags na organização da web. **Revista E-Compós**, Ago. 2007. Disponivel em: <http://www.compos.org.br>. Acesso em: 23 Jan. 2015.

BAPTISTA, A. A.; CATARINO, M. E. Folksonomia: um novo conceito para a organização dos recursos digitais na Web. **Revista Data Grama Zero**, Jun. 2007. Disponivel em: <http://www.dgz.org/jun07/Art_04.htm>. Acesso em: 19 Set. 2014.

BAREFOOT, D.; SZABO, J. **Manual de marketing em mídias sociais.** São Paulo: Novatec, 2010.

BRANDÃO, J. M. N. Apêndice 11: métodos da Query Set. **Aprendendo Django**, 2009. Disponivel em: <http://www.aprendendodjango.com/apendice-11-metodos-da-queryset>. Acesso em: 12 Jan. 2015.

BRANDT, M. B. **Etiquetagem e Folksonomia: uma análise sob a óptica dos processos de organização e recuperação**. Universidade de Brasília. Brasília, p. 142. 2009.

BUSH, V. As we may think. **The Atlantic Monthly**, Julho 1945. Disponivel em: <http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881>. Acesso em: 10 Novembro 2014.

CASTELLS, M. **A galáxia da internet:** reflexões sobre a internet, os negócios e a sociedade. Rio de Janeiro: Ed. Jorge Zahar, 2003.

CAVAZZA, F. Social media landscape 2013. **Fred cavazza**, 2013. Disponivel em: <http://www.fredcavazza.net/2013/04/17/social-media-landscape-2013>. Acesso em: 15 Out. 2014.

CERN. World Wide Web. **Cern**, 28 Out. 2014. Disponivel em: <http://info.cern.ch/hypertext/www/theproject.html>. Acesso em: 28 Out. 2014.

CIÊNCIA 2.0. Web x Web 2.0. **Ciência 2.0**, 15 Out. 2014. Disponivel em: <http://cienciadoispontozero.wordpress.com>. Acesso em: 15 Out. 2014.

COHN, M. **User stories applied:** for agile software development. Indiana: Addison-Wesley, 2009. 268 p.

COUTINHO, C. P.; JUNIOR, J. B. B. Blog e wiki: os futuros professores e as ferramentas da Web 2.0. **Simpósio Internacional de Informática Educativa no Porto - Portugal**, Nov. 2007. Disponivel em: <http://hdl.handle.net/1822/7358>. Acesso em: 23 Jul. 2014.

DESY. Hotlist do Mosaic. **Desy**, 27 Out. 2014. Disponivel em: <http://www.desy.de/web/mosaic/help-on-hotlist-view-old.html>. Acesso em: 27 Out. 2014.

DJANGO. Django documentation: everything you need to know about Django. **Django Project**, Mai. 2014. Disponivel em: <http://docs.djangoproject.com/en/1.8>. Acesso em: 26 Jan. 2015.

FARKAS, M. G. **Social softwares in libraries:** building collaboration, communication and community online. New Jersey: Information Today Inc., 2007.

FERREIRA, F. A história da internet e a popularização do vídeo. **Getempo**, p. 46-56, mar./abr. 2014. Disponivel em: <http://www.getempo.org>. Acesso em: 2 Out. 2014.

FILHO, D. D. L. O fio de Ariadne e a arquitetura da informação na WWW. **DataGramaZero - Revista de Ciência da Informação**, Dezembro 2003. Disponivel em: <http://www.dgz.org.br/dez03/Art_02.htm>. Acesso em: 14 Novembro 2014.

FREEMAN, S.; PRYCE, N. **Desenvolvimento de software orientado a objetos, guiado por testes**. 1°. ed. São Paulo: Alta Books, 2012. 385 p.

GOOGLE. Chrome Bookmarks. **Chrome Developer**, 2015. Disponivel em: <https://developer.chrome.com/extensions/bookmarks>.

HAENLEIN, M.; KAPLAN, A. M. User of the world, unite! The challenges and opportunities of social media. **Business Horizons**, Paris, n. 53°, p. 59-68, 2010.

HAMILL, P. **Unit test frameworks:** tools for high-quality software development. 1°. ed. United States of America: O'Reilly, 2009. 216 p.

HOLOVATY, A.; MOSS, J. K. **The definitive guide to Django:** web development done right. Nova York: Apress, 2009. 536 p.

HUNT, A.; THOMAS, D. **O programador pragmático:** de aprendiz a mestre. E-book: Bookman, 2010.

INTERNET INNOVATION. Mídias Sociais: Conceito e definição. **Internet Innovation**, 16 Janeiro 2013. Disponivel em: <http://www.internetinnovation.com.br/blog/glossario/midias-sociais-conceito-e-definicao/>. Acesso em: 2014 Dezembro 12.

LASAR, M. Before Netscape: the forgotten Web browsers of the early 1190s Before Netscape: the forgotten Web browsers of the early 1990s. **Arstechnica**, 11 Out. 2011. Disponivel em: <http://arstechnica.com>. Acesso em: 9 Jan. 2015.

LEFFINQWELL, D. **Agile software requirements:** Lean requirements practice for teams, programs, and the enterprise. 1°. ed. United States of America: Addison-Wesley, 2010. 518 p.

LOMAS, C. P. Seven things you should know about social bookmarking. **Educause**, Mai. 2005. Disponivel em: <http://www.educause.edu/library/resources/7-things-you-should-know-about-social-bookmarking>. Acesso em: 3 Dez. 2014.

LUTZ, M. **Programming Python**. 4°. ed. United States of America: O'Reilly, 2010.

MARTIN, R. C. The Clean Code Blog. **The cycles of TDD**, 17 Dez. 2014. Disponivel em: <http://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html>. Acesso em: 13 Fev. 2015.

MILANI, F.; PRIKLADNICKI, R.; WILLI, R. **Métodos ágeis para desenvolvimento de software**. Porto Alegre: Bookman, 2014.

MOUNIER, P. **Os donos da rede:** as tramas políticas da internet. São Paulo: Edições Loyola, 2006.

MOZILLA. Create bookmarks to save your favorite webpages. **Mozilla Support**, 2015. Disponivel em: <https://support.mozilla.org/en-US/kb/create-bookmarks-save-your-favorite-webpages>.

NUNES, S. **World Wide Web**, 2011. Disponivel em: <http://web.fe.up.pt/~ssn/2011/cdi/04-web.pdf>.

O'REILLY, T. What is Web 2.0: design patterns and business models for the next generation of software. **O'Reilly**, Set. 2005. Disponivel em: <http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html>. Acesso em: 19 Nov. 2014.

PASCOAL, R. **Colaboração e cognição na World Wide Web (Dissertação de Mestrado em Tecnologias da Inteligência e Design Digital) - Pontifícia Universidade Católica de São Paulo**. São Paulo: [s.n.], 2008.

PERCIVAL, H. J. W. **Test-driven development with Python - obey the testing goat:** using Django, Selenium, and Javascript. United States of America: O'Reilly, 2014. 449 p.

PEZZÈ, M.; YOUNG, M. **Teste e análise de software:** processos, princípios, técnicas. São Paulo: Bookman, 2008. 512 p.

PIMENTEL, V. Los lenguajes de desarrollo web del futuro: Python. **Anexom**, 31 Jul. 2009. Disponivel em: <http://www.anexom.es/tecnologia/los-lenguajes-de-desarrollo-web-del-futuro-python>. Acesso em: 2 Dez. 2014.

PIRNAT, M. Web development with Python and Django. **Speaker Deck**, 2014. Disponivel em: <https://speakerdeck.com/mpirnat/web-development-with-python-and-django-2014>. Acesso em: 12 Out. 2014.

PRESSMAN, R. S. **Engenharia de Software:** Uma abordagem profissional. 7ª Edição. ed. São Paulo: McGraw-Hill, 2011.

PYTHON. Python 3.4.3 documentation. **Python**, 19 Mai. 2015. Disponivel em: <http://docs.python.org/3>. Acesso em: 28 Jan. 2015.

RAMALHO, L. Django ORM: o básico. **Turing**, 2009. Disponivel em: <http://turing.com.br/material/acpython/mod3/django/orm1.html>. Acesso em: 21 Dez. 2014.

RIGBY, R. **Vinte e oito mentes que mudaram o mundo:** os maiores desbravadores da gestão de todos os tempos. Rio de Janeiro: Elsevier, 2012. 209 p.

ROETTGERS, J. Happy birthday Delicious: the pioneer of tags turns ten. **Gigaom**, 18 Set. 2013. Disponivel em: <gigaom.com/2013/09/18/ten-years-delicious>. Acesso em: 11 Dez. 2014.

THE PYTHON SOFTWARE FOUNDATION. 26.3. Unittest - unit test frameworks. **Python Org**, 24 Mar. 2015. Disponivel em: <https://docs.python.org/3/library/unittest.html>. Acesso em: 12 Fev. 2015.

TYMCHUK, A. Bookmarks no Internet Explorer. **Alek Setjym Chuk**, 2014. Disponivel em: <http://aleksetjymchuk.narod.ru/245c.gif>. Acesso em: 15 Out. 2014.

W2VR. Diagrama de hipertexto de Ted Nelson. **W2VR**, 14 Out. 2014. Disponivel em: <http://www.w2vr.com/timeline/nelson.html>. Acesso em: 20 Dez. 2014.

WIKIPEDIA. Tela de entrada no Exec PC BBS. **Wikipedia**, 14 Out. 2014. Disponivel em: <http://en.wikipedia.org/wiki/file?exec_pc_bbs.png>. Acesso em: 14 Out. 2014.