BAOBÁXIA

NA ROTA DOS BAOBÁS

REDE MOCAMBOS

NÚCLEO DE PESQUISA E DESENVOLVIMENTO DIGITAL

Baobáxia

na Rota dos Baobás

Relatório

Primeiro semestre

Parceria

Fundação Banco do Brasil

CASA DE CULTURA TAINÃ

Relatório Parcial de Execução

Projeto "Baobáxia na Rota dos Baobás" - n. 12537

Período: 06/2013 - 12/2013

Campinas, SP
16 Dezembro 2013

Pela Casa de Cultura Tainã

Presidente - Antonio Carlos Santos Silva
CPF 72150742853

Pela FBB / Agencia BB

Nome:
CPF:

"Vamos fazer um mundo mais do nosso jeito..."

Zumbi dos Palmares

REDE MOCAMBOS
Núcleo de Pesquisa e Desenvolvimento Digital
Casa de Cultura Tainã

Resumo

O projeto "Baobáxia na Rota dos Baobás" trata da concepção, desenvolvimento e implementação de uma arquitetura distribuída, voltada para a integração de redes locais mesmo em localidades nas quais a conexão à internet seja instável, lenta ou intermitente. Parte-se da experiência acumulada pela Rede Mocambos, que trabalha com a integração de mais de duzentas comunidades em todas as regiões do país através da apropriação de tecnologias, identificando pontos críticos em que a precariedade do acesso à internet se torna um impeditivo para a efetiva comunicação entre essas comunidades. O projeto parte do pressuposto de que não basta usar as tecnologias de informação já existentes - é preciso moldar o próprio desenvolvimento dessas tecnologias, para que atendam às demandas da sociedade. Para isso, adota como princípio básico e metodologia de trabalho os fundamentos do software livre - tanto na gestão das equipes de trabalho quanto nas soluções tecnológicas que utilizará.

Sumário

Acrónimos

RM	**R**ede **M**ocambos
SP	**S**ervice **P**rovider
IdP	**Id**entity **P**rovider
API	**A**pplication **P**rogramming **I**nterface
RFC	**R**equest **F**or **C**omments
JSON	**J**ava**S**cript **O**bject **N**otation
P2P	**P**eer To **P**eer
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol
YAML	**Y**et **A**nother **M**arkup **L**anguage
XMPP	e**X**tensible **M**essaging and **P**resence **P**rotocol
SSO	**S**ingle **S**ign **O**n
VSAT	**V**ery **S**mall **A**perture **T**erminal
DRY	**D**on't **R**epeat **Y**ourself
MVC	**M**odel **V**iew **C**ontroller
ORM	**O**bject **R**elational **M**apper
NPDD	**N**úcleo de **P**esquisa e **D**esenvolvimento **D**igital
NFC	**N**úcleo de **F**ormação **C**ontinuada
NCP	**N**úcleo de **C**omunicação e **P**edagogia
GESAC	**G**overno **E**letrônico **S**erviço de **A**tendimento ao **C**idadão

Parte I

Julho/Agosto

Capítulo 1

Organização e inicio do projeto

O projeto inicia formalmente com as assinaturas, no dia 17 de junho, na agencia BB de Campinas. O NPDD responsável pela condução do projeto realiza algumas reuniões para definir o grupo de trabalho da Rede Mocambos que vai ser contratada e as metas e etapas do projeto.

1.1 Equipe

Antonio Carlos Santos Silva Coordenador e articulador de ações nacionais e internacionais

Elaine da Silva Tozzi Administração e produção.

Lourenço Ribeiro Articulador e técnico linux, realiza oficinas de apropriação tecnológica e instalação de telecentro.

Milson dos Santos Silva Articulador e técnico linux, realiza oficinas de apropriação tecnológica e instalação de telecentro.

Paulo Sérgio Barbosa Articulador e técnico linux, realiza oficinas de apropriação tecnológica e instalação de telecentro.

Priscila Vera dos Santos Articuladora e técnico linux, realiza oficinas de apropriação tecnológica e instalação de telecentro.

Vincenzo Tozzi Coordenador do desenvolvimento do Baobáxia e do Projeto.

1.2 Encontro de Culturas Tradicionais/GO

A Rede Mocambos participou do XIII Encontro de Culturas Tradicionais da Chapada dos Veadeiros, e pela ocasião organizamos uma reunião do projeto no Mercado Sul de Taguatinga, para definir a atuação local da equipe que vai realizar oficinas e ações locais de acordo com as proprias realidades, informando e relatando atravez da wiki.

FIGURA 1.1: Encontro quilombola na Chapada dos Veadeiros

1.3 Oficinas de introdução a instrumentalização tecnológica/MA

Por Milson Onileto, em 26 julho 2013[1]:

> Rede Mocambos maranhão/Centro Cultural Alagbedê, vem realizando atividades de formações continuada em parceria com a Escola de Capoeira

[1] http://culturadigital.br/casaferreirodedeus/2013/07/26/ola-mundo/

Angola Mandingueiros do Amanha. As oficinas focadas em apropriação tecnológica em software livre linux e ferramentas mocambolicas, trabalhando o social através do digital. entendendo que nossos territórios transcende o geográfico. as aulas acontecem todos os sábado na sede da escola na rua Portugal, a parti das 09:00 da manha. As atividades são divididas como aulas para os alunos do projeto e um horário especial para os pais e mães dos mesmos.

As formações são as primeiras de uma serie itinerante de formações pelos núcleos da rede no estado e parceiros.

FIGURA 1.2: Oficinas no Centro Cultural Alagbedê

Por Milson Onileto, em 25 agosto 2013[2]:

Nessa manha do dia 24.08.2013 aconteceu a segunda oficina de apropriação tecnológica,a oficina partiu da ideia que temos que brigar por um territorio tecnologico que vem nos sendo negado ou vem sendo usado de forma não-critica, pela falta de acesso a informações e instrumentos tecnológicos livre. A rede mocambos vem criando esse vieis entre as tradicionais e o mundo tecnológico moudado ao nosso modo criando um mundo cada vez mais do nosso jeito.

[2] http://culturadigital.br/casaferreirodedeus/2013/08/25/segunda-oficina-de-apropriacao-tecnologica/

FIGURA 1.3: Oficinas no Centro Cultural Alagbedê

1.4 Debian Day/ES

No 17 de Agosto, o Núcleo de Cidadania Digital (NCD), juntamente com o Grupo de Usuários Debian do Espírito Santo (Gud-ES), realizou, no Centro de Artes da Ufes, o Debian Day, que contou com a participação da Rede Mocambos falando sobre apropriação tecnológica. O Debian Day trata-se de uma comemoração aberta do aniversário do Projeto Debian, cuja data de lançamento oficial foi 16 agosto de 1993, visa a produção de um sistema operacional livre. Dentre das atividades, ocorridas uma foi a apresentação da rede mocambos falando sobre apropriação tecnológica das comunidades quilombolas e movimentos sociais, apos a apresentação rolou uma conversar com os participantes e organização do evento, surgindo possibilidades de parcerias com o Núcleo de Cidadania Digital (NCD), Grupo de Usuários Debian (Gud-ES), Tux-ES e Revista Espirito Livre.[3]

[3] http://elegbaraguine.wordpress.com/2013/08/24/debian-day-vitoria-es/, http://softwarelivre.org/gud-es, http://www.mocambos.net/weblog/2013/08/24/rede-mocamdos-no-debian-day-vitoria-es/.

Capítulo 2

NPDD

O Núcleo de Pesquisa e Desenvolvimento Digital, NPDD, envolve pessoas com conhecimento técnicos de varias comunidade e realidades da Rede. O NPDD é responsável pelo desenvolvimento e manutenção das ferramentas digitais da Rede, cuidando atualmente dos portais (www.mocambos.net, wiki.mocambos.net, mapa.mocambos.net, galeria.mocambos.net), das contas email (@mocambos.net e @mocambos.org) e de criar documentação de base sobre as ferramentas digitais.

2.1 Informações

2.1.1 Endereços e contatos

Para informações e contatos podem escrever ao seguinte email, nosso principal canal de comunicação:
suporte@mocambos.org

Existe também uma lista de discussão do NPDD hospedada no riseup.net:
mocambos-npdd@lists.riseup.net

Distrito Federal	**São Paulo**	**Sicilia/Itália**
Mercado Sul	Casa de Cultura Tainã	BOCS
QSB 12/13 Loja 7	Rua Inhambú, 645	Via Piersanti Mattarella, 8
Taguatinga/DF	Campinas/SP	Bagheria
	Telefone: (19) 32282993	

2.2 Consultorias

2.2.1 Dynamite

A versão 0.1 do projeto que foca na questão de metadados, conforme plano de trabalho aprovado, foi desenvolvida com a consultoria da Associação Cultural Dynamite (nota fiscal 0001000 - 09/2013).

2.2.2 Gunga

A versão 0.2 do projeto que foca na gestão de repositórios multiplos, conforme plano de trabalho aprovado, foi desenvolvida com a consultoria da Gunga (nota fiscal 0145 - 11/2013).

2.3 Metodologia

O projeto é conduzido pelo NPDD (Núcleo de Pesquisa e Desenvolvimento Digital da Rede Mocambos), em sinergia com os demais Núcleos (NCP de Comunicação e Pedagogia, NFC de Formação Continuada). À equipe fixa de desenvolvedores se soma a participação de colaboradores e especialistas em diferentes áreas. Sob a coordenação do NPDD, o desenvolvimento se da de forma distribuída, utilizando-se ferramentas colaborativas via internet como git, wiki e irc. A metodologia de desenvolvimento segue o modelo “Agile” que prevê o lançamento frequente de código funcionante para avaliação contínua por parte dos usuários finais, permitindo a correção e a melhoria ao longo do trabalho.

2.3.1 Wiki

A documentação do projeto é disponível no wiki da Rede Mocambos no endereço:
http://wiki.mocambos.net/NPDD/Baobáxia

2.3.2 Codigo

O codigo do projeto é disponível em licença GPLv3 no Github no endereço:
http://github.com/RedeMocambos/baobaxia

2.3.3 Necessidades/Issues

As necessidades do projeto são registradas no sistema de issues do github no endereço:
http://github.com/RedeMocambos/baobaxia/issues

Parte II

Setembro/Outubro

Capítulo 3

Estruturando as Rotas locais e os Metadados

O projeto continua com as articulações e oficinas locais no Rio Grande do Sul e no norte. A equipe de desenvolvimento, com o apoio de consultores da Dynamite, realiza um estudo e uma primeira implementação da gestão dos metadados do Baobáxia.

3.1 Articulando com os terreiros

No dia 11 de setembro de 2013, aconteceu uma reunião com representantes do Mansu Nangetu terreiro da Mãe Nangetu, onde a possibilidade de fazer uma imersão sobre a rede mocambos com as comunidades de terreiro de Belém do Pará foi fechada. A data pra a oficina de instrumentalização tecnológica e ferramentas mocambólicas com datas de 08 à 10 do mês de Outubro (data sujeita à alterações), tendo em vista uma utilização critica da internet que de acordo com dados de pesquisas nacionais relaciona a porcentagem de pouca utilização dessa ferramenta por parte das comunidades negras e tradicionais de terreiro.

Durante a conversa, foi colocado por Milson Onilètó a importância da apropriação desse território tecnológico, e dessa forma fortalecer as praticas tradicionais e das organizações politicas das comunidades.

Mãe Nangetu falou da articulação das comunidades tradicionais nas discussões nacionais sobre as políticas públicas, principalmente as culturais. Ela relatou que estão acontecendo constantemente as conferências em Belém e que é importante que a juventude de terreiro se aproprie destas discussões politicas e que uma organização maior é necessária, que é responsabilidade da juventude manter essa luta ao lado dos mais velhos, afinal, o conhecimento ancestral deve ser repassado para preservarmos nossa história, nossas raízes. Arthur Leandro falou um pouco de seu histórico dentro do movimento afro religioso e de sua própria história de vida (que renderia um belo livro) e se mostrou bastante interessado em compor essa articulação local junto à rede mocambos, e que a casa está disposta a mobilizar na data estabelecida as lideranças estratégicas para a juventude de terreiro e dessa forma fortalecer mais um núcleo rede mocambos em Belém do Pará.

3.2 Oficina Terreiro Mãe Nangetu/PA

A oficina de apropriação tecnológica aconteceu no dia 9 de outubro de forma natural, durante a oficina de vídeo, quando o grupo percebeu, que era mais viável se instrumentalizar sobre as questões básicas de software livre e informática. Dessa forma foi necessário readequá a programação uma vez que seria apresentado a oficina de vídeo. A oficina foi pensada em três momentos de atividade que eram introdução a história dos computadores e internet que é o momento para procurarmos causar uma reflexão crítica desses da história e criação desses meios, aproveitando para falar de Rede Mocambos seu ideal de luta e de como nos inspiramos na causa palmarina para discutir esse território digital excludente. Foi mostrado filmes curtos depois de apresentação de slides sobre o tema. Esse filmes falaram sobre a funcionalidade das principais peças na maquina. Dessa forma esse vídeo foi um complemento que de forma visual facilitou o entendimento do que é o computador.

O segundo momento foi pensado para contextualização desse entendimento tecnológico e a pratica propriamente dita. Partindo pela instalação do software, utilização dos principais comandos e instalação dos principais programas. Esse momento foi acontecendo a parti de provocações do grupo.

O terceiro momento foi pensado para falar sobre as ferramentas tecnológicas da Rede Mocambos, onde falamos da historia da wiki e sua forma de utilização. Passamos pelo

processo de criação de usuário e todos os passos para criação dos publicação de conteúdos. O mapa foi apresentado e foi descrito sua importância para as comunidades porem o tempo não foi o bastante para mostra sua utilização.

3.3 Oficina em Bujarú/PA

A atividade na comunidade Quilombola de São judas Tadeu foi linda. Chegamos Um hora da tarde quando o Aldo representante quilombola foi nos pegar na balsa e levar pra comunidade, a presidente da associação Kátia recebeu muito bem a equipe, relatando a situação da comunidade e que eles já esparavão a anos essa instalação. Foi notório a euforia da comunidade que logo se aproximou e quiz saber se seu sonho se realizaria naquele momento. O que deveria começar no outro dia pela manha, iniciou-se no mesmo passo da chegada. Começamos a instalação do telecentro e com as oficinas que iam acontecendo naturalmente como conversas, os jovens e crianças, mulheres e homens da comunidade todos interagindo com o processo. Logo cada um estava contando pela primeira vez um computador. depois da instalação e da troca de ideias sobre informática e apropriação tecnológica, começamos a falar da realidade das pessoas que viviam ali, e que muitos se quer ja tinha assistido filme em uma tela grande. começamos a pensar em uma mostra de cinema que aconteceu no espaço do telecentro, logo a sala esta cheia de crianças, jovens, mulheres, todo mundo juntos, vidrados nas aventuras de Kiriku, o filma acabou e observar os olhares pidões querendo mais das crianças foi o bastante pra em poucos cliques, começar o filme Quilombos, que conta a historia do Quilombo dos Palmares. depois do filme as atividades terminarão. Pela manhã começamos ouvindo os mais velhos da comunidade e visitando as áreas de plantações que rodeava toda a comunidade com uma bela forma de sustentabilidade, ouvimos como começarão as plantações e conhecemos um dos igarapé que cortava e dividia as comunidades. Após todo o rolê fomos para o telecentro onde terminamos a instalação e demos continuidade as oficinas dessa vez de instalação e noções básicas de Linux Ubuntu (comandos e principais programas, entendendo as principais diferenças entre o sistema operacional Windows e o Linux, observamos que ninguém si quer tinha ouvido falar do Linux antes. Após essa atividade fizemos a segunda mostra de cinema com curtas para criançada com temáticas ambientais. Isso durante a instalação demorada de todo o sistema do telecentro e configuração do Roteador wireless.

3.4 Pajelança Quilombolica Digital/PA

Ewé Ni Bo Bo Ni Ti Segun
Ewé Ni Bo Bo Ni Ti Orisá
Ewé Ni Bo Bo Ni Ti Segun[1]

Babá (Mãe Nalva)

A Rede Mocambos e o Coletivo Casa Preta na perspectiva de garantir para essas comunidades a sua cultura, os seus costumes, a resignificação e o protagonismo da própria história, convida amigas e amigos para participar da "Pajelança Quilombólica Digital", que objetiva fortalecer o conhecimento e empoderamento do saber livre a partir da visão africana. Ao longo da semana, foram realizadas rodas de conversa, oficinas de vídeo, apropriação tecnológica, tambores, herbário e um dia dedicado ao debate sobre ser Mulher preta na ativa com a juventude feminina da Casa Preta convidando as mulheres pretas que inspiram suas lutas. A Pajelança Quilombólica Digital foi realizada no período de 1 a 7 de setembro na sede do Coletivo Casa Preta que fica na rua Roso Danin, 780, entre 2a de Queluz e Juvenal Cordeiro, no bairro Canudos-Terra Firme, em Belém.

3.5 Encontros/RS

3.5.1 Ipadé da Juventude Quilombola e Indígena

O Ipadé (encontro, reunião) da Juventude Quilombola e Indígena, foi feito com o jeito afro-gaúcho. A atividade, que buscava conectar ancestralidade e tecnologias digitais, teve o desafio de se recriar pelo fato de não haver conexão local com a internet, com isso, as atividades tecnológicas e oficinas foram desenvolvidas independentemente da internet para sua realização. Nesta formação continuada abordamos, dentro das tecnologias, a apropriação tecnológica e oficinas de vídeo e edição. As atividades ocorreram de 11 a 13 de outubro, na ComPaz, em Triunfo (RS), congregando Memória, Resistência e Comunicação no fortalecimento da juventude através da conexão entre saberes ancestrais

[1] "Todas as folhas tem o poder de curar.
Todas as folhas dos orixás tem o poder de curar"

FIGURA 3.1: Pajelança Quilombolica Digital na Casa Preta, Belem

e novas tecnologias. A partir da produção de conteúdos feita do nosso jeito, o Ipadé foi um espaço de sensibilização em relação às tecnologias, no registro dos conhecimentos locais através de textos, vídeos e fotos.

Durante os três dias, houve intensa troca de conhecimentos e vivências. Os jovens experimentaram ser protagonistas na apropriação tecnológica livre da internet. Seguem detalhamento das atividades do Ipadé.

Inicialmente foi realizada a oficina de apropriação tecnológica, onde todos conheceram a máquina por dentro, desmistificando para muitos que a máquina pode estragar se for manipulada. Todos foram estimulados a tocar nas peças. Numa mistura de tecnologia com as nossas raízes, com aquilo que nos é ancestral, com o que vem antes da gente. Se pensamos no Baobá, uma árvore que vive mais de 2000 anos, nossos projetos têm que ir além de nós, pensando também em nossos filhos, netos, bisnetos e assim por diante. A Morada da Paz possui uma Bioconstrução (a Bio), iniciada em 2005 (e reformada ou expandida por meio de mutirões constantemente), no período do Fórum Social Mundial, quando a mesma recebeu os materiais da casa. Para os jovens que participaram, foi uma interessante referência de uma casa de barro construída recentemente e bem cuidada,

dado que em suas comunidades, esta era uma forma tradicional de moradia, que atualmente não existe mais. No Ipadé, trabalhamos as tecnologias mais modernas, mas sem perder a noção das próprias origens. Ao mesmo tempo que foi falado de programas de computadores, também o Baobá, a casa de barro, as ervas de chá e outros estavam em questão. O núcleo entende que essas coisas estão ligadas e, vendo as tecnologias como ferramentas, podemos fortalecer as nossas raízes e que a tecnologia não é o fim em si, não é um objetivo extremo ter um computador, e sim pensar como é possível usar esse computador. Também a oficina de construção de instrumentos musicais (o xequerê) fez essa conexão. Avaliação: Esse encontro serviu para conversarmos sobre o que é nosso mundo, o que pensamos e o que poderíamos fazer para mudar o nosso mundo e no que essas ferramentas tecnológicas podem nos ajudar, como produzir um vídeo, um sabonete ou alguma coisa localmente. O grande desafio é como juntar essas coisas, como fazer uma rádio comunitária, as atividades que podem estar na rádio e na comunidade, por exemplo. A ideia pode ser resumida em uma frase de Jorge Rasta, da Bahia: “devemos plugar os tambores nos computadores”.

Contamos com a presença de rodas de trocas ao som dos tambores e do berimbau do Mestre de capoeira Pretto, e de dois grandes Mestres da cultura popular brasileira e afro-brasileiras, os Mestre Paraqueda e Mestre Chico.

No Ipadé de Fala, Mestre Chico nos ensinou algumas palavras em yoruba: agô — licença, mojúbà - seja bem vindo, agbá- velho/a, aiye – terra, amaci - ervas maceradas na aguá, asé- força, babá – pai, borí- comida à cabeça. Êpô- azeite de dendê, Filá - gorro de cabeça, Fuxico – fofoca/segredo, Ilé – pombo, ile – terra, ilu– cidade, Jurema - bebida de caboclo, Ojô – chuva,, òjò – dançar, ojoz – dia, Eu – emi, t u – iwo, ele – oun, nós – auá, vós – euyin, meu/minha – mi, teu/tua – tiré, seu/dele/dela – tire, nosso – tiwa, deles/delas – tiwon, Mestre Chico é um dos pouco conhecedores da língua Yorubá.

3.5.2 Pajelança Quilombolica Digital/RS

De 21 a 23/10/2013 no Afro-Sul.

A pajelança teve inicio dia 21 de outubro, dia de chegança dos participantes. As atividades foram divididas nos Núcleos de Formação Continuada da Rede Mocambos, nas cidades de Porto Alegre, Guaíba e Triunfo. As atividades foram realizadas em um

FIGURA 3.2: Oficinas no Ipadé da Juventude Quilombola e Indígena

primeiro momento no Afrosul Odomode, sediado em Porto Alegre, com rodas de conversa oficinas de gravações de áudio e outros.

FIGURA 3.3: Pajelança Quilombolica Digital no AfroSul Odomode

FIGURA 3.4: Oficina Pajelança no AfroSul Odomode

Capítulo 4

Metadados

4.1 Descrição geral da pesquisa de Metadados e políticas de metadados

A pesquisa sobre metadados para o sistema Baobáxia compreendeu as definições diretamente relacionadas aos metadados das mídias e também a participação na modelagem geral dos objetos do sistema, assim como a definição de formatos para troca de dados. Por metadados entendemos toda a informação a respeito de si, seja para a mídia, seja para as Mucuas como para o sistema. Deste modo, num entendimento amplo de metadados, consideramos que qualquer dado descritivo a respeito do sistema como um todo deve ser entendido como um metadado, e logo deve ser armazenado em arquivos de definição.

Essa diretiva partiu de uma orientação em conjunto com o desenvolvimento do núcleo do software Baobáxia (https://github.com/RedeMocambos/baobaxia). Deste modo, tudo que envolve a descrição de elementos do sistema pode ser considerado um metadado. Os metadados, embora tenham como objetivo a estabilidade da informação, devem contemplar a possibilidade de alterações nas suas definições, bem como a expansão futura.

De início, definem-se dois tipos principais de metadado:

- metadado do sistema / políticas (arquivos de configuração, regras de funcionamento)

- metadado do acervo/das mídias

Não coube à pesquisa em metadados a definição das políticas, mas a orientação de como estas deveriam ser armazenadas e estruturadas. As políticas de sistema dizem respeito a definições gerais sobre a configuração do software e de critérios estabelecidos politicamente, isto é, atendendo às demandas da comunidade usuária e gestora do sistema Baobáxia. Entretanto, há uma escolha que entende a importância da distinção entre o sistema e suas configurações, estas que devem ser implementadas de acordo com as necessidades locais da Mucua ou do repositório (baobáxia / Rede Mocambos).

A partir da definição de políticas, ocorre a tomada de decisões pelo sistema, respeitando aos critérios estabelecidos. Essa definição possui normas técnicas próprias, mas é abrangente o bastante para permitir que novas políticas sejam incorporadas ao sistema conforme surjam novas necessidades.

4.2 Mídias e Mucuas

Os metadados relacionados a mídias e mucuas constituem o núcleo do sistema. Em tratando-se de um sistema de acervo distribuído, os nós (Mucuas) e arquivos contidos nesses nós (Mídias) são os principais elementos do sistema do ponto de vista dos conteúdos.

As mídias sempre estão associadas a mucuas originadoras (as quais originaram a publicação), e dividem-se por tipos:

- vídeos
- imagens (fotografias, desenhos etc)
- áudios (músicas, entrevistas, programas de rádio etc)
- arquivos (documentos, textos e demais arquivos)

Cada tipo de mídia possui especificidades do ponto de vista de seus metadados, ao que buscou-se definir o que há de comum entre todas. Os metadados foram definidos preliminarmente como o mínimo essencial possível, sendo que qualificadores adicionais devem

ser estendidos ou como tags ou como metadados específicos do tipo de arquivo. Assim, chegou-se aos seguintes campos, cuja definição encontra-se nos arquivos de políticas (ver adiante):

- data (AAAA-MM-DD)
- uuid (identificação única do arquivo)
- title (título, texto não único)
- comment (comentário, texto aberto)
- author (mocambola, associado a quem publicou o arquivo)
- type (tipos das mídias [vídeo|imagens|áudios|arquivos])
- format (formato do arquivo, definido em arquivo de políticas)
- origin (mucua, o servidor a partir do qual arquivo foi publicado)
- repository (referência ao repositório git annex a que está vinculado o arquivo)
- tags (etiquetas, múltiplas; somente descritoras ou também funcionais)

Além da mídia, as mucuas (nós do acervo, servidores locais) possuem também metadados específicos, bem como associados. Os metadados específicos dizem respeito a informações ligadas diretamente ao servidor; as associadas, ao conteúdo hospedado no servidor. Dessa forma, temos:

- note (texto geral sobre a mucua)
- description (nome da mucua)
- uuid (identificador único da mucua)

Quanto ao metadado associado, diz respeito a todas as etiquetas de conteúdos que estejam hospedados na mucua. Está previsto como um metadado descritivo sobre a mucua um relatório agrupando todas as etiquetas dos arquivos hospedados, o que permite aos mocambolas (usuári@s) terem uma ideia geral sobre as características do acervo de determinada Mucua.

4.3 Armazenamento do metadado: arquivos, padrões adotados e intercâmbio

Conforme já assinalado, metadados e políticas são armazenados em arquivos descritores. Para tal foi feita uma pesquisa a respeito dos formatos a serem adotados para armazenamento destes dados. Teriam de atender os seguintes requisitos:

A armazenamento em suporte aberto, não proprietário

B ótima capacidade de intercâmbio entre linguagens

C amplo desenvolvimento de bibliotecas de código em distintas linguagens de programação

D capacidade para armazenamento de objetos complexos

E bom desempenho

F tamanho diminuto

G adequação a plataformas RESTful e serviços de dados (web services)

Inicialmente, levantou-se três possibilidades, todas elas atendendo os itens A e B:

XML eXpansible Markup Language

YAML Yet Another Markup Languge

JSON JavaScript Object Notation

O primeiro formato, XML, tem a seu favor o fato de ser largamente adotado para intercâmbio de dados e para descrição de informações já há muitos anos (C). Conta no entanto com algumas desvantagens sobretudo do ponto de vista do desempenho (E), além de gerar uma não desprezível quantidade de informações extra especialmente ao lidar com o armazenamento de objetos complexos, o que tem impactos severos ao se pensar numa rede de acervos com conectividade baixa ou nula (ponto F). Além disso, conta com possíveis problemas no armazenamento de objetos complexos ao redundar numa estrutura de dados pesada (D).

O segundo formato, YAML, apresenta-se como um possível sucessor do XML, tendo se inspirado neste. Tem como resultado arquivos sintéticos e de tamanho diminuto (E e F). Possui ótima capacidade de armazenamento de objetos complexos (D). Apesar de contar com mais funcionalidades que o JSON - como a possibilidade de estabelecer relacionamentos funcionais lógicos internos, conta ainda com menor desenvolvimento e compatibilidade (C e G), ainda que seu futuro seja promissor.

O terceiro formato, JSON, é considerado uma forma de reduzir o overhead computacional do XML (E), sendo uma excelente alternativa para armazenamento de objetos complexos (D), sendo sintético e de tamanho diminuto (F). Conta com grande desenvolvimento em uma série de softwares (C) pode-se dizer que atualmente é o novo padrão para intercâmbio de dados, sendo base para numerosas tecnologias baseadas em plataformas RESTful (G)

Desse modo, foi escolhido o formato JSON como padrão para intercâmbio de dados, definição de políticas e metadados.

4.4 Politicas

4.4.1 Media

Definição dos metadados de mídia:

formats definem-se os tipos de formatos aceitos

priority tipos prioritários aceitos pelo sistema

metadata definição dos campos com tipo de dados aceito. Pode receber uma expressão regular (ex.: `/^\w_-$/`)

```
{
    "formats": [
        {
        "priority": [
            {
            "type": "ogg"
        },
        {
            "type": "mpeg"
        }
    ]
    }
    ],
    "metadata": [
```

```
    {
        "title": "string",
        "author": "string",
        "origin": "string",
        "date": "dd/mm/yyyy",
        "type": "string",
        "format": "string",
        "license": "string",
        "origin": "string",
        "repository": "string",
        "tags": [
        {
            "name": "string"
        }
        ]
    }
    ]
}
```

4.4.2 Mucua

```
{
    "uuid": "uuid",  // hash  git annex uuid  "b4debebe-b4e7-11e2-877b-736e379b6ff5"
    "description": "string", // string  git annex [description] "taina"
    "repositories": [ // array of strings with the name of the repositories
    {
        "name": "string"   // string name   "redemocambos"
    }
    ]
}
```

Parte III

Novembro/Dezembro

Capítulo 5

Infraestrutura e primeiras Mucuas

O NPDD foi inaugurado com a chegada das 4 mucuas que eram previstas no projeto (servidores comunitários). A estruturação das Rotas locais continuou com a instalação de alguns telecentros. O desenvolvimento do Baobáxia esta focado na gestão de repositorios de diferentes redes.

5.1 Mucuas

Foram adquiridos 4 computadores como primeiras mucuas do Baobáxia. Apos reunião em Brasilia, em ocasião da inauguração da nova sede do NPDD, se propus destinar as Mucuas para as seguinte comunidades:

dpadua Mercado Sul de Taguatinga/DF

akoni Comunidade Zumbi dos Palmares/MA

exu Terreiro Cultural Coco de Umbigada/PE

mestreborel AfroSul Odomode/RS

Essas vao somar a "dandara" a mucua da Casa de Cultura Tainã e "acotirene" atualmente hospedada, com o projeto Saravá, na Unicamp. Outras mucuas, baseadas em maquinas

recicladas, foram simbolicamente entregues durante o IV Encontro Nacional da Rede Mocambos.

5.2 Instalação do telecentro de Pitimandeua/PA

Dia 14 de Novembro de 2013, mais um telecentro quilombola foi instalado e conectado. A comunidade Quilombola Menino Jesus de pitimandeua, localizada no munícipio de pitimandeua, proximo a Castanhal-PA, recebeu os articuladores da rede mocambos Guinê Ribeiro e DJ RG, que realizaram o processo de instalação do telecentro da comunidade. Essa iniciativa e uma ação conjunta dos projetos Baobaxia e Núcleos de formação continuda da Rede mocambos.[1]

5.3 Telecentro das Ilhas-Abaitetuba-Rio Genipauba/PA

Dia 13 de novembro de 2013, a Rede mocambos instalou o telecentro da associação de artesãs das ilhas de Abaitetuba, rio Genipauba. http://galeria.mocambos.net/Instala-o-do-telecentro-das-Ilhas–Abaitetuba–Rio-Genipauba

5.4 Visita aos quilombos de Guajará Mirim e Itacoã Mirim/PA

Dia 12/11/2013

Visita nas comunidades com o objetivo de instalar/verificar os Telecentros, quais os materiais e maquinários entregues, disponibilidade de conexão, estrutura e espaço como salas, escola, associação, e a relação da comunidade com a tecnologia e comunicação[2].

Itacoã Comunidade com aproximadamente 150 famílias, que desenvolvem atividades de plantio, produção de farinha, entre outras que fazem parte do cotidiano da região. Conversa sobre rotinas, informes locais (ano eleitoral, dezembro, para designar representantes legais da comunidade, propostas e melhorias), realizada

[1] http://www.mocambos.net/pt-br/weblog/2013/11/20/Destaque/
[2] http://galeria.mocambos.net/Instala-o-dos-Telecentro-de-Guajar-e-Itaco--PA

FIGURA 5.1: Fotos das Mucuas no NPDD

na residência de seu Zeca (morador local), no Telecentro chegou apenas o data show, não tem conexão, mesas, computadores e espaço destinado à instalação.

Guajará Mirim Comunidade tem em torno de 120 famílias,realizam atividades de manejo, plantio e produção, mas acabam por se limitar no processo de desenvolvimento do próprio espaço, através de projetos, parcerias e novos investimentos, decorrente dos interesses políticos e econômicos predominantes na região, empresários, gabinetes, prefeituras, tem retido o fluxo de investimentos predestinados à melhoria e crescimento das comunidades. Isso reflete na questão sócio econômica e ambiental da região, como o mal uso da terra (extração de areia) e suas propriedades, ocasionando o desequilíbrio ecológico, desobstrução/erosão do solo e a não produtividades e preservação da flora extensa e rica. Telecentro na Escola de Ensino Fundamental St^{a} Marta, recebeu as mesas, 11 computadores, data show, mas não tem antena, conexão, a sala destinada para instalação não tem estrutura elétrica (fiação de modo geral). A conversa teve a participação e colaboração de Valdinei (Pedra), representante da comunidade e atuante do Projeto Ijé Ofé, Marco Antônio (cabeludo), vice-presidente da Associação. A instalação dos telecentros não foi efetivada, pois em Itancoã não chegou maquinário, apenas data show e, em Guajará Miri o espaço reservado na escola não tem estrutura. Foram feitos registros de fotos e vídeos (Guine e Reginaldo-Dj.RG), entrevistas com os representantes sobre a realidade dos telecentros locais. As comunidades quilombolas reconhecem as suas necessidades, limitações, e mesmo com as problemáticas existentes a resistência e trabalho persistem, pois as ações tem continuidade, seja nas escolas, nas associações e com os próprios moradores. Os telecentros são importantes para a comunicação entre as comunidades, para o crescimento e projetos de ensino nas escolas, para a integração e envolvimento de jovens e adultos, no intuito de fortalecer os interesses (cultura, identidade, valores) e o comprometimento com a comunidade, atribuindo o que se tem de direito que vai muito além da terra. Como encaminhamento, os representantes Pedra e Marcos, providenciarão um profissional eletricista para fazer um orçamento e ajustes necessários, no espaço que foi destinado para a instalação do Telecentro, à parti disso comunicar (agendar) a Rede para uma nova visita, contactar um correio mais próximo e ver as possibilidades de entrega/endereço, para receber antena ou qualquer outro maquinário que for preciso ser entregue.

Material produzido: relatoria (Suhellen) , áudio, vídeo e imagens (Reginaldo (RG) e Guine) -

Vídeos:

- docs/relatos com os reprentantes, Pedra e Marco. (Guajará Miri)
- Cantiga com D.Ana Faustina “Ama do Boi” (representante da secretaria de cultura da região),bumba boi batuques e jogos. (Guajará Miri)
- relatos da trajetória da viagem (Suh, RG, Guine) -

Contatos: -Valdinei (Pedra); 9249-5913 (vivo) e 8164-2455 (tim) -Marco Antônio T.Galiza (Cabeludo): 9224-2249 (vivo)

Capítulo 6

Repositórios

A arquitetura descentralizada do Baobáxia é baseada em repositórios *git* e *git-annex*.

Um elemento importante para o acervo multimídia são as operações de sincronização. As ferramentas baseadas no *git* herdam a sua natureza descentralizada e a capacidade de comunicar de forma transparente usando vários protocolos. Em particular é interessante a possibilidade de executar sincronizações com sistemas de armazenamento massivo, característica essencial na fase de criação de um novo nó, onde a primeira sincronização via rede poderia levar dias. As transferências contudo, no caso do *git-annex*, são executadas através do protocolo *rsync*[1], que gerencia eventuais interrupções, evitando retransmissões onerosas.

6.0.1 Git

Git é um sistema multiplataforma para o controle de versão distribuído, projetado para ser rápido e usável mesmo em grande projetos.

As características principais incluem:

- é totalmente distribuído e cada clone de um repositório contem o histórico inteiro das versões e no qual podem ser efetuadas operações independentemente de conexões de rede o de servidores centrais. As mudanças podem ser copiadas entre um clone e o outro e são mantidos em *branch* (ramos) diferentes, facilitando

[1] *rsync* é um Software Livre para a transfêrencia rápida e incremental de arquivos disponível no `http://rsync.samba.org/`.

as operações de *merge* (fusão). Os repositórios são facilmente acessíveis através do eficiente protocolo do Git, que além de suportar HTTP, pode funcionar junto com SSH, para obter conexões seguras e um sistema de autenticação solido e bem comum.

- suporta o *branching* (ramificação), e o *merging* (fusão), de maneira rápida e conveniente, incluindo uma serie de ferramentas para visualizar e navegar o histórico não linear das versões.
- é muito rápido e escala mesmo em projetos muito grandes e com muitas mudanças, graças a um eficiente sistema de empacotamento e memorização do histórico (é considerado o mais eficiente entre os sistemas atualmente disponíveis).
- associa um nome de versão, para cada *commit*, que é função do histórico inteiro, por isso, uma vez publicada uma versão, não é possível alterar as velhas sem ser notado. As versões podem também ser etiquetadas e assinadas digitalmente com GPG.

Git é um sistema completo que, em bom estilo Unix, é organizado em programas e comandos independentes, pensados para ser facilmente usáveis, seja automaticamente através de *scripting* seja de maneira interativa pelo usuário final. Git é, então, uma base solida para o desenvolvimento de aplicações orientadas a sincronização, a portabilidade e a gestão autônoma e descentralizada.

6.0.2 git-annex

git-annex[2] permite a gestão de arquivos com *git*, sem a necessidade de adicionar os arquivos dentro *git*. Mesmo se pode parecer paradoxal, é útil quando se trabalha com arquivos muito grandes que *git* atualmente não gerencia facilmente por limitações devidas a memoria, tempo ou espaço no disco.

Mesmo sem manter o histórico das mudanças do conteúdo do arquivo, ter a possibilidade de gerenciar arquivos com *git*, de movê-los, e exclui-los, numa árvore de pasta versionada, com uso de *branches* e de clones distribuídos, são todos bons motivos para usar *git*. E

[2] *git-annex* é um programa que estende as funcionalidades do *git* em gerir arquivos de grande tamanho disponível no `http://git-annex.branchable.com`.

os arquivos anexos (por isso o nome *git-annex*) podem coexistir no mesmo repositório *git* com os arquivos regularmente versionados.

git-annex transforma os arquivos anexos em *link* simbólicos, que são normalmente versionados por *git*.

O conteúdo dos arquivos é mantido por *git-annex* em um acervo chave/valor distribuído que corresponde aos clones de um dado repositório *git*. Praticamente *git-annex* memoriza o conteúdo do arquivo em uma subpasta de `.git/annex/`.

A primeira vez que um arquivo é adicionado no *git-annex*, é calculada uma chave, normalmente fazendo um *hash* do seu conteúdo. *git-annex* todavia suporta vários *backend* que podem produzir diferentes tipos de chaves. O arquivo que é adicionado no *git* nada mais é que um *link* simbólico para a chave memorizada no `.git/annex/`. Se o conteúdo do arquivo for modificado, é gerada uma outra chave, e o *link* é alterado.

O conteúdo do arquivo pode ser transferido de um repositório para outro por *git-annex*, que além de manter controle de quem mantem o que, permite criar um mapa das copias disponíveis e impor um número mínimo de cópias. Essas informações são mantidas em um *branch* separado, chamado "*git-annex*", e as operações de sincronização, são simplesmente *push* e *pull* entre os vários clones dos repositórios.

git-annex suporta:

- localização das cópias (*location tracking*)
- download seletivo dos conteúdos
- gestão da confiança dos repositórios
- gestão do número minimo de cópias
- vários *backend* para as chaves (SHA[3], WORM[4])
- vários *backend* para os conteúdos/valores (BUP[5], rsync, web, S3[6])

[3] Secure Hash Algoritm, (SHA), é um algoritmo usado em sistemas chave/valor onde as chaves são calculadas através de uma função criptográfica dos valores.

[4] O algoritmo WORM identifica os arquivos em base ao nome, dimensão e data de alteração.

[5] BUP é um sistema para *backup* a alta eficiência disponível no: `https://github.com/apenwarr/bup`.

[6] Amazon Simple Storage Service, (S3) é uma infraestrutura para a memorização dos dados totalmente redundante, disponível no: `aws.amazon.com/`.

6.1 Gestão de repositórios multiplos

A arquitetura baseadas em repositórios se adapta ao contexto de "redes federadas" onde para cada "Rede" corresponde um repositório. A solução proposta no Baobáxia inclui já a possibilidade de cadastrar e gerenciar varias redes. A associação rede/repositório permite a escalabilidade da arquitetura e uma gestão diferenciada dos conteúdos. Cada mucua pode se associar a diferentes redes, por exemplo, Rede Mocambos, Estaçoes Digitais, etc.

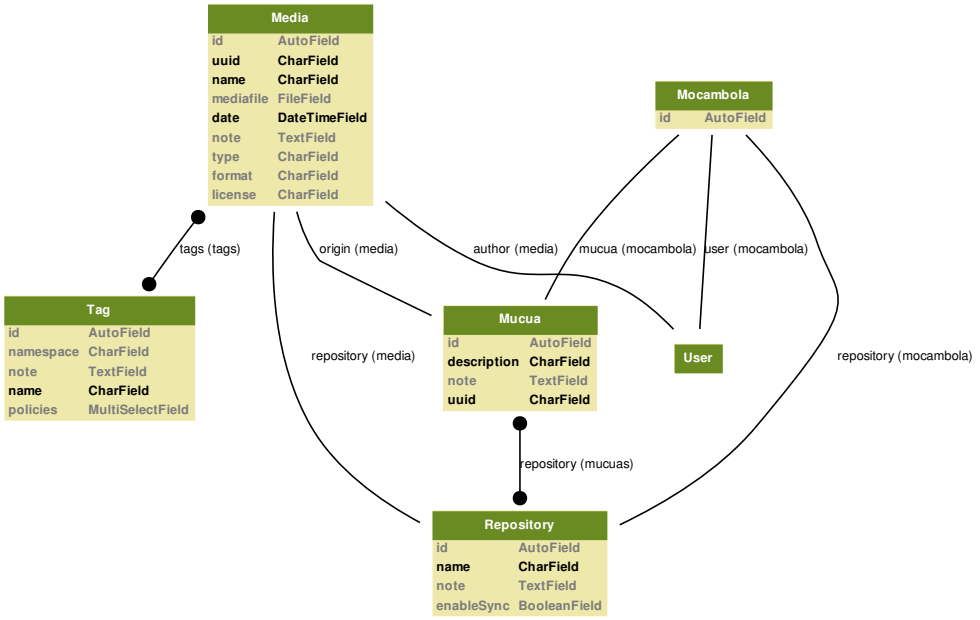

FIGURA 6.1: Diagrama UML do BBX

6.1.1 API e rotas

A API REST proposta prevê a gestão de repositórios multiplos em diferentes mucuas, usando o padrão de endereços:

/repositorio/mucua/

Por exemplo, para acessar um “media” (uuid aa9f9019-e4f2-4040-bc46-c2e15b66bebc) publicado na “Rede Mocambos”, na mucua “Dandara” o endereço é:

$$/mocambos/dandara/media/aa9f9019-e4f2-4040-bc46-c2e15b66bebc$$

ou no caso da “Estaçao Digital” na “Samambaia” seria:

$$/fbb/samambaia/media/aa9f9019-e4f2-4040-bc46-c2e15b66bebc$$

O detalhamento da API e das rotas será definido em capitulo especifico.

6.1.2 Autenticação

O Baobáxia pode gerenciar dados de varias redes/organizações e portanto precisa diferenciar os usuários por Rede e Mucua[7]. No caso as credenciais dos usuários são mantidas em arquivos versionados pelo git nos repositórios correspondentes.

Para manter compatibilidade com outras aplicações do django mantivemos o User padrão do django usando somente o campo username para codificar as informarções necessárias.

Um mocambola[8] é definido por nome, mucua e repositorio no padrão:

$$nome@mucua.repositorio.tld$$

.

O django é muito flexivel e suporta diferentes sistemas de autenticação, com a possibilidade de definir seu proprio backend personalizado.

O backend de autenticação personalizado, “FileBackend” se encontra em *bbx/auth.py*.

No BBX, os usuários são serializados em formato *json* e armazenados seguindo o seguinte padrão de localização:

$$/repositorio/mucua/mocambolas/usuario.json$$

[7] A mucua de alguma forma identifica uma comunidade
[8] Usuário

Por exemplo, no caso do mocambola “zumbi” da Casa de Cultura Tainã cuja mucua se chama “dandara”:

zumbi@dandara.mocambos.net

/mocambos/dandara/mocambolas/zumbi@dandara.mocambos.net

Notar que o repositório não inclui a extensão de dominio de primeiro nível, no caso do exemplo “.net”, que permanece no username do mocambola.

"A força da rede esta nos nós"

TC

Apêndice A

Listagem do código

A.1 bbx

A.1.1 bbx/auth.py

```
# -*- coding: utf-8 -*-
from django.conf import settings
from django.contrib.auth.models import User, check_password
from mocambola.serializers import UserSerializer
from mucua.models import Mucua
from repository.models import Repository
from bbx.settings import MOCAMBOLA_DIR, REPOSITORY_DIR

from StringIO import StringIO
from rest_framework.parsers import JSONParser

from urlparse import urlparse
import os
import re

try:
    import json                # Python 2.6
except ImportError:
    import simplejson as json  # Python 2.5

class FileBackend(object):
    """
    Authenticate against the serialized User object in repository
    """

    # TODO LOW: Limpar a arrumar melhor o codigo

    def authenticate(self, username=None, password=None):
        match = re.findall("(.*)@(.*)\.(.*)\.(.*)$", username)
        if match:
            current_mocambola, current_mucua, current_repository, term = match[0]
            # verifica se mucua e repositorio sao validos
            try:
                current_mucua = Mucua.objects.get(note = current_mucua)
            except Mucua.DoesNotExist:
                return None
            try:
                current_repository = Repository.objects.get(name = current_repository)
            except Repository.DoesNotExist:
                return None
        else:
            print "invalid address"
            return None
```

```
        # Get file from MOCAMBOLA_DIR
        mocambola_path = os.path.join(str(REPOSITORY_DIR), str(current_repository),
    str(current_mucua), MOCAMBOLA_DIR)

        for jmocambola in os.listdir(mocambola_path):
            print "jmocambola: ", jmocambola
            print "current_mocambola: ", current_mocambola

            if jmocambola == username + '.json':
                # Deserialize the customized User object
                mocambola_json_file = open(os.path.join(mocambola_path, jmocambola))
                data = JSONParser().parse(mocambola_json_file)
                u = User()
                serializer = UserSerializer(u, data=data)
                print "serializer.errors: ", serializer.errors
                print "serializer.is_valid: ", serializer.is_valid()

                current_user = serializer.object

                print username , "==", current_user.username

                login_valid = (username == current_user.username)
                pwd_valid = check_password(password, current_user.password)

                print "current_user: ", current_user
                print "login_valid: ", login_valid
                print "pwd_valid: ", pwd_valid

                if login_valid and pwd_valid:
                    try:
                        user = User.objects.get(username=username)
                    except User.DoesNotExist:
                        print "User.DoesNotExist"
                        # Create a new user. Note that we can set password
                        # to anything, because it won't be checked; the password
                        # from settings.py will.
                        user = User(username=username, password=current_user.password, \
                                    is_staff=current_user.is_staff,
    is_superuser=current_user.is_superuser)
                        user.save()
                    print "User:", user
                    return user
                else:
                    print "invalid username and/or password"
                    return None
                return True
            # fim do if
        # fim do for
        return None

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except User.DoesNotExist:
            return None
```

A.1.2 bbx/utils.py

```
# -*- coding: utf-8 -*-
from django.db import models
from django import forms
from django.utils.text import capfirst

import os
import errno

def check_if_path_exists_or_create(path):
    try:
        os.makedirs(path)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            raise
```

```
# Multi Select Field, originally taken from:
# http://djangosnippets.org/snippets/1200/

class MultiSelectFormField(forms.MultipleChoiceField):
    widget = forms.CheckboxSelectMultiple

    def __init__(self, *args, **kwargs):
        self.max_choices = kwargs.pop('max_choices', 5)
        super(MultiSelectFormField, self).__init__(*args, **kwargs)

    def clean(self, value):
        if not value and self.required:
            raise forms.ValidationError(self.error_messages['required'])
        if value and self.max_choices and len(value) > self.max_choices:
            raise forms.ValidationError('You must select a maximum of %s choice%s.'
                    % (apnumber(self.max_choices), pluralize(self.max_choices)))
        return value

class MultiSelectField(models.Field):
    __metaclass__ = models.SubfieldBase

    def get_internal_type(self):
        return "CharField"

    def get_choices_default(self):
        return self.get_choices(include_blank=False)

    def _get_FIELD_display(self, field):
        value = getattr(self, field.attname)
        choicedict = dict(field.choices)

    def formfield(self, **kwargs):
        # don't call super, as that overrides default widget if it has choices
        defaults = {'required': not self.blank, 'label': capfirst(self.verbose_name),
                    'help_text': self.help_text, 'choices':self.choices}
        if self.has_default():
            defaults['initial'] = self.get_default()
        defaults.update(kwargs)
        return MultiSelectFormField(**defaults)

    def validate(self, value, model_instance):
        # Needed couse it's a custom field .. see more:
        # https://groups.google.com/forum/#!topic/django-users/J0NXIUo7TdY
        # Should do something?
        return

    def get_db_prep_value(self, value, connection, prepared=False):
        if isinstance(value, basestring):
            return value
        elif isinstance(value, list):
            return ",".join(value)

    def to_python(self, value):
        if isinstance(value, list):
            return value
        return value.split(",")

    def contribute_to_class(self, cls, name):
        super(MultiSelectField, self).contribute_to_class(cls, name)
        if self.choices:
            func = lambda self, fieldname = name, choicedict =
     dict(self.choices):",".join([choicedict.get(value,value) for value in
     getattr(self,fieldname)])
            setattr(cls, 'get_%s_display' % self.name, func)
```

A.2 media

A.2.1 media/models.py

```
# -*- coding: utf-8 -*-

from django.db import models
from django.contrib.auth.models import User
from django.forms import ModelForm
from tag.models import Tag
from bbx.settings import REPOSITORY_DIR, POLICIES_DIR

from django.dispatch import receiver
from django.db.models.signals import post_save
from django.dispatch import receiver

from django.utils.translation import ugettext_lazy as _

import os
import uuid
import logging
import subprocess
from datetime import datetime
import exceptions
from importlib import import_module

try:
    from django.utils.encoding import force_unicode # NOQA
except ImportError:
    from django.utils.encoding import force_text as force_unicode # NOQA

TYPE_CHOICES = ( ('audio', 'audio'), ('imagem', 'imagem'), ('video', 'video'),
     ('arquivo','arquivo') )
FORMAT_CHOICES = ( ('ogg', 'ogg'), ('webm', 'webm'), ('mp4', 'mp4'), ('jpg','jpg'), ('pdf',
     'pdf') )


def mediaFileName(instance, filename):
    logging.debug(os.path.join(getFilePath(instance) + instance.getFileName()))
    return os.path.join(getFilePath(instance), instance.getFileName())

def generateUUID():
    return str(uuid.uuid4())

def getFilePath(instance):
    if instance.date == '':
        t = datetime.now
        date = t.strftime("%y/%m/%d/")
    else:
        date = instance.date.strftime("%y/%m/%d/")

    t = datetime.now()
    return os.path.join(REPOSITORY_DIR, instance.getRepository(),
                        instance.getMucua(), instance.getType(),
                        date)

def getTypeChoices():
    return TYPE_CHOICES

def getFormatChoices():
    return FORMAT_CHOICES

class Media(models.Model):
    uuid = models.CharField(_('uuid'),
                            help_text = _('Media universal unique identifier'),
                            max_length=36, default="")
    name = models.CharField(_('name'),
                            help_text = _('Media name'),
                            max_length=100, default='')
    mediafile = models.FileField(upload_to=mediaFileName, blank=True)
    date = models.DateTimeField(_('date'),
                                help_text = _('Media criation date'))
    note = models.TextField(_('note'),
                            help_text=_('Note.. use as you wish!'),
                            max_length=300, blank=True)
    author = models.ForeignKey(User)
    origin = models.ForeignKey('mucua.Mucua', related_name='media')
    type = models.CharField(_('type'),
                            help_text = _('Type of the media, like image, document, video, ...'),
                            max_length=14, choices=TYPE_CHOICES,
                            default='arquivo', blank=True)
    format = models.CharField(_('format'),
                              help_text = ('Format of the media, like ogg, jpg, pdf, ...'),
                              max_length=14, choices=FORMAT_CHOICES,
```

```
                                default='ogg', blank=True)
    license = models.CharField(_('license'),
                               help_text = _('License of the media, like, cc, gpl, bsd, ...'),
                               max_length=100, blank=True)
    repository = models.ForeignKey('repository.Repository', related_name='media')
    tags = models.ManyToManyField(Tag, related_name='tags')

    def __init__(self, *args, **kwargs):
        super(Media, self).__init__(*args, **kwargs)
        self._meta.get_field('uuid').default = force_unicode(uuid.uuid4())

    def __unicode__(self):
        return self.name

    def getName(self):
        return self.name

    def getFileName(self):
        return str(self.uuid)  + '.' + self.format

    def getRepository(self):
        return self.repository.getName()

    def getMucua(self):
        return self.origin.description

    def getType(self):
        return self.type

    def getFormat(self):
        return self.format

    # # perform validation
    # def clean(self):
    #     from django.core.exceptions import ValidationError
    #     try:
    #         self.full_clean()
    #     except ValidationError as e:
    #         # do stuff
    #         print e

    def getTags(self):
        return self.tags

    class Meta:
        ordering = ('date',)

class TagPolicyDoesNotExist(exceptions.Exception):
    def __init__(self,args=None):
        self.args = args


@receiver(post_save, sender=Media)
def startPostSavePolicies(instance, **kwargs):
    """Intercepta o sinal de *post_save* de objetos multimedia (*media*) e inicializa as policies
     de post-save"""
# FIX: parece que nao intercepta o sinal quando se cria um Media,
# somente funciona nos "saves" seguidos. Deve ser um problema de
# disponibilidade da çrelaao com a tag.

    tags = instance.getTags()
    if tags.all():
        for tag in tags.all():
            try:
                for policy in tag.policies:
                    print policy
                    if "postSave" in policy:
                        policyModule = "policy." + policy
                        module = import_module(policyModule)
                        result = getattr(module, policy)(instance)
            except TagPolicyDoesNotExist:
                return []
```

A.2.2 media/serializers.py

```
from django.forms import widgets
from rest_framework import serializers
from media.models import Media, FORMAT_CHOICES, TYPE_CHOICES
from tag.models import Tag
#from mucua.models import Mucua
from tag.serializers import TagSerializer
from mucua.serializers import MucuaSerializer
#from repository.serializers import RepositorySerializer
from rest_framework.renderers import JSONRenderer

from django.db.models import get_model


class MediaSerializer(serializers.ModelSerializer):
    # com essas linhas, media puxa apenas referencia (nao objeto completo)
    # nao sei se mantemos assim ou se puxamos o relacionamento completo
    tags = serializers.RelatedField(many = True)
    origin = serializers.RelatedField(many = False)
    repository = serializers.RelatedField(many = False)
#    author = serializers.RelatedField(many = False)

#    tags = TagSerializer(required=False)
#    origin = MucuaSerializer()

#    from repository.serializers import RepositorySerializer
#    repository = RepositorySerializer()

    class Meta:
        model = Media
        fields = ('date', 'uuid', 'name', 'note', 'author', 'type', 'format', 'license',
     'mediafile', 'origin', 'repository', 'tags')
#        depth = 1   # se comentar linhas de cima, ativar essa

    def restore_object(self, attrs, instance=None):
        """
        Create or update a new media instance, given a dictionary
        of deserialized field values.

        Note that if we don't define this method, then deserializing
        data will simply return a dictionary of items.
        """
        if instance:
            # Update existing instance
            instance.date = attrs.get('date', instance.date)
            instance.uuid = attrs.get('uuid', instance.uuid)
            instance.name = attrs.get('name', instance.name)
            instance.note = attrs.get('note', instance.note)
            instance.author = attrs.get('author', instance.author)
            instance.origin = attrs.get('origin', instance.origin)
            instance.type = attrs.get('type', instance.type)
            instance.format = attrs.get('format', instance.format)
            instance.license = attrs.get('license', instance.license)
            instance.mediafile = attrs.get('mediafile', instance.mediafile)
            instance.tags = attrs.get('tags', instance.tags)
            instance.repository = attrs.get('repository', instance.repository)

            return instance

        # Create new instance
        return Media(**attrs)

    def getJSON(self):
        return JSONRenderer().render(self.data)
```

A.2.3 media/views.py

```
from rest_framework import status
from rest_framework.decorators import api_view
from rest_framework.response import Response
from django.shortcuts import get_object_or_404, render, render_to_response, redirect
from django.http import HttpResponseRedirect, HttpResponse
from django.contrib.auth.models import User
from django.db.models import Q
```

```
from media.models import Media, mediaFileName, getFilePath
from tag.models import Tag
from media.forms import MediaForm
from media.serializers import MediaSerializer
from media.models import getTypeChoices, getFormatChoices
from bbx.settings import DEFAULT_MUCUA, DEFAULT_REPOSITORY
import datetime
import os
import subprocess
import uuid
from os import path
import mimetypes

from mucua.models import Mucua
from repository.models import Repository

redirect_base_url = "http://localhost:8000/"  # TODO: tirar / mover

@api_view(['GET'])
def media_list(request, repository, mucua, args=None, format=None):
    """
    List all medias, or search by terms
    """

    if request.method == 'GET':
        """
        list medias
        """

        # pegando sessao por url
        redirect_page = False

        # REPOSITORIO: verifica se existe no banco, senao pega a default
        try:
            mucua = Mucua.objects.get(description = mucua)
        except Mucua.DoesNotExist:
            mucua = Mucua.objects.get(description = DEFAULT_MUCUA)
            redirect_page = True

        try:
            repository = Repository.objects.get(name = repository)
        except Repository.DoesNotExist:
            repository = Repository.objects.get(name = DEFAULT_REPOSITORY)
            redirect_page = True

        # redirect
        if redirect_page:
            return HttpResponseRedirect(redirect_base_url + repository.name + '/' +
    mucua.description + '/bbx/search/')

        # TODO LOW: futuramente, otimizar query de busca - elaborar query

        # listagem de conteudo filtrando por repositorio e mucua
        medias = Media.objects.filter(repository = repository.id).filter(origin = mucua.id)
        # sanitizacao -> remove '/' do final
        args = args.rstrip('/')

        # pega args da url se tiver
        if args:
            for arg in args.split('/'):
                # verifica se a palavra eh tipo, formato ou tag e filtra
                if arg in [key for (key, type_choice) in getTypeChoices() if arg == type_choice]:
                    medias = medias.filter(type__iexact = arg)
                elif arg in [key for (key, format_choice) in getFormatChoices() if arg ==
    format_choice]:
                    medias = medias.filter(format__iexact = arg)
                else:
                    medias = medias.filter(Q(tags__name__icontains = arg ) | Q(name__icontains =
    arg) | Q(note__icontains = arg))

        # serializa e da saida
        serializer = MediaSerializer(medias, many=True)
        return Response(serializer.data)


@api_view(['GET', 'PUT', 'DELETE', 'POST'])
def media_detail(request, repository, mucua, pk = None, format=None):
    """
    Retrieve, create, update or delete a media instance.
    """
```

```
    # pegando sessao por url
    redirect_page = False

    try:
        mucua = Mucua.objects.get(description = mucua)
    except Mucua.DoesNotExist:
        mucua = Mucua.objects.get(description = DEFAULT_MUCUA)
        redirect_page = True

    try:
        repository = Repository.objects.get(name = repository)
    except Repository.DoesNotExist:
        repository = Repository.objects.get(name = DEFAULT_REPOSITORY)
        redirect_page = True

    # redirect
    if redirect_page:
        return HttpResponseRedirect(redirect_base_url + repository.name + '/' + mucua.description
     + '/media/')

    # TODO: get author (url?)
    author = User.objects.get(pk = 1)

    if pk:
        try:
            media = Media.objects.get(uuid=pk)
        except Media.DoesNotExist:
            return Response(status=status.HTTP_404_NOT_FOUND)

    if request.method == 'GET':
        if pk == '':
            return HttpResponseRedirect(redirect_base_url + repository.name + '/' +
     mucua.description + '/bbx/search')

        serializer = MediaSerializer(media)
        return Response(serializer.data)

    elif request.method == 'PUT':
        if pk == '':
            return HttpResponseRedirect(redirect_base_url + repository.name + '/' +
     mucua.description + '/bbx/search')
        media.name = request.DATA['name']
        media.note = request.DATA['note']
        media.type = request.DATA['type']
        media.license = request.DATA['license']
#         media.date = request.DATA['date']

        media.save()
        if media.id:
            tags = request.DATA['tags'] if iter(request.DATA['tags']) == True else
     request.DATA['tags'].split(',')
            media.tags.clear()
            for tag in tags:
                try:
                    if tag.find(':') > 0:
                        args = tag.split(':')
                        tag = args[1]
                    tag = Tag.objects.get(tag = tag)
                except Tag.DoesNotExist:
                    tag = Tag.objects.create(tag = tag)
                    tag.save()

                media.tags.add(tag)

            # TODO: return serialized data
            return Response("updated media - OK", status=status.HTTP_201_CREATED)
        else:
            return Response("error while creating media", status=status.HTTP_400_BAD_REQUEST)

        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    elif request.method == 'POST':
        """
        create a new media
        """
```

```
        # Linha curl mista para testar upload E mandar data
        # $ curl -F "name=teste123" -F "tags=entrevista" -F "note=" -F "license=" -F
    "date=2013/06/07" -F "type=imagem" -F "mediafile=@img_0001.jpg" -X POST
    http://localhost:8000/redemocambos/dandara/media/ > /tmp/x.html
        media = Media(repository = repository,
                      origin = mucua,
                      author = author,
                      name = request.DATA['name'],
                      note = request.DATA['note'],
                      type = request.DATA['type'],
                      license = request.DATA['license'],
                      date = request.DATA['date'],
                      mediafile = request.FILES['mediafile']
                      )

        media.save()
        if media.id:
            # get tags by list or separated by ','
            tags = request.DATA['tags'] if iter(request.DATA['tags']) == True else
    request.DATA['tags'].split(',')
            for tag_name in tags:
                try:
                    if tag_name.find(':') > 0:
                        args = tag.split(':')
                        tag_name = args[1]
                        tag_namespace = args[0]
                    tag = Tag.objects.get(name=tag_name)
                except Tag.DoesNotExist:
                    tag = Tag.objects.create(name=tag_name)
                    tag.save()

                media.tags.add(tag)
            media.save() # salva de novo para chamar o post_save

            # TODO: return serialized data
            return Response("created media - OK", status=status.HTTP_201_CREATED)
        else:
            return Response("error while creating media", status=status.HTTP_400_BAD_REQUEST)

    elif request.method == 'DELETE':

        media.delete()

        return Response(status=status.HTTP_204_NO_CONTENT)
```

A.2.4 media/urls.py

```
from django.conf.urls import patterns, url
from rest_framework.urlpatterns import format_suffix_patterns

urlpatterns = patterns('media.views',
    url(r'^(?P<repository>\w+)/(?P<mucua>\w+)/bbx/search/(?P<args>[\w\/-_]*)$', 'media_list'),
    url(r'^(?P<repository>\w+)/(?P<mucua>\w+)/media/(?P<pk>[\w\-]*)$', 'media_detail'),
    url(r'^medias/(?P<pk>[0-9]+)/$', 'media_detail'),
)

urlpatterns = format_suffix_patterns(urlpatterns)
```

A.3 repository

```
          /
      _ \ /
       \_| |/__
 <==     | /  ()
 ==>     |/   NPDD/Rede Mocambos
```

```
 nnn    |
        |    http://wiki.mocambos.net/wiki/NPDD
        |    "Vamos fazer um mundo digital mais do nosso jeito..."
______/~~~\_____________________________________________________________ - - -

       /
      GITANNEX                                        Software LIVRE! GPLv3

Aplicacao django para gerir repositorios Git Annex (git-annex.branchable.com)


Pode usar com o django mmedia:
https://github.com/RedeMocambos/mmedia
```

A.3.1 repository/models.py

```
# -*- coding: utf-8 -*-

from django.db import models
from django.db.models.base import ModelBase
from django.contrib.auth.models import User
from django.conf import settings
from django.db.models.signals import post_save
from django.dispatch import receiver

from media.models import Media
from media.models import getFilePath

from django.db.models import get_model
from repository.signals import filesync_done

from django.utils.translation import ugettext_lazy as _
from bbx.settings import DEFAULT_REPOSITORY


import os
import datetime
import subprocess
import logging
import exceptions

"""
Modelos da aplicacao Django.

Neste arquivo sao definidos os modelos de dados da aplicacao *gitannex*.
"""

# REPOSITORY_CHOICE é uma tupla com repositorios dentro da pasta /annex
REPOSITORY_CHOICES = [ ('mocambos', 'mocambos'), ('sarava', 'sarava') ]
logger = logging.getLogger(__name__)
repository_dir = settings.REPOSITORY_DIR


# Connecting to Media signal
@receiver(post_save, sender=Media)
def gitMediaPostSave(instance, **kwargs):
    """Intercepta o sinal de *post_save* de objetos multimedia (*media*) e adiciona o objeto ao
     repositorio."""
    from media.serializers import MediaSerializer
    logger.debug(instance.type)
    logger.debug(type(instance))
    gitAnnexAdd(instance.getFileName(), getFilePath(instance))
    serializer = MediaSerializer(instance)
    mediapath = getFilePath(instance)+'/'
    mediadata = instance.uuid + '.json'
    fout = open(mediapath + mediadata, 'w')
    fout.write(str(serializer.getJSON()))
    fout.close()
    gitAdd(mediadata, mediapath)
    gitCommit(instance.getFileName(), instance.author.username, instance.author.email,
     getFilePath(instance))

def getDefaultRepository():
```

```
    return Repository.objects.get(name = DEFAULT_REPOSITORY)

def getAvailableRepositories():
    return REPOSITORY_CHOICES

def _getAvailableFolders(path):
    """Procura as pastas que podem ser inicializada como repositorio, retorna a lista das
     pastas."""
    folderList = [( name , name ) for name in os.listdir(path) \
                      if os.path.isdir(os.path.join(path, name))]
    return folderList

def gitAdd(fileName, repoDir):
    """Adiciona um arquivo no repositorio."""
    logger.info('git add ' + fileName)
    cmd = 'git add '+ fileName
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    pipe.wait()

def gitCommit(fileTitle, authorName, authorEmail, repoDir):
    """Executa o *commit* no repositorio impostando os dados do author."""
    logger.info('git commit --author="' + authorName + ' <' + authorEmail +'>" -m "' + fileTitle
     + '"')
    cmd = 'git commit --author="' + authorName + ' <' + authorEmail +'>" -m "' + fileTitle + '"'
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    pipe.wait()

def gitPush(repoDir):
    """Executa o *push* do repositorio, atualizando o repositorio de origem."""
    logger.info('git push ')
    cmd = 'git push '
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    pipe.wait()

def gitPull(repoDir):
    """Executa o *pull* do repositorio, atualizando o repositorio local."""
    logger.info('git pull ')
    cmd = 'git pull '
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    pipe.wait()

def gitGetSHA(repoDir):
    """Resgata o codigo identificativo (SHA) da ultima revisao do repositorio, retorna o
     codigo."""
    logger.info('git rev-parse HEAD')
    cmd = 'git rev-parse HEAD'
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    output,error = pipe.communicate()
    logger.debug('>>> Revision is: ' + output)
    return output

def gitAnnexAdd(fileName, repoDir):
    """Adiciona um arquivo no repositorio *git-annex*."""
    logger.info('git annex add ' + fileName)
    cmd = 'git annex add ' + fileName
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    pipe.wait()

def gitAnnexMerge(repoDir):
    """Executa o *merge* do repositorio, reunindo eventuais diferencias entre o repositorio local
     e remoto."""
    logger.info('git annex merge ')
    cmd = 'git annex merge '
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    pipe.wait()

def gitAnnexCopyTo(repoDir):
    """Envia os conteudos binarios para o repositorio remoto."""
    # TODO: Next release with dynamic "origin"
    logger.info('git annex copy --fast --to origin ')
    cmd = 'git annex copy --fast --to origin'
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    pipe.wait()

def gitAnnexGet(repoDir):
    """Baixa os conteudos binarios desde o repositorio remoto."""
    # TODO: Next release with possibility to choice what to get
    logger.info('git annex get .')
    cmd = 'git annex get .'
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
```

```
    pipe.wait()

def gitAnnexSync(repoDir):
    """Sincroniza o repositorio com os outros clones remotos."""
    # TODO: Next release with possibility to choice what to get
    logger.info('git annex sync')
    cmd = 'git annex sync'
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir)
    pipe.wait()

def gitAnnexStatus(repoDir):
    """View all mucuas in a given repository"""
    logger.info('git annex info/status')
    cmd = 'git annex info --json'
    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir, stdout=subprocess.PIPE)
    return pipe.stdout.read()
    #except GitAnnexCommandError:
    #    cmd = 'git annex info --json'
    #    pipe = subprocess.Popen(cmd, shell=True, cwd=repoDir, stdout=subprocess.PIPE)
    #    return pipe.stdout.read()



def runScheduledJobs():
    """Executa as operacoes programadas em todos os repositorios. """
    allRep = Repository.objects.all()
    for rep in allRep:
        if rep.enableSync:
            # TODO: Manage time of syncing
            # if rep.syncStartTime >= datetime.datetime.now():
            rep.syncRepository()


class Repository(models.Model):
    """Classe de implementacao do modelo de repositorio *git-annex*.

    Atributos:
        name: nome do repositorio (campo preenchido por *_getAvailableFolders()*)
        note: Note.. use as you wish!
        enableSync = flag booleano para abilitar ou disabilitar a sincronizacao
    """

    name = models.CharField(_('name'),
                            help_text=_('Repository name taken from available repositories'),
                            max_length=60,
                            choices=_getAvailableFolders(repository_dir))
    note = models.TextField(_('notes'),
                            help_text=_('Note.. use as you wish!'),
                            max_length=300, blank=True)

    enableSync = models.BooleanField(_('synchronize'),
                                     help_text=_('Tick here to enable'
                                                 'syncronization of this repository'),)

    class Meta:
        ordering = ('name',)
        verbose_name = _('repository')
        verbose_name_plural = _('repositories')

    def __unicode__(self):
        return self.name


    def getName(self):
        """Retorna o nome do repositorio."""
        return str(self.name)

    def getPath(self):
        """Retorna o caminho no disco (path) do repositorio."""
        return os.path.join(repository_dir, self.getName())

    def createRepository(self):
        """Cria e inicializa o repositorio."""
        _createRepository(self.name, self.getPath())

    def cloneRepository(self):
        """Clona e inicializa o repositorio."""
        _cloneRepository(self.getPath, self.name)
```

```
    def syncRepository(self):
        """Sincroniza o repositorio com sua origem."""
        gitAnnexSync(self.repositoryURLOrPath)

        filesync_done.send(sender=self, name=self.getName(), \
                           repositoryDir=self.getPath())

    def save(self, *args, **kwargs):
        super(Repository, self).save(*args, **kwargs)


class GitAnnexCommandError(exceptions.Exception):
    def __init__(self,args=None):
        self.args = args
```

A.3.2 repository/signals.py

```
import logging
import django.dispatch

"""
Arquivo de definicao dos sinais.

Os sinais sao usados para interligar diferentes *apps* do Django.
"""

def get_subclasses(classes, level=0):
    """
    Procura as subclasses da uma classe dada, retorna a lista de subclasses.

    Return the list of all subclasses given class (or list of classes) has.
    Inspired by this question:

     http://stackoverflow.com/questions/3862310/how-can-i-find-all-subclasses-of-a-given-class-in-python
    """
    # for convenience, only one class can can be accepted as argument
    # converting to list if this is the case
    if not isinstance(classes, list):
        classes = [classes]

    if level < len(classes):
        classes += classes[level].__subclasses__()
        return get_subclasses(classes, level+1)
    else:
        return classes

def receiver_subclasses(signal, sender, dispatch_uid_prefix, **kwargs):
    """
    Decorador para conectar todos os sinais do *receiver* e das subclasses do *receiver*.

    A decorator for connecting receivers and all receiver's subclasses to signals. Used by
     passing in the
    signal and keyword arguments to connect::

        @receiver_subclasses(post_save, sender=MyModel)
        def signal_receiver(sender, **kwargs):
            ...
    """
    def _decorator(func):
        all_senders = get_subclasses(sender)
        logging.info(all_senders)
        for snd in all_senders:
            signal.connect(func, sender=snd, dispatch_uid=dispatch_uid_prefix+'_'+snd.__name__,
     **kwargs)
        return func
    return _decorator


## Novo sinal para alertar que os repositorios sao sincronizados
filesync_done = django.dispatch.Signal(providing_args=["repositoryName", "repositoryDir"])
```

A.3.3 repository/management/commands/run scheduled jobs.py

```
from django.core.management.base import NoArgsCommand, CommandError

from repository.models import Repository, runScheduledJobs

"""
Definicoes do comando para executar as operacoes planejadas.
"""

class Command(NoArgsCommand):
    """Executa as operacoes planejadas."""
    help = 'Run scheduled jobs related to git repositories'

    def handle_noargs(self, **options):
        runScheduledJobs()
```

A.3.4 repository/serializers.py

```
from django.forms import widgets
from rest_framework import serializers
from repository.models import Repository

class RepositorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Repository
        fields = ('name', 'note', 'enableSync')

    def restore_object(self, attrs, instance=None):

        if instance:
            # Update existing instance
            # TODO

            return instance

        # Create new instance
        return Repository(**attrs)
```

A.4 mucua

A.4.1 mucua/models.py

```
# -*- coding: utf-8 -*-

from django.db import models
#from repository.models import Repository
from django.conf import settings
#from repository.models import gitAnnexStatus
from django.contrib.auth.models import User
from bbx.settings import DEFAULT_REPOSITORY, DEFAULT_MUCUA
from django.db.models import get_model
from django.db.utils import DatabaseError
import exceptions

from django.contrib import admin

# MUCUA_NAME_UUID é uma tupla com nome e uuid da mucua (pode ler do settings.py)
# MUCUA_NAME_UUID = settings.MUCUA_NAME_UUID
# MUCUA_NAME_UUID = [ ('a30a926a-3a8c-11e2-a817-cb26bd9bc8d3','dandara'),
    ('0492621a-4195-11e2-b8c7-43de40a4e11c','acotirene') ]

def getDefaultMucua():
    return Mucua.objects.get(note = DEFAULT_MUCUA)
```

```
def getAvailableMucuas(uuid=None, repository=None):
    if not repository:
        try:
#            repository_model = get_model('repository', 'Repository')
            from repository.models import getDefaultRepository
            repository = getDefaultRepository()
        except DatabaseError:
            return []

    import json
    from repository.models import gitAnnexStatus
    jsonRepositoryStatus = json.loads(gitAnnexStatus(repository.getPath()))

    if uuid:
        for mucua in jsonRepositoryStatus['semitrusted repositories']:
            if mucua['uuid'] == uuid:
                print "Mucua description: ", mucua['description']
                return mucua['description']
    else:
        return [( mucua['uuid'], mucua['description'] ) for mucua in
     jsonRepositoryStatus['semitrusted repositories']]


class RepositoryDoesNotExist(exceptions.Exception):
    def __init__(self,args=None):
        self.args = args

class MucuaAdmin(admin.ModelAdmin):
    readonly_fields=('uuid',)


class Mucua(models.Model):
    description = models.CharField(max_length=100, editable=False)
    note = models.TextField(max_length=300, blank=True)
    uuid = models.CharField("Mucua", max_length=36, choices=getAvailableMucuas(),
     default='dandara')
    repository = models.ManyToManyField('repository.Repository', related_name='mucuas')
    mocambolas = models.ManyToManyField(User, through='mocambola.Mocambola',
     related_name='mucuas')

    def getDescription(self):
        return self.description

    def save(self, *args, **kwargs):
        print "Self.uuid: ", self.uuid
        self.description = getAvailableMucuas(uuid=self.uuid)
        super(Mucua, self).save(*args, **kwargs) # Call the "real" save() method.

    def __unicode__(self):
        return self.description

    class Meta:
        ordering = ('description',)
```

A.4.2 mucua/serializers.py

```
from django.forms import widgets
from rest_framework import serializers
from mucua.models import Mucua

class MucuaSerializer(serializers.ModelSerializer):
    class Meta:
        model = Mucua
        fields = ('description', 'note', 'uuid', 'repository', 'mocambolas')

    def restore_object(self, attrs, instance=None):
        if instance:
            # Update existing instance
            instance.description = attrs.get('description', instance.description)
            instance.note = attrs.get('namespace', instance.note)
            instance.uuid = attrs.get('uuid', instance.uuid)

            return instance
```

```
        # Create new instance
        return Mucua(**attrs)
```

A.4.3 mucua/views.py

```
from rest_framework import status
from rest_framework.decorators import api_view
from rest_framework.response import Response
from django.shortcuts import get_object_or_404, render, render_to_response, redirect
from django.http import HttpResponseRedirect, HttpResponse
from bbx.settings import DEFAULT_MUCUA
from mucua.models import Mucua,getAvailableMucuas,getDefaultMucua
from repository.models import Repository
from mucua.serializers import MucuaSerializer

@api_view(['GET'])
def mucua_list(request, repository = None):
    """
    List all mucuas
    """
    if repository:
        try:
            repository = Repository.objects.get(name = repository)
        except Repository.DoesNotExist:
            return Response("Repository not found")

    mucuas = getAvailableMucuas(None, repository)   # retorna tupla de mucuas
    mucuas_list = []

    if mucuas is None:
        return Response(None)

    for mucua_obj in mucuas:
        mucua_note = mucua_obj[1]

        try:
            mucua = Mucua.objects.get(note = mucua_note)

        except Mucua.DoesNotExist:
            print "not found: ", mucua_note
            #return Response("Mucua not found")

        if mucua:
            mucuas_list.append(mucua)

    serializer = MucuaSerializer(mucuas_list, many=True)

    return Response(serializer.data)

@api_view(['GET'])
def mucua_get_default(request):

    mucuas_list = []
    mucuas_list.append(getDefaultMucua())
    serializer = MucuaSerializer(mucuas_list, many=True)

    return Response(serializer.data)
```

A.5 mocambola

A.5.1 mocambola/models.py

```
# -*- coding: utf-8 -*-
```

```
from django.contrib.auth.models import User
from django.db import models
from django.db.models import get_model
from repository.models import Repository, gitAdd, gitCommit
from bbx.settings import REPOSITORY_DIR, MOCAMBOLA_DIR
from mocambola.serializers import UserSerializer
from django.dispatch import receiver
from django.db.models.signals import post_save
from urlparse import urlparse
import os

def getFilePath(instance):
    return os.path.join(REPOSITORY_DIR, instance.repository.getName(),
                        instance.mucua.getDescription(), MOCAMBOLA_DIR)

class Mocambola(models.Model):
    mucua = models.ForeignKey('mucua.Mucua')
    user = models.ForeignKey(User)
    repository = models.ForeignKey(Repository)

    def __unicode__(self):
        return self.user.username

    def save(self, *args, **kwargs):
        # Serializar aqui o na post_save
        return super(Mocambola, self).save(*args, **kwargs)



@receiver(post_save, sender=Mocambola)
def MocambolaPostSave(instance, **kwargs):
    """Intercepta o sinal de *post_save* do Mocambola, serialize a adiciona o objeto ao
     repositorio."""

    from bbx.utils import check_if_path_exists_or_create
    serializer = UserSerializer(instance.user)
    mocambolapath = getFilePath(instance)+'/'
    mocamboladata = instance.user.get_username() + '.json'
    check_if_path_exists_or_create(mocambolapath)
    fout = open(mocambolapath + mocamboladata, 'w')
    fout.write(str(serializer.getJSON()))
    fout.close()
    gitAdd(mocamboladata, mocambolapath)
    gitCommit(mocamboladata, instance.user.get_username(), instance.user.email, \
                  instance.repository.getPath())

# TODO HIGH: Precisa serializar o user tambem na çcriaao e update
# diretamente pelo usuario


@receiver(post_save, sender=User)
def UserPostSave(instance, **kwargs):
    """Intercepta o sinal de *post_save* do User, adiciona mocambola pegando o nome do user."""

    # TODO HIGH: conseguir importar Mucua

    mucua_model = get_model('mucua', 'Mucua')

    # TODO LOW: substituir por regexp
    current_mocambola, mucua_repository = instance.username.split("@")
    rep = urlparse('http://' + mucua_repository)
    current_mucua, current_repository, current_tld = rep.hostname.split('.')

    print "current_mocambola:", current_mocambola
    print "current_repository:", current_repository
    print "username:", instance.username

    mucua = mucua_model.objects.get(description = current_mucua)
    repository = Repository.objects.get(name = current_repository)
    mocambola, created = Mocambola.objects.get_or_create(mucua=mucua,
                          user=instance,
                          repository=repository)

    print "Created: ", created
    print "mocambola: ", mocambola
```

A.5.2 mocambola/views.py

```
# Create your views here.
```

A.6 tag

A.6.1 tag/models.py

```
# -*- coding: utf-8 -*-

"""
 Tags (etiqueta) in Bbx define some behaviours of the system, beside qualifying
 contents. Each tag can be associated to a set of policies
 (POLICIES_DIR).

 Actually policies will be linked to Django Signals selected by name. For
 example, a policy called "postSave_copyToTaina" is imported and executed
 on post save signals.

 TODO: O arquivo  na real contem a çãfuno mesmo.. ãno precisa de
       json. As tags periodicamenta ãvo ser definidas e carregada
       com as fixtures.

"""

from django.db import models
from bbx.settings import POLICIES_DIR
from bbx.utils import MultiSelectField
import json
import os
import exceptions

# This is to specify to south how to work with MultiSelectField:
# http://south.readthedocs.org/en/latest/customfields.html
from south.modelsinspector import add_introspection_rules
add_introspection_rules([], ["bbx.utils.MultiSelectField"])


class PoliciesPersistentDataUnavailable(exceptions.Exception):
    def __init__(self,args=None):
        self.args = args

def getAvailablePolicies():
    """Get a list of available policies from POLICIES_DIR."""
    policiesList = [( os.path.splitext(name)[0], os.path.splitext(name)[0]) \
                        for name in os.listdir(POLICIES_DIR) if name.endswith(".py") \
                        if not (name.startswith("__")) ]
    return policiesList

class Tag(models.Model):
    namespace = models.CharField(max_length=10, blank=True, default='')
    note = models.TextField(max_length=300, blank=True)
    name = models.CharField(max_length=26)
    policies = MultiSelectField(max_length=100, choices=getAvailablePolicies(), blank=True)

    def __unicode__(self):
        return self.namespace + ":" + self.name if self.namespace != '' else self.name

    def getId(self):
        """
        Returns tag's id built as namespace + name, ex.:
        bbx:publico ("name" attribute shoul'd be "name" FIX:
        rename!)

        """
        return self.namespace + ":" + self.name if self.namespace != '' else self.name

    def _getPoliciesFilename(self):
        """
```

```
        Policies file is built with POLICIES_DIR and tag's id

        """
        return POLICIES_DIR +'/'+ self.getId() + '.json'

    def setNamespace(self):
        """
        Sets tag's namespace like in bbx:publico gets "bbx" :)

        """
        if self.name.find(':') > 0:
            args = self.name.split(':')
            self.namespace = args[0]

    def setName(self):
        """
        Sets tag's name. FIX

        """
        if self.name.find(':') > 0:
            args = self.name.split(':')
            self.name = args[1]

    def save(self, *args, **kwargs):
        """
        Save also tag's policies to a JSON file.

        """

        self.setNamespace()
        self.setName()
        super(Tag, self).save(*args, **kwargs)

    class Meta:
        ordering = ('name',)
        unique_together = ("namespace", "name")
```

A.6.2 tag/serializers.py

```
from django.forms import widgets
from rest_framework import serializers
from tag.models import Tag

class TagSerializer(serializers.ModelSerializer):
    class Meta:
        model = Tag
        fields = ('id', 'namespace', 'note', 'name')

    def restore_object(self, attrs, instance=None):

        if instance:
            # Update existing instance
            instance.id = attrs.get('id', instance.id)
            instance.namespace = attrs.get('namespace', instance.namespace)
            instance.note = attrs.get('namespace', instance.note)
            instance.name = attrs.get('name', instance.name)

            return instance

        # Create new instance
        return Tag(**attrs)
```

A.6.3 tag/views.py

```
# Create your views here.
```