RELATÓRIO DE CUMPRIMENTO DO OBJETO ETAPA II - Ecossistemas de Software Livre - Abril 2018

Carla Silva Rocha Aguiar (Coordenadora do Projeto)

02 de Maio de 2018

Introdução

O presente relatório apresenta o acompanhamento do trabalho realizado no projeto "Ecossistemas de Software Livre", Termo de Cooperação para Descentralização de Crédito, Processo Ofício No 0646/2017/FUB-UnB, Vigência Outubro 2017 à Outubro 2019. O relatório apresentado é referente aos avanços realizados na Etapa II (Janeiro 2018 à Março 2018), de acordo com o cronograma do Plano de Trabalho.

Toda alteração no cronograma proposto foi realizada a partir de renegociação com a CGTEC do Ministério da Cultura, e tais alterações estão descritas no relatório.

FASE DE PLANEJAMENTO/EXECUÇÃO

O período de Janeiro 2018 à Março de 2018 contemplou as fases de planejamento e execução. Abaixo serão apresentados, brevemente, os principais avanços alcançados no período. Toda a documentação e acompanhamento do projeto está disponibilizado e pode ser acessado na organização do laboratório lappis-unb, e no repositório específico do projeto lappis-unb/EcossistemasSWLivre. Todo o planejamento e execução das tarefas podem ser acompanhados tanto nas *issues* quanto nas páginas *wiki*.

Abaixo serão apresentados os principais avanços alcançados no período, por pacote de trabalho (de acordo com o Plano de Trabalho). Os avanços apresentados de acordo com o pacote de trabalho e com cronograma, no período citado.

Legado em Software Livre

Os repositórios presentes na organização MinC não possuem uma padronização: muitos deles tem pouca ou nenhuma documentação, alguns nem possuem licenças de software, testes automatizados, integração contínua, métricas de qualidade de código. A pouca conformidade com os modelos seguidos por comunidades de software livre, dificulta ou limita a contribuição de interessados em coloborar com os sistemas MinC.

Muito sistemas legados carecem de testes automatizados, boa documentação e práticas de desenvolvimento contínuo, o que dificulta enormemente qualquer forma de evolução. Estes também são fatores críticos na curva de aprendizado de novos desenvolvedores e criam uma barreira para a existência de comunidades de software livre/aberto colaborando com tais sistemas. Vários projetos mantidos pelo Ministério da Cultura possuem as características acima citados.

Durante a primeira etapa do projeto foi priorizado a visão "legacy in the box" (legado em uma caixa, tradução literal), no qual o foco foi isolar alguns projetos mantidos pelo Ministério da Cultura por meio de Docker[1]. Essa solução gera o benefício de criar ambientes de desenvolvimento e produção estáveis,

[1] Docker fornece uma camada adicional de abstração e automação de virtualização de nível de sistema operacional. http://www.docker.com

fazendo com que diminua o tempo de configuração de ambiente. Essa abordagem traz um grande benefício pois possibilita o uso de práticas DevOps mesmo em sistemas legados. Esse modelo de isolar pacotes de software legados através de containers Docker possibilita um pipeline de entrega contínua, deploy continuo, e diminui a fronteira entre a equipe de infraestrutura e equipe de desenvolvimento. Já foram observados benefícios dessa abordagem, principalmente em feedback de desenvolvedores e mantenedores da infraestrutura, feito de forma espontânea. Pretende-se ainda fazer tanto uma avaliação qualitativa quanto quantitativa dessa abordagem.

Nessa segunda etapa do projeto, usamos uma segunda forma de lidar com software legado, sempre com o intuito de aplicar técnicas modernas de engenharia de software e padrões de comunidade de software livre, a fim de viabilizar o uso desses projetos legados em comunidades de software livre e em pipelines automatizados. O foco então foi transformar um software legado em software livre, a partir de técnicas de refatoração de código, e suite de testes automatizados.

Com isso, abordamos um dos objetivos do pacote que é "Pesquisa em metodologias de refatoração de sistemas legados", adotando padrões de comunidades de software livre: desde documentação técnica, quanto código de qualidade (respeitando métricas de qualidade de software), cobertura de testes, suite de testes automatizado, ferramenta de integração contínua, e pipeline de deploy contínuo. Para que pudessemos alcançar esses objetivos, foi escolhido a API do Salic como estudo de caso, uma vez que esse é um sistema relativamente pequeno, de grande relevância e impacto no ecossistema Salic. A compreensão da API do Salic também auxilia no pacote de trabalho "Aprendizado de Máquina Lei Rouanet", uma vez que grande parte do trabalho consiste em acessar e processar dados providos da API (e demanda de dados geram demandas para a evolução da mesma).

As ações programadas para esta etapa de acordo com o plano de trabalho foram:

- [x] Realizar Estudos de conteinerização
- [x] Realizar Estudo de refatoração em software legado
- [x] Realizar Estudos sobre práticas de DevOps aplicada a software legado

Grande parte do time foi alocado por dois meses nessa grande tarefa de refatorar a API do Salic, e os principais avanços alcançados nessa etapa foram:

1. Adicionada instalação automazada do ambiente de desenvolvimento através do Virtualenv[2] e do Docker, a documentação está no README.
2. A qualidade do código foi melhorada através das seguintes atividades:
   - Os SQL's em forma de textos foram refatorados, agora é utilizado o SQLAlchemy[3]. Essa refatoração melhora a manutenibilidade do código e também permite que o API do Salic funcione com qualquer banco de dados que o SQLAlchemy oferece suporte.
   - O Python utilizado no projeto foi atualizado para a versão 3 (originalmente era utilizado a versão 2 do Python).
   - Utilização do Flake8[4] para melhorar a estrutura do código.
   - Adicionado banco de dados local para o ambiente de desenvolvimento.
   - Classificação no Code Climate foi de "F" para "A", resultado da redução do débito técnico.
   - Criados testes para os endpoints da API, onde é testado se os dados das requisições são recebidos corretamente.
   - Adicionada integração, build e deploy contínuo.
   - Documentação do projeto atualizada.

A mudança da utilização de strings SQL para o código Python usando SQLAlchemy ocorreu para que além de melhorar a manutenção do código, o SQLAlchemy possui otmizações e suporte para se conectar com outros sistemas de banco de dados, por exemplo, caso o Salic passe a utilizar o PostgreSQL todo o sistema da API Salic continuará funcionando corretamente.

O Flake8 é uma ferramenta de análise estática de código que confere algumas normas que deixam o código mais legivel, padronizado e manutenível, a refatoração do código utilizando o Flake8 visou melhorar a manutenção do API adequando o código as normas do Flake8.

[2] Virtualenv é um simulador de ambientes virtuais isolados para projetos Python. http://virtualenv.pypa.io/
[3] SQLAlchemy é uma biblioteca Python de mapeamento objeto-relacional SQL. http://www.sqlalchemy.org/
[4] http://flake8.pycqa.org

Antes da refatoração não era possível levantar um ambiente de desenvolvimento, pois era necessário estar conectado ao banco de dados do Salic, porém agora, com o banco de dados local quem quiser contribuir com o projeto pode levantar o ambiente em seu próprio computador e usar um banco SQLite local, além disso, para se conectar a um banco de dados basta setar algumas variáveis de ambiente e o desenvolvedor pode conectar a um banco de dados remoto, como por exemplo um banco de dados de homologação.

Foi utilizado o Code Climate, um sistema que analisa a qualidade do código-fonte e atribui uma classificação ao projeto, essa ferramenta verifica coisas como duplicação de código informando em quais pontos estão estas duplicações.

Os testes da API foram criados para que ao realizar manutenção no código seja possivel ter uma garantia de que não foi introduzido bugs no sistema, anteriormente era difícil saber se o sistema está funcionando corretamente após o termino de uma manutenção. Também foram criados testes que comparam os resultados das requisições ao novo projeto de API refatorado com a API original que está atualmente em produção, para se ter uma garantia de que ao atualizar para a nova versão em produção os sistemas que usam a API irão continuar funcionando.

Afim de facilitar que a adição de novas features no salic-api possam chegar ao sistema em produção de forma mais rápida e prática, foi criado uma pipeline de deploy contínuo, onde é executado os testes do projeto, é checado se a build está sendo gerada corretamente e depois é feito o deploy para o servidor.

Todas as melhorias implementadas acima, fez com que o projeto da API do Salic atendesse os padrões de comunidades de software livre, além de atender os requisitos de DevOps para entrega e deploy contínuo (build de testes). Para tal, foram realizados ao total 300 commits (no qual foi aberto um pull request para o projeto no repositório do MinC). A API foi então colocado em um ambiente de homologação no laboratório Lappis, e após todos testes passarem nesse periodo de homologação, o projeto será entregue para o Ministério.

O acompanhamento do projeto realizado pode ser encontrado em https://github.com/lappis-unb/salic-api.

Catálogo de Softwares Culturais

O principal objetivo nessa etapa é exercitar em todo ciclo de projeto a experimentação e inovação contínua, de forma a subsidiar a pesquisa realizada na Etapa 5. Nesse período foram abordados dois objetivos desse pacote: “Aplicação de práticas de experimentação e inovação contínua no desenvolvimento do projeto de Catálogo de Software Culturais”, e “Transferência de conhecimento e capacitar a equipe de servidores e técnicos do MinC em práticas de gestão e desenvolvimento de software aberto, colaborativo e contínuo”.

Ações programadas para esta etapa de acordo com o plano de trabalho estão listados abaixo:

- [x] Realizar Estudos de tecnologias e práticas devops;
- [x] Realizar Estudos repositórios MinC;
- [x] Elaborar Relatório de Resultado dos Estudos;
- [x] Realizar estudos sobre funcionalidades de catálogo de software

Todas as atividades relacionadas às ações listadas acima foram 100% finalizadas.

No último item, o foco do produto foi alterado de “catálogo de software” para “Promova Cultura”. Tal mudança foi acordado com os gestores do Ministério. Apesar da mudança de foco do produto, a nova visão não altera o objetivo principal do pacote, que é a “Aplicação de práticas de experimentação e inovação contínua no desenvolvimento do projeto de Catálogo de Software Culturais”, além da execução de um ciclo completo de projeto de software.

Grande parte do objetivo de transferência de conhecimento e capacitação da equipe de servidores técnicos do MinC foi concentrado nesse período em práticas DevOps. Para tal, além de encontros técnicos para apresentação das práticas experimentadas no laboratório, alguns documentos técnicos foram elaborados para tal fim. Toda a documentação foi disponibilizada no repositório do laboratório

https://gitlab.com/lappis-unb/docs, disponibilizada também como anexo no final deste documento, os documentos cobrem tanto a primeira quanto a terceira meta do período.

Foi elaborado documentação descrevendo todo o pipeline usado para deploy contínuo no laboratório com os seguintes tutoriais:

1. GitLab CI/CD: Guia relacionado ao uso da Integração Contínua e Deploy contínuo no Gitlab;
2. Overview e exemplo básico(pt-br): Um guia que ensina como usar o gitlab CI/CD para gerar integração contínua e deploy contínuo em um projeto básico;
3. Usando Docker Compose (pt-br): Um guia que ensina como usar o GitLab CI/CD para gerar integração contínua com o Docker Compose em um projeto ágil.
4. Integrando GitLab CI/CD com projeto GitHub(pt-br): Um procedimento que possibilita o uso do GitLab CI/CD no projeto GitHub.

Toda a documentação foi realizada em português e disponibilizada para acesso.

Referente à segunda meta "Realizar Estudos repositórios MINC" nesse período foi aprofundado o estudo sobre as funcionalidades do Salic e como a execução da lei Rouanet é realizada no Salic. Foram realizadas diversas reuniões técnicas com a equipe da SEFIC, desde a equipe responsável pela adsimibilidade até a equipe responsábel pela avaliação de resultados. Os objetivos dessas reuniões foram: (a) compreender o processo (fases, etapas) da lei de Incentivo, (b) identificar os principais envolvidos/stakeholders em cada etapa, (c) levantar os principais pontos de melhoria. A partir desses levantamentos, vamos na próxima etapa, propor melhorias, ou por meio do assistente virtual (chatbot) ou por meio de algoritmos de aprendizagem de maquinas, ou por meio do sistema "Promova Cultura" ou mesmo por meio de novos requisitos para o Salic.

Referente à última meta "Realizar estudos sobre funcionalidades de catálogo de software", primeiramente o sistema denominado no plano de trabalho "Catálogo de Software" foi renomeado para "Promova Cultura" em comum acordo entre a Universidade e a CGTEC. O motivo dessa alteração foi um entendimento mútuo que desenvolver um catálogo de software não seria prioritário no momento, e um sistema mais abrangente, que pudesse expor informações relevantes da lei de incentivo agrega mais valor para o ministério. Por isso, foram realizadas duas design sprints com duração de 2 semanas cada, guiadas pela equipe de Experiência de Usuário/ Design a fim de levantar possíveis escopos do sistema "Promova Cultura". O processo foi desenvolvido pela Google Ventures, uma vertente do Google focado em testar e acelerar ideias que ainda estão em estágio inicial de desenvolvimento. Essa abordagem permite iniciar um projeto de software orientada à usabilidade.

Alguns protótipos de baixa fidelidade foram propostos. Na próxima etapa serão realizados protótipos em código de algumas das soluções propostas na design sprint, a fim de amadurecer a proposta de produto. Após esse período, em agosto está previsto a aplicação de testes de uso dos protótipos. Toda a execução da design sprint pode ser acompanhada em https://github.com/lappis-unb/PromovaCultura.

Práticas de gestão colaborativa

Nessa etapa será realizada uma pesquisa exploratória tendo como objeto de estudo os movimentos, organizações, desenvolvedores e demais stakeholders que atuam na gestão colaborativa de software aberto. O principal objetivo do trabalho de gestão colaborativa dessas comunidades de software aberto é manter um conjunto de ações de governança digital e comunicação que aproveite ao máximo esse potencial em favor das necessidades do órgão e das metas comuns às organizações parte das comunidades. Esse esforço envolve um trabalho de mapeamento de atores de cada comunidade (atuais e potenciais futuros), assessoria para planejamento conjunto, facilitação de fluxos de comunicação e mobilização, realização de atividades conjuntas para integração, identificação de oportunidades externas, assessoria para comunicação e divulgação ao público externo à comunidade e apoio para solução de conflitos.

Ações programadas para esta etapa de acordo com o plano de trabalho:

- [x] Realizar Estudos sobre processo de planejamento conjunto
- [x] Identificar grupos de opinião

Todas as atividades relacionadas as ações listadas acima foram 100% finalizada. Nessa etapa, foi focado na estratégia de colaboração entre os laboratórios de pesquisa que contribuem para os repositórios MinC:

1. Frente metodologias ágeis e devops: Por ser uma prática comum ao LAPPIS, oriunda da própria história do laboratório e reforçada pelas disciplinas práticas da FGA/UnB, é um ativo que pode ser compartilhado na comunidade de laboratórios;
    - Oficinas para intercâmbio de método de trabalho (funcionamento das sprints, wikis, decisão sobre pull requests etc) - Maio ou junho/2018
    - Evento prático de devops para apresentar o resultado da pesquisa e fazer intercâmbio prático.
2. Frente de tecnologias livres para chatbots: Pela carência de tecnologias livres desse tipo e as inúmeras aplicações na qualificação dos serviços públicos essa pode ser uma oportunidade para os outros laboratórios. O Lappis pode auxiliar a incorporação dessa tecnologia nos serviços digitais em desenvolvimento pelos outros laboratórios, assim como já está fazendo com o Salic e pode fazer com o Mapas Culturais.
    - Lançamento da tecnologia no MinC e Workshop técnico na semana seguinte para os interessados em conhecer/colaborar na tecnologia;
    - Incorporação de metas estratégicas conjuntas para devops e chatbots em outros serviços digitais do MinC;
    - Implementação de boas práticas para governança da comunidade em torno desse ativo tecnológico (chatbots) visando aumento da contribuição de desenvolvedores externos ao Lappis, com foco nos times dos laboratórios parceiros.
3. Frente de Governança de Comunidades: Essa frente envolve pesquisa e realização de eventos conjuntos com temas estratégicos para a colaboração aberta nas tecnologias desenvolvidas e mantidas pelo Estado.

Intercâmbio de pesquisa através do compartilhamento de referências e produção conjunta de artigos - Seminário Interno com UFG - Jul/2018 à Out/2018.

Realização de eventos conjuntos sobre o tema da Governança de Comunidades Open Source com adesão do Estado e Contratação Pública de TICs - Maio ou junho/2018.

Aprendizado de Máquina Lei Rouanet

O principal objetivo é o estudo de técnicas de Aprendizado de Máquina que possam apoiar o sistema de recomendação e fiscalização da lei Rouanet. Nessa etapa será realizada uma pesquisa exploratória em técnicas de aprendizado de máquina e processamento de linguagem natural. Tais técnicas e algoritmos serão aplicados para melhorar a experiência de usuário (UX) por meio da proposta de chatbots como interface entre os proponentes na lei Rouanet e o Ministério da Cultura.

Além disso, técnicas de aprendizado de máquinas serão estudadas para automatizar processos nas trilhas de auditorias, tanto na etapa de habilitação e aprovação, quanto na etapa de prestação de contas. O objetivo é auxiliar auditores a encontrar erros, inconsistências e detectar anomalias nas submissões.

Ações programadas para esta etapa de acordo com o plano de trabalho:

- [x] Realizar Estudo Lei Rouanet/SALIC
- [x] Realizar Estudo de aprendizado de máquina
- [x] Realizar Estudo processamento linguagem natural
- [x] Realizar Estudos de chatbots

Todas as atividades relacionadas as ações listadas acima foram 100% finalizadas. Segue resumo da execução das atividades:

Foi desenvolvido uma versão inicial do bot – versão 0.1 (beta) – com o framework Hubot Natural[5], o desenvolvimento aconteceu após estudos sobre ferramentas para criação de chatbots. Decidiu-se utilizar o Rocket.Chat como interface para o chatbot, compondo a solução em conjunto com o Hubot Natural.

[5]Hubot Natural é um chatbot de Processamento de Linguagem Natural para o Rocket.Chat. https://github.com/RocketChat/hubot-natural

Realizou-se evolução do projeto Hubot Natural, com contribuições da equipe ao repositório oficial do projeto. Além de colaboração com os desenvolvedores core do projeto Rocket.Chat para avaliação do melhor caminho para futuras evoluções.

Esta primeira versão foi treinada com uma base de conhecimento criada a partir de documentos disponibilizados pela ouvidoria da SEFIC, importante destacar que neste primeiro treinamento foi incluido especialmente conhecimentos avançados sobre a lei de incentivo, deixando de fora da base conhecimento básicos necessários para responder adequadamente questões mais básicas.

Levantou-se um ambiente de homologação em https://rouana.lappis.rocks, incluindo uma landing page da Rouana com instruções de como validar e homologar o assistente virtual, onde através da base de conhecimento criada a partir dos documentos disponibilizados pela ouvidoria da SEFIC, avaliou-se a eficácia do chatbot através de testes de usuários incluindo servidores do MinC, pesquisadores e alunos do Lappis.

Os testes realizados com chatbot versão 0.1 (beta) em ambiente de homologação revelaram que o assistente virtual com as tecnologias selecionadas atende perfeitamente as necessidades do projeto, indicando que o caminho trilhado até o momento está em sintonia com a missão final de proporcionar um novo canal aos cidadãos para compreender e tirar dúvidas sobre a lei Rouanet.

Os dados coletados e feedback dos usuários durante a fase de homologação serão utilizados para direcionar a evolução e melhorias, identificou-se inicialmente que a base de conhecimento necessita de evolução, especialmente com questões mais simples.

Contribuimos com a documentação do repositório do Hubot Natural, incluindo documentar o processo de configuração do LiveTransfer, tradução da documentação do Hubot Natural para o inglês e adoção de solução de documentação para o Hubot Natural. Foi feito também levantamento de práticas e ferramentas para instrumentalização com ferramentas para análise estática como Coffeelint e Codeclimate, além de integração contínua ao projeto.

Realizou-se também pesquisa e implementação de melhores práticas de UX para interfaces conversacionais, necessária para melhoria na experiência do usuário ao utilizar o assistente virtual da lei Rouanet.

Em paralelo a todo este trabalho, estudou-se tecnologias para criação de uma nova versão do bot, incluindo frameworks para criação de chatbots mais inteligentes, exemplos: Rasa, AIVA, Botpress, IBM blue mix, Seq2seq, Hubot-playbook e Neo4j. Estes frameworks foram avaliados na prática e algumas tecnologias foram analisadas em detalhes, como: Rasa-NLU + BotPress + RocketChat e Rasa-core + Rasa-nlu.

A implementação da nova versão do bot foi iniciada em paralelo ao desenvolvimento da versão 0.1 (beta) citada anteriormente, já utilizando uma abordagem mais poderosa de desenvolvimento de bots; escolha de mudança de arquitetura e tecnologias a serem usadas para a próxima versão do chat.

Em complemento ao desenvolvimento do chatbot realizamos estudos para compreensão do processo de projetos incentivados via Lei Rouanet, incluindo estudo de tecnologias de aprendizado de máquina a fim de auxiliar o processo de admissão e prestação de contas do Salic.

Neste sentido, iniciou-se estudos e testes de algoritmos para detecção de anomalias em itens das planilhas orçamentárias de projetos submetidos ao Salic, utilizando técnicas de aprendizado de máquina, tanto na extração de características relevantes para o problema (*Exploratory Data Analysis* e *Data Wrangling*), quanto na classificação de novos dados (usando modelos básicos de regressão do módulo *Scikit-learn*).

São dois os objetivos dessa frente de trabalho:

1. Auxiliar o processo de admissão e prestação de contas do Salic: automatizar tarefas simples e repetitivas de tais processos para otimizar da criação à conclusão de projetos culturais;
2. Fornecer insumos para um sistema de transparência do Salic: fornecer métricas utilizadas para mapear as categorias e regiões de maior incentivo e para incentivar novos produtores culturais.

A frente está trabalhando na criação de uma API que deve se comunicar, a princípio, com o Salic. Contudo, futuramente novos sistemas também podem realizar requisições à API para extrair métricas sobre projetos culturais.

O desenvolvimento desta frente está sendo feito com o levantamento de hipóteses e evolução da API. A metodologia utilizada é a *Hypothesis-Driven Development*, focada em criação e validação contínua de hipóteses de aprendizado de máquina, seguida de implementação na API das hipóteses confirmadas na etapa de validação.

A API está em desenvolvimento em Python, utilizando-se o framework Django. Três hipóteses já foram levantadas e estão sendo validadas:

1. Relação entre o tempo e a mudança dos preços de itens da planilha orçamentária de um projeto;
2. Identificação de itens superfaturados a partir do histórico de projetos aprovados e recusados e;
3. Categorização e identificação de similaridade de um projeto a partir de sua planilha orçamentária vigente.

Caso as hipóteses se confirmem, serão implementadas e será possível verificar, para cada projeto, se sua planilha orçamentária contém itens possivelmente superfaturados e quais os projetos mais similares com o projeto em questão.

Aferição e aceitação de produtos de software

O objetivo geral desta frente de pesquisa é auxiliar os times de desenvolvimento e gestores de TI do MinC a aprimorarem sua capacidade em tomar decisões acerca da qualidade das versões dos produtos de software entregues por seus fornecedores.

Ações programadas para esta etapa de acordo com o plano de trabalho:

- [x] Revisão da área
- [x] Diagnóstico sobre as práticas atualmente adotadas pelo MinC de garantia da qualidade de produto
- [ ] Elaborar Plano de Pesquisa-Ação

Nessa etapa foram aplicadas surveys com os gestores do MinC e desenvolvedores seniores do Lappis e MinC. O objetivo do survey foi fazer uma análise qualitativa sobre o projeto e sobre práticas devops e práticas de comunidades de software livre.

Nesse período também foi realizado a análise do survey aplicado aos alunos. O resultado é apresentado em anexo.

Acompanhamento Financeiro

O valor do repasse referente à Etapa II foi de R$202.600,00. Todo esse repasse foi na rubrica 30.90.20, referente à auxílio Financeiro a Pesquisa (Bolsas). Desse repasse, um total de R$190.635,90 foi executado na Etapa II, representando na prática que o orçamento foi consumido apenas na categoria mão-de-obra. Todo esse valor foi executado no pagamento das bolsas do time, e o valor gasto por frente do projeto pode ser visto na figura abaixo.

Assinatura

Responsável pela Execução:

Nome: Carla Silva Rocha Aguiar (Coordenadora do Projeto)

Assinatura:

Data: 06/04/2018

Valores Executados – Entrega 2 – Período: Janeiro a Março/2018

Descrição		Valor Executado	Total Parcial	Total Executado
Mão de Obra	Catálogo de Softwares Culturais	R$ 65.700,00	R$ 170.000,00	R$ 190.635,90
	Legado em Software Livre	R$ 54.600,00		
	Gestão de Práticas Colaborativas	R$ 0,00		
	Aprendizado de Máquina Lei Rouanet	R$ 49.700,00		
Outras Despesas	Pessoa Jurídica		R$ 20.635,90	
	Material de Consumo			
	Material Permanente			
	Despesas Operacionais CDT	R$ 20.635,90		

Mão de Obra	R$ 170.000,00
Despesas Op CDT	R$ 20.635,90

Figure 1: Detalhamento da execução do repasse na Etapa II.

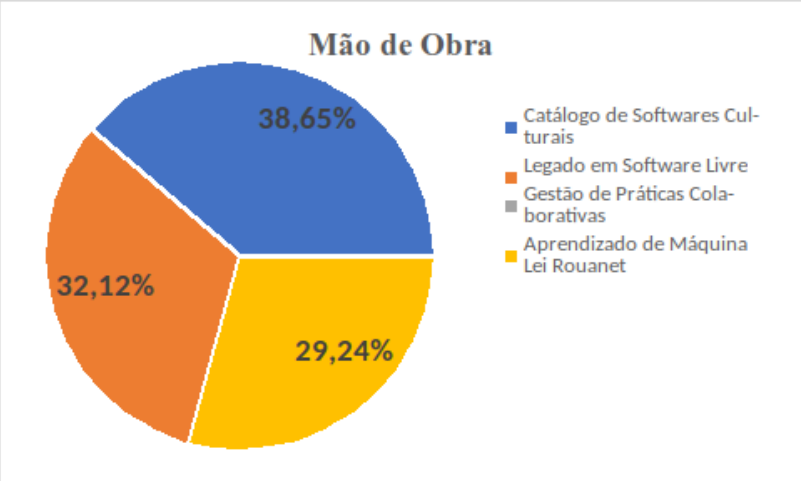

Figure 2: Neste gráfico é possível observar a representação do percentual do custo da mão-de-obra incidido em cada equipe do projeto. A maior alocação de recursos encontram-se nas equipes do Catálogo de Softwares Culturais(representado pela cor azul), uma vez que grande parte das funcionalidades desenvolvidas são providas através desta frente, e a equipe do Aprendizado de máquina(representado pela cor verde), que desenvolveu o chatbot.

Anexo I - GitLab CI/CD

Este *doc* tem por objetivo capacitar um *dev* em utilizar o **GitLab CI/CD** em projetos que exigem estruturas básicas de configuração. Para um melhor aproveitamento deste *doc* é recomendável ter realizado com completude o guia básico.

Introdução

Docker Compose é uma ferramenta para definição e execução de aplicações de múltiplos *containers* **Docker**. Através de um arquivo de configuração YAML é possível definir os serviços da aplicação e suas interações. Esse arquivo é utilizado como entrada em um CLI capaz de iniciar os serviços configurados em um simples comando.

Enquanto o **Docker** permite a definição e o gerenciamento de um único *container*, **Docker Compose** define e gerencia múltiplos *containers* e suas interações.

Dentre os principais benefícios, incluem:

- Facilidade de definição dos serviços;
- Uma vez definido o arquivo de configuração, o uso de simples comandos inicia a aplicação e todos os seus serviços (*containers*), incluindo suas interações;
- Ideal para desenvolvedores configurarem o ambiente local;
- É uma das camadas de configuração em orquestradores de *containers* como Kubernetes e Cattle (Orquestrador do Rancher)

Se o projeto da aplicação que estiver desenvolvendo utiliza **Docker Compose** para definição do ambiente em nível de teste, desenvolvimento e/ou produção, utilizar **Docker Compose** na integração contínua é uma opção.

Utilizando Docker Compose no GitLab CI/CD

Para exemplificar o uso do **Docker Compose** no **GitLab CI/CD**, foi criado um simples respositório chamado *characters* consolidando o uso das duas ferramentas no estágio de teste. Ao fim da leitura deste exemplo você será capaz de reproduzir o uso do **Docker Compose** no **CI/CD** do seu projeto.

Projeto Modelo

O sistema *characters* é um pequeno projeto Phoenix com banco de dados em PostgreSQL que define uma API RESTful de uma única entidade chamada `Character`, conforme descrito no README do projeto:

```
// curl -X GET -H "Accept: application/json" {HOST}:{PORT}/api/v1/characters/1
"data": {
  "id": 1,
  "first_name": "Jon",
  "last_name": "Snow",
  "age": 14,
  "origin": "A Song of Ice and Fire"
}
```

Rotas da API

As rotas, conforme especificado no comando `mix phx.routes`, são:

```
// Apresenta todos os Characters
Rota: "GET /api/v1/characters",
Modelo de cURL: `curl -X GET -H "Accept: application/json" {host}:{port}/api/v1/characters`
Exemplo de resultado: {
  "data": [
    {
      "id": 1,
      "first_name": "Jon",
      "last_name": "Snow",
      "age": 14,
      "origin": "A Song of Ice and Fire"
    }, {
      "id": 2,
      "first_name": "Walter",
      "last_name": "White",
      "age": 50,
      "origin": "Breaking Bad"
    }, {
      "id": 3,
      "first_name": "Locke",
      "last_name": "Cole",
      "age": 25,
      "origin": "Final Fantasy VI"
    }, {
      "id": 4,
      "first_name": "Arthas",
      "last_name": "Menethil",
      "age": 24,
      "origin": "Warcraft III"
    }, {
      "id": 5,
      "first_name": "Dominick",
      "last_name": "Cobb",
      "age": 37,
      "origin": "Inception"
    }, {
      "id": 6,
      "first_name": "Vincent",
      "last_name": "Vega",
      "age": 27,
      "origin": "Pulp Fiction"
    }
  ]
}

// Apresenta o Character de id com o valor {:id}
Rota: "GET /api/v1/characters/:id"
Modelo de cURL: `curl -X GET -H "Accept: application/json" {host}:{port}/api/v1/characters/{:id}`
Exemplo de resultado: {
  "data": {
    "id": 1,
    "first_name": "Jon",
    "last_name": "Snow",
    "age": 14,
    "origin": "A Song of Ice and Fire"
  }
```

```
}

// Cria um novo Character
Rota: "POST /api/v1/characters"
Modelo de cURL: `curl -X POST -H "Accept: application/json" -H "Content-Type: application/json" -d '{
Exemplo de dado: {
  "character": {
    "first_name": "Jon",
    "last_name": "Snow",
    "age": 14,
    "origin": "A Song of Ice and Fire"
  }
}
Exemplo de resultado: {
  "data": {
    "id": 1,
    "first_name": "Jon",
    "last_name": "Snow",
    "age": 14,
    "origin": "A Song of Ice and Fire"
  }
}

// Atualiza parcialmente o Character de id com o valor {:id}
Rota: "PATCH /api/v1/characters/:id"
Modelo de cURL: `curl -X PATCH -H "Accept: application/json" -H "Content-Type: application/json" -d '
Exemplo de dado: {
  "character": {
    "last_name": "Stark"
  }
}
Exemplo de resultado: {
  "data": {
    "id": 1,
    "first_name": "Jon",
    "last_name": "Stark",
    "age": 14,
    "origin": "A Song of Ice and Fire"
  }
}

// Substitui o Character de id com o valor {:id}
Rota: "PUT /api/v1/characters/:id"
Modelo de cURL: `curl -X PUT -H "Accept: application/json" -H "Content-Type: application/json" -d '{"
Exemplo de dado: {
  "character": {
    "first_name": "João",
    "last_name": "das Neves",
    "age": 15,
    "origin": "As Crônicas de Gelo e Fogo"
  }
}
Exemplo de resultado: {
  "data": {
    "id": 1,
    "first_name": "João",
```

```
    "last_name": "das Neves",
    "age": 15,
    "origin": "As Crônicas de Gelo e Fogo"
  }
}

// Remove o Character de id com o valor {:id}
Rota: "DELETE /api/v1/characters/:id"
Modelo de cURL: `curl -X DELETE -H "Accept: application/json" {host}:{port}/api/v1/characters/{:id}`
```

Configuração do Docker Compose

Convencionalmente, projetos que utilizam **Docker Compose** mantêm 3 arquivos de configuração:

1. `docker-compose.test.yml` (ou `test.yml`): Configura a aplicação para seu ambiente de teste. Utilizada pelos *devs* para testes isolados localmente ou em ferramentas de integração contínua que suportam **Docker Compose**;
2. `docker-compose.dev.yml` (ou `local.yml`): Configura a aplicação para seu ambiente de desenvolvimento. Utilizada apenas pelos *devs* para uso do sistema localmente;
3. `docker-compose.prod.yml` (ou `production.yml`): Configura a aplicação para seu ambiente de produção.

Cada tipo de configuração pode exigir *containers*, variáveis de ambiente e comandos diferentes. Portanto, é comum existir uma pasta `compose` na raíz do projeto com as configurações de cada ambiente.

Executando o Projeto Localmente

O arquivo de configuração `docker-compose.dev.yml` define os serviços **api** e **db**, como pode ser visto a seguir:

```
version: '3.3'

services:
  api:
    container_name: characters-api-dev
    build:
      context: .
      dockerfile: ./compose/dev/api/Dockerfile
    depends_on:
      - db
    env_file:
      - ./compose/dev/db.env
      - ./compose/dev/api.env
    ports:
      - 4000:4000
    volumes:
      - ./api:/code

  db:
    container_name: characters-db-dev
    env_file:
      - ./compose/dev/db.env
    image: postgres
    volumes:
      - ./postgres/dev/data:/var/lib/postgresql/data
```

O arquivo `./compose/dev/api/Dockerfile` define o *container* do serviço **api**, como pode ser visto a seguir:

```
FROM elixir

RUN mix local.hex --force && \
    mix local.rebar --force && \
    mix archive.install --force \
    https://github.com/phoenixframework/archives/raw/master/phx_new.ez

COPY ./compose/dev/api/entrypoint.sh /entrypoint.sh
COPY ./compose/dev/api/start.sh /start.sh
COPY ./api /code

WORKDIR /code

EXPOSE 4000

ENTRYPOINT ["/entrypoint.sh"]

CMD ["/start.sh"]
```

E os respectivos *scripts* `entrypoint.sh` e `start.sh`:

```
#!/usr/bin/env bash

cmd="$@"

printf "\n## Mix Version\n\n"
mix -v
mix phx.new -v

printf "\n## Updating Dependencies\n\n"
mix deps.get
mix deps.compile

printf "\n## Creating Database\n\n"
mix ecto.create
mix ecto.migrate

exec $cmd

printf "\n## Initializing API\n\n"
mix phx.server
```

Por fim, os arquivos `api.env` e `db.env` contendo as variáveis de ambiente dos serviços:

```
MIX_ENV=dev
POSTGRES_HOST=db

POSTGRES_USER=characters_dev
POSTGRES_PASSWORD=characters_dev
```

A API ficará disponível na porta `4000` de seu `localhost` e os dados do **PostgreSQL** ficarão armazenados em `./postgres/dev/data`.

Para iniciar os serviços da API em modo de desenvolvimento, execute:

```
docker-compose -f docker-compose.dev.yml up
```

Para acessar a **API**, utilize o *browser* para as rotas `GET` ou qualquer outro programa que possa definir e executar **REST**. Por exemplo:

```
curl -X GET -H "Accept: application/json" localhost:4000/api/v1/characters
```

Irá listar todos os *characters* definidos. Como o banco de dados está vazio, a **API** irá retornar o seguinte **json**:


```
{
  "data": []
}
```


Para semear o banco com dados de exemplo, execute:

```
docker-compose -f docker-compose.dev.yml exec api mix run priv/repo/seeds.exs
```

Ao executar novamente o comando para listar os *characters*, a **API** irá retornar o seguinte **json**:


```
{
  "data": [{
    "origin": "A Song of Ice and Fire",
    "last_name": "Snow",
    "id": 1,
    "first_name": "Jon",
    "age": 14
  }, {
    "origin": "Breaking Bad",
    "last_name": "White",
    "id": 2,
    "first_name": "Walter",
    "age": 50
  }, {
    "origin": "Final Fantasy VI",
    "last_name": "Cole",
    "id": 3,
    "first_name": "Locke",
    "age": 25
  }, {
    "origin": "Warcraft III",
    "last_name": "Menethil",
    "id": 4,
    "first_name": "Arthas",
    "age": 24
  }, {
    "origin": "Inception",
    "last_name": "Cobb",
    "id": 5,
    "first_name": "Dominick",
    "age": 37
  }, {
    "origin": "Pulp Fiction",
    "last_name": "Vega",
    "id": 6,
    "first_name": "Vincent",
    "age": 27
  }]
}
```


Para desativar a aplicação e seus serviços, execute:

```
docker-compose -f docker-compose.dev.yml down
```

Para remover os volumes e as imagens locais geradas, execute o comando com as seguintes flags adicionais:

```
docker-compose -f docker-compose.dev.yml down --rmi local -v
```

Executando a Aplicação em Ambiente de Teste

O arquivo de configuração `docker-compose.test.yml` define os serviços **api** e **db** em ambiente de teste, como pode ser visto a seguir:

```
version: '3.3'

services:
  api:
    container_name: characters-api-test
    build:
      context: .
      dockerfile: ./compose/test/api/Dockerfile
    depends_on:
      - db
    env_file:
      - ./compose/test/db.env
      - ./compose/test/api.env
    volumes:
      - ./api:/code

  db:
    container_name: characters-db-test
    env_file:
      - ./compose/test/db.env
    image: postgres
    volumes:
      - ./postgres/test/data:/var/lib/postgresql/data
```

As diferenças entre o arquivo de teste e o de desenvolvimento são:

- Os nomes dos *containers* estão sinalizadas como *test*;
- A referência do **Dockerfile** é o de teste;
- As variáveis de ambiente são as de teste;
- O diretório do **PostgreSQL** é o de teste.

O arquivo `./compose/test/api/Dockerfile` define o *container* do serviço **api**, como pode ser visto a seguir:

```
FROM elixir

RUN mix local.hex --force && \
    mix local.rebar --force && \
    mix archive.install --force \
    https://github.com/phoenixframework/archives/raw/master/phx_new.ez

COPY ./compose/test/api/entrypoint.sh /entrypoint.sh
COPY ./compose/test/api/test.sh /test.sh
COPY ./api /code

WORKDIR /code

EXPOSE 4000

ENTRYPOINT ["/entrypoint.sh"]
```

```
CMD ["/test.sh"]
```

As diferenças entre o **Dockerfile** de teste e o de desenvolvimento são:

- O caminho do *entrypoint* é o de teste;
- O *script* de execução é o de teste.

E os respectivos *scripts* `entrypoint.sh` e `test.sh`:

```
#!/usr/bin/env bash

cmd="$@"

printf "\n## Mix Version\n\n"
mix -v
mix phx.new -v

printf "\n## Updating Dependencies\n\n"
mix deps.get

exec $cmd
```

```
#!/usr/bin/env bash

printf "\n## Performing Tests\n\n"
mix test
```

As diferenças entre os *scripts* de teste e o de desenvolvimento são:

- O *entrypoint* não compila as dependências;
- O *entrypoint* não cria o banco de dados e nem migra;
- O comando roda a suíte de testes ao invés de iniciar o servidor da **API**.

Por fim, os arquivos `api.env` e `db.env` contendo as variáveis de ambiente dos serviços:

```
MIX_ENV=test
POSTGRES_HOST=db
```

```
POSTGRES_USER=characters_test
POSTGRES_PASSWORD=characters_test
```

A API não ficará disponível na porta `4000` de seu `localhost`, pois o arquivo de configuração não faz a configuração de portas. Para executar a aplicação em ambiente de teste, utilize o seguinte comando:

```
docker-compose -f docker-compose.test.yml run --rm api
```

Docker Compose irá inicializar o serviço **api** e todos os serviços associados à ele (no caso: **db**) e executará o comando padrão da imagem (`mix test`). Os resultados dos testes devem ser:

```
## Performing Tests

.................

Finished in 0.1 seconds
17 tests, 0 failures
```

Para remover os volumes e as imagens locais geradas, execute o comando:

```
docker-compose -f docker-compose.dev.yml down --rmi -v
```

Configuração do GitLab CI/CD

O arquivo de configuração **GitLab CI/CD** (`.gitlab-ci.yml`) do projeto é definido:

```yaml
image: docker
services:
  - docker:dind

stages:
  - test
  - update-registry

variables:
  STAGING_IMAGE: $CI_REGISTRY_IMAGE:staging
  LATEST_IMAGE: $CI_REGISTRY_IMAGE:latest

test:
  stage: test
  before_script:
    - apk add --no-cache py-pip
    - pip install docker-compose
  script:
    - docker-compose -f docker-compose.test.yml run --rm api

push staging image:
  stage: update-registry
  script:
    - docker login -u "gitlab-ci-token" -p "$CI_JOB_TOKEN" $CI_REGISTRY
    - docker build -f compose/dev/api/Dockerfile -t $STAGING_IMAGE .
    - docker push $STAGING_IMAGE
  only:
    - /develop/
  tags:
    - docker

push latest image:
  stage: update-registry
  script:
    - docker login -u "gitlab-ci-token" -p "$CI_JOB_TOKEN" $CI_REGISTRY
    - docker build -f compose/prod/api/Dockerfile -t $LATEST_IMAGE .
    - docker push $LATEST_IMAGE
  only:
    - /master/
  tags:
    - docker
```

O *job* em destaque para este guia é o `test`.

Sua imagem `docker` é herdada da configuração raíz e o serviço `docker:dind` (**dind** significa *docker in docker*) permite a utilização do CLI do **Docker**.

A configuração definida em `before_script` adiciona pip e instala o **Docker Compose**.

A configuração definida em `script`, por fim, executa as configurações do serviço `api` definida no arquivo `docker-compose.test.yml`.

As pipelines executadas no projeto podem ser vistas nos seguintes *links*:

- Primeiro pipeline da *branch* test;
- Segundo pipeline da *branch* test;
- Pipeline da *branch* develop;
- Pipeline da *branch* master.

Anexo II - Estudo sobre ferramenta de chatbots

Introdução

A fim de identificar as ferramentas mais adequadas ao contexto do MINC, foram analisadas as seguintes ferramentas:

- Botpress
- RASA
- IBM Bluemix

Levando em conta os seguintes critérios:

- Vantagens e Limitações
- Licença
- Analytics (indicadores e dados coletados sobre as interações com o bot)

Botpress

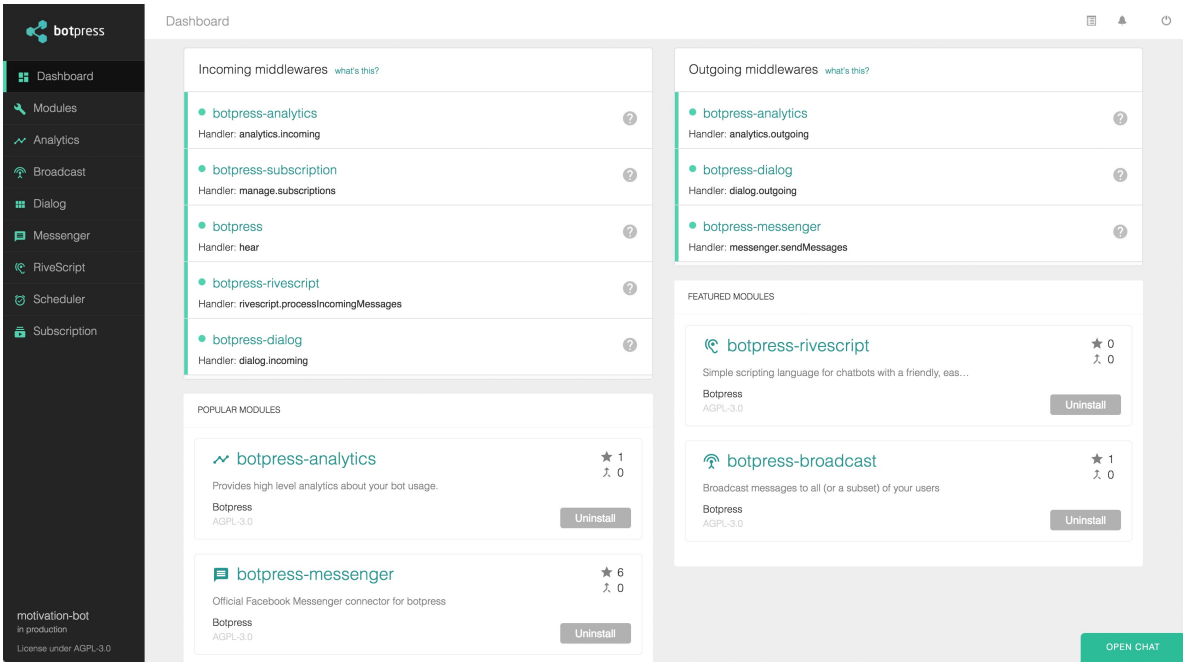

Figure 3: botpress

Vantagens

- Permite que o bot faça perguntas ao usuário
- Permite que sejam enviadas opções para que o usuário selecione
- Possui ferramenta de análise
- Possibilita integração externa

Limitações

- Não há NLP (Natural Language Processing) nativo, entretanto é fácil a integração com módulos com NLP, como RASA, API.AI (Dialogflow), WIT.AI, entre outros.

- Apesar de possibilitar integração externa, é necessária a criação de um conector. Há conectores tanto oficiais, quanto criados pela comunidade. Já existe um conector para páginas web criado pela comunidade, porém em versão alpha, ainda não contemplando o envio de imagens, vídeos, áudios e etc.

Licença

O bot utiliza duas licenças: AGPLv3 e a Botpress Proprietary License.

A Botpress Proprietary License permite que os módulos sejam usados apenas para a construção de chatbots, não podendo ser utilizados para construção de outras plataformas comerciais, como uma própria plataforma de desenvolvimento de chatbots, sem permissão prévia da Botpress.

O core e os módulos utilizados em um projeto devem utilizar a mesma licensa.

Para mais detalhes, acesse o FAQ.

Analytics

O botpress oferece uma dashborad onde é possível visualizar as seguintes informações:

- Número total de usuários por plataforma
- Uso por gênero
- Quantidade de usuários ativos nas últimas duas semanas
- Média de interações que os usuários têm com o bot em um dia
- Retenção por dia
- Horas em que o bot esteve ocupado para os últimos sete dias

Exemplos

Integração com páginas web

Para integração do bot com páginas web é necessária a instalação do módulo botpress-web através do seguinte comando:

```
npm install botpress-web@next
```

Inclua o código abaixo dentro da tag `<body>` da página a qual você quer integrar o chat:

```
<script>
    window.botpressSettings = {
        hostname: "localhost:3000" // <<-- Mude para o hostname do seu bot
    };
</script>
<script>
    !function() {
        function t() {
        var t = n.createElement("script");
        t.type="text/javascript", t.async=!0,t.src="http://"+a.hostname+"/api/botpress-web/inject.js'
        var e = n.getElementsByTagName("script")[0];
        e.parentNode.insertBefore(t,e)
        }

    var e = window, a = e.botpressSettings,n = document;
    e.attachEvent ? e.attachEvent("onload",t): e.addEventListener("load",t,!1)
    }();
</script>
```

Se a URL do host não for https, dentro do diretório do seu projeto, vá até o diretório **node_modules/botpress-web/bin** e no arquivo **node.bundle.js** troque a URL da definição do host para http.

Reinicie a aplicação do bot e a janela de chat deverá aparecer na página web.

Integração com wit.ai

Para adicionar NLP ao bot é necessária a integração com algum módulo que forneça essas funcionalidades. A seguir será mostrado um exemplo de integração com o WIT.AI.

Primeiramente é necessária a instalação do módulo através do seguinte módulo:

```
botpress install wit
```

Crie uma aplicação no WIT.AI e cadastre os contextos com suas respectivas questões, conforme este este tutorial.

Copie o **Service Access Token** na aba **Settings** do WIT.AI e cole no campo **Access Token** da opção Wit.ai no Botpress, marcando a opção Understanding.

[[images/witi-set.png]]

Neste exemplo, no arquivo content.yml foram configuradas as respostas para cada contexto definido no WIT.AI:

```
projetos:
  - typing: true
    text: |
      Para a Pessoa Física e para o Empresário Individual - EI com enquadramento em Microempresário I
      Para os demais enquadramentos de Empresário Individual - EI o valor máximo é de R$ 5.000.000,00
      Para a Empresa Individual de Responsabilidade Limitada - EIRELI, Sociedades Limitadas - Ltda. e

ingressos:
  - typing: true
    text: |
      A comercialização dos ingressos ou outros produtos culturais resultantes do projeto não possui
      O preço médio do produto cultural a ser vendido a critério do proponente referente à cota de 50
      Rege a matéria o art. 53 da IN 01/17, em especial a alínea "e" do inciso I e os §s 4º e 5º.
```

E por fim, foi feita a configuração para que quando o WIT.AI retornar que a frase feita pelo usuário representa um contexto, o bot identifique e retorne as respostas definidas para o contexto retornado:

```
module.exports = function(bp) {

  bp.hear({'wit.entities.intent[0].value': 'ingressos'}, (event, next) => {
    event.reply('#ingressos')
  })

  bp.hear({'wit.entities.intent[0].value': 'projetos'}, (event, next) => {
    event.reply('#projetos')
  })
}
```

Teste o bot:

[[images/bot.png]]

Há ainda módulos prontos para a integração com outras ferramentas, além do Botpress NLU que permite a integração com o RASA ou LUIS de forma que possam ser configurados pela própria interface do Botpress:

botpress nlu demo

IBM Watson Conversation

O watson conversation é a solução da IBM para chatbots. Há um curso online que dá uma breve introdução sobre os chatbots e o IBM watson

O fluxo de trabalho no Watson é:

1. Criar as `intents`: que são intenções que o usuário possui. Cada intenção é formada por um conjunto de perguntas / frases, no mínimo 5, e um título identificador da intenção. Nesta etapa é bom prever muitas variações das perguntas e possíveis erros gramaticais que o usuário pode cometer para melhorar a acuracia do treinamento.
2. Criar as `entities`: repsentam partes elementares de uma mensagem, por exemplo, um local, uma data ou um tipo de relação pessoal. Cada entidade possui um texto que representa ela, e vários sinônimos.
3. Criar o diálogo: O diálogo é um fluxo de conversa com os passos em que ocorre um fluxo natural de conversa real. Ele é descrito como uma sequencia de estatos, e cada estado possui: intent ou ação, uma série de respostas para mostrar pro usuário, uma ação que deve ser tomada ao fim daquele estado. A ação do fim pode ser pular para um estado específico, pedir input para o o usuário, entre outras. Cada estado pode ter sub-estados.

Intents

[[images/intents-watson.png]]

Entities

[[images/entities-watson.png]]

Dialog

[[images/dialog-watson.png]]

Vantagens

- Tem integração por padrão com o Slack e o Facebook Messenger
- É possível integrar com outras aplicações
- Inteface simples e agradável
- Suporte a analytics, com poucas métricas

Limitações

- Reconhecimento semi-automatico de entidades. Você precisa pré-cadastrar as opções
- Código fechado e precificação por uso

Licença

Copyright - código fechado da IBM.

O Pricing é baseado no número de conversas feitas por mês.

Analytics

O IBM Watson extrai as seginte métricas:

- Gráfico de número de chats por hora
- Número total de chats
- Número de mensagens sem classificação (não identificou intenção ou entidade)
- Top 3 intenções e entidades

Rasa NLU e Rasa Core

Rasa NLU

O Rasa NLU é uma ferramenta open source para processamento de linguagem natural, sendo focada em classificação de intenções e extração de identidades. Rasa é um conjunto de API's para construção de um parser que utiliza as bibliotecas de NLP e ML existentes. Sendo que, este pode ser utilizado como uma alternativa à ferramentas como: `wit`,`LUIS`,`Dialog Flow`, e etc.

Segundo o estudo Evaluating Natural Language Understanding Services for Conversational Question Answering Systems, é possível perceber que o Rasa possui um desempenho muito bom comparado às principais ferramentas comerciais de processamento de linguagem natural, tanto em relação à desempenho quanto à acurácia da extração de entidades e intenções.

Utilizando o Rasa NLU

Para utilizar o Rasa basta instalar o pacote do Rasa NLU, escolher o modelo de backend que melhor se aplica ao contexto em questão, e seguir o tutorial de instalação. Como o Rasa utiliza bibliotecas python de ML para processamento, tanto o rasa_nlu quanto o backend podem ser instalados utilizando o `pip`.

Os dados para treinamento do Rasa NLU seguem um padrão que consiste na entrada de texto a ser processada, definição da intenção correspondente àquela entrada e das entidades presentes. Os arquivos de treinamento ficam no diretório `data`.

Para selecionar o pipeline de backend a ser utilizado, para isso o Rasa NLU utiliza um arquivo `JSON` que define o pipeline a ser utilizado, o diretório onde estão os arquivos para treinamento, onde devem ser criados os modelos gerados a partir do treinamento e outros metadados. Todas as possíveis configurações que podem ser utilizadas no arquivo de configuração podem ser encontradas na documentação oficial. Se o pipeline sendo utilizado for o do `SPACY + SKLEARN` por exemplo, pode-se criar um arquivo chamado `config_spacy.json`.

Depois de especificar o arquivo de configuração bastará rodar a linha de comando para treinar os modelos, e em seguida subir o server do rasa. A partir daí já será possível processar textos e o Rasa NLU devolverá um `JSON` com a classificação da intenção e as entidades identificadas. Para um melhor entendimento da execução desses passos, basta seguir as instruções de construção de um bot básico.

Rasa Core

O RASA Core é uma ferramenta livre para construção de sistemas de conversação, como Messengers e Chat Bots.

Utilizando o Rasa Core

Para utilização do Rasa Core, além das dependências instaladas para utilização do Rasa NLU, é necessário instalar apenas o seu pacote, seguindo a documentação.

É preciso ter um arquivo `domain.yml` que define o universo de atuação do bot. Dentro desse arquivo são especificadas as intenções e ações a serem utilizadas durante a execução do bot.

Dentro do diretório `data` deve existir um arquivo `nlu.md` que define os textos relacionados à cada intenção. E um arquivo `stories.md` que descreve os contextos de conversação esperados a partir das intenções.

Na pasta raiz do projeto é necessário um arquivo de configuração como o utilizado no Rasa NLU, que defina as configurações da pipeline a ser utilizada.

Por último, é necessário executar os comandos para treinar o Rasa NLU e o Rasa Core. São nesses passos onde serão gerados os modelos e os arquivos de treinamento que o bot consumirá. E então,

executar o comando para subir o server do Rasa Core. Para mais informações a respeito da execução destes passos, basta seguir os passos para construção de um bot básico usando o Rasa Core.

Vantagens

- Livre e grátis;
- Todos os serviços citados acima não informações relevantes à respeito das tecnologias usadas(algoritmos de ML e datasets), diferentemente do RASA, que é mais flexível e te permite facilmente escolher às ferramentas a serem utilizadas no módulo de backend;
- O Rasa apresenta performance similar ou superior aos serviços acima citados;
- Possui funcionalidade de importar dados em lote;

Limitações

- Uma das desvantagens em relação aos serviços NLU que são baseados em nuvem é que o Rasa não é tão facilmente escalável;
- O Rasa possui uma pequena base para treinamento inicial. Por exemplo, para o backend usando o MITIE, o Rasa vem com um modelo de linguagem inicial de aproximadamente 300 MB. Enquanto que as outras ferramentas, por serem comerciais, são alimentadas com grandes entradas de dados;

Licença

Tanto o Rasa NLU como o Rasa Core utilizam a licença permissiva **Apache License 2.0** - Rasa NLU license/Rasa Core license.

Analytics

Ambas as bibliotecas, Rasa core e Rasa NLU, não oferecem nativamente suporte à extração e análise de métricas.

Integração dos serviços: BotPress + Rasa NLU + Spacy

Utilização do BotPress

É preciso instalar o pacote do BotPress através do npm.

```
npm install -g botpress
```

Para iniciar um projeto com o BotPress é preciso utilizar o comando abaixo indicando o nome do seu projeto.

```
botpress init my-bot
```

Por último basta iniciar o serviço do BotPress.

```
botpress start
```

Após a execução dos passos anteriores, o BotPress estará rodando por padrão na porta 5000.

Para o BotPress se comunicar com o rasa é preciso instalar o Middleware do rasa, utilizando o comando abaixo.

```
botpress install rasa
```

Em seguida, na aba `Rasa NLU` no dashboard à esquerda é necessário configurar o endereço do host onde o seu Rasa NLU está rodando no campo `Rasa Address`. Se o servidor do Rasa estiver rodando no mesmo host basta apontar para a respectiva porta, que por padrão é a porta 5000. Também é preciso definir no campo `Project Name` o nome da pasta de projeto onde estão os modelos do rasa, que por padrão é a pasta `default`.

Se o servidor do Rasa NLU estiver rodando corretamente o Botpress já estará conectado corretamente e para utilizar o Rasa como backend basta configurar uma resposta que utilize o Middleware do Rasa no arquivo `index.js`. Como no exemplo abaixo, em que sempre que o Rasa NLU indentificar a intenção `greet` o Botpress usará uma das respostas definidas na tag 'greet_response', dentro do arquivo 'content.yml'.

```
bp.hear({'rasa_nlu.intent.name': 'greet'}, (event) => {
  event.reply('#greet_response')
})
```

Utilização do Rasa NLU

É possível instalar o Rasa NLU diretamente pelo pip. No contexto desenvolvimento é mais interessante clonar o repositório do Rasa uma vez que provavelmente vários arquivos serão modificados.

Uma vez configurados os arquivos de acordo com o contexto em questão, é necessário executar o comando abaixo para treinar os modelos com o backend em questão.

```
python -m rasa_nlu.train -c sample_configs/config_spacy.json
```

E então basta subir o servidor do Rasa.

```
python -m rasa_nlu.server -c sample_configs/config_spacy.json
```

Utilizando Botpress web

O Botpress web permite a utilização do chat de duas formas, com um iframe que pode ser utilizado embutido em qualquer página, ou como um web-app na tela inteira. Para utilizar o Botpress web é necessário instalar o pacote via botpress ou npm:

```
botpress install botpress-platform-webchat
```

Uma vez instalado o Middleware web, é possível ver se este está ativado na aba Middleware no menu principal do Botpress, à esquerda no dashboard.

Para utilizar a `view mobile` basta acessar o link a seguir a partir do seu hostname: ${HOSTNAME}/lite/?m=platform-webchat&v=fullscreen (e.g http://localhost:3000/lite/?m=platform-webchat&v=fullscreen

Para embutir o chat em uma página qualquer basta adicionar o código abaixo ao final da tag `<body>` em qualquer arquivo html.

```
<script src="<host>/api/botpress-platform-webchat/inject.js"></script>
<script>window.botpressWebChat.init({ host: '<host>' })</script>
```

Integração dos serviços: Rasa NLU + Rasa Core

Para a instalação do Rasa Core através do pip, utilize o comando abaixo:

```
pip install rasa_core
```

Para a instalação do Rasa NLU, utilize o comando abaixo ou instale através do repositório conforme descrito no tópico anterior (Integração dos serviços: BotPress + Rasa NLU + Spacy).

```
pip install rasa_nlu
```

É possível utilizar o sklearn, spaCy e MITIE como backend para o Rasa NLU. Para utilização do spaCy, execute os seguintes comandos:

```
pip install rasa_nlu[spacy]
python -m spacy download en_core_web_md
python -m spacy link en_core_web_md en
```

O tutorial do Rasa Core descreve um passo a passo para a criação de um bot simples utilizando estas tecnologias.

Anexo III - Planejamento Estratétigo

Planejamento referente à Etapa II do projeto (Março 2018 à Junho 2018)

Metas Estratégicas

- Transformar softwares culturais abrindo-os para práticas colaborativas e abertas (meta estratégica 1)
- Estruturar a equipe e definir os primeiros passos da frente Governança (meta estratégica 3)
- Refatorar e re-escrever chatbot usando tecnologias mais "inteligentes", iniciando a modelagem de um produto lappis de assistentes virtuais para serviços públicos (meta estratégica 4)
- Coleta de dados para estudo sobre melhoria das práticas de gestão e desenvolvimento de software livre (meta estratégica 3)
- Primeitos passos para estudos sobre processos técnicos e gerenciais para aferição e aceitação de produtos de software (meta estratégica 6)

Épicas

Governança

- Sistematizar colaboração entre os atores internos e externos ao minc
- Realizar e fomentar colaboração em software livre nos projetos do minc

Chatbot

- Chat em produção com respostas sobre a lei e sobre o sistema salic
- Evolução do bot com tecnologias mais inteligentes para refinamento de contexto

Plataforma

- Modelo de processo multidisciplinar da equipe
- Protótipos da solução VerSalic

Salic-ML

- Sistematizar modelo de processo e trabalho com machine learning
- Desenhar arquitetura da solução ML

Pesquisa/Paulo

- Análise qualitativa da execução do modelo e fluxo de colaboração entre lappis/minc
- Revisão sistemática sobre devops e mapa conceitual

Anexo IV - Resultados Pesquisa Devops Pesquisa Survey de Acompanhamento

Abaixo segue o resultado da análise do survey realizado com os alunos participantes do projeto.

O formulário enviado para os alunos pode ser acessado em:

https://docs.google.com/forms/d/1SpZMX8qYLZGl7q6nTO4JPpI4eFbMHAJHP5NivG-jMhw/prefill