UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

VANDERLEI LEITE LOUREIRO

APOIO AO DESENVOLVIMENTO DE STI ATRAVÉS DE UM SERVIÇO TUTOR ADAPTÁVEL À CLASSIFICADORES

RUSSAS

2020

VANDERLEI LEITE LOUREIRO

APOIO AO DESENVOLVIMENTO DE STI ATRAVÉS DE UM SERVIÇO TUTOR ADAPTÁVEL À CLASSIFICADORES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Marcio Costa Santos

RUSSAS

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

L555a Loureiro, Vanderlei Leite.
Apoio ao desenvolvimento de STI através de um serviço tutor adaptável à classificadores / Vanderlei Leite Loureiro. – 2020.
44 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Engenharia de Software, Russas, 2020.
Orientação: Prof. Dr. Marcio Costa Santos.

1. Sistema Tutor Inteligente. 2. Microsserviço. 3. Representational State Transfer (REST). 4. Árvore de decisão. I. Título.

CDD 005.1

VANDERLEI LEITE LOUREIRO

APOIO AO DESENVOLVIMENTO DE STI ATRAVÉS DE UM SERVIÇO TUTOR ADAPTÁVEL À CLASSIFICADORES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Marcio Costa Santos (Orientador)
Universidade Federal do Ceará (UFC)

Dr. Bonfim Amaro Júnior
Universidade Federal do Ceará (UFC)

Prof. Ms. Filipe Maciel Roberto
Universidade Federal do Ceará (UFC)

Aos meus pais, nascidos e criados no interior, sem ensino superior, que sempre me incentivaram aos estudos e batalharam para que cada conquista minha fosse possível.

AGRADECIMENTOS

Ao Prof. Dr. Marcio Costa Santos por me orientar em meu Trabalho de Conclusão de Curso.

Aos meus pais, tios, primos e avós por todo apoio e compreensão de minha ausência em determinados momentos.

À todos os professores, que direta ou indiretamente, contribuíram para minha formação acadêmica e pessoal.

Aos amigos que tive a oportunidade de compartilhar moradia durante o período de graduação, fazendo com que a proximidade geográfica à universidade tornasse o processo de formação um pouco mais fácil.

“A experiência é um troféu composto por todas as armas que nos feriram.”

(Marco Aurélio)

RESUMO

Sistemas Tutores Inteligentes são softwares educacionais aperfeiçoados com técnicas de inteligência artificial para proporcionar um ensino mais adaptável e exclusivo ao aluno. O desenvolvimento de software traz dificuldades na arquitetura à medida que cresce e essas dificuldades levaram ao padrão de microsserviços. Entretanto, ainda não foram encontrados registros de STIs construídos em uma arquitetura de microsserviços. Visto isso, foi buscando realizar um apoio na construção e experimentação de novos STIs por terceiros nessa arquitetura. Foi desenvolvido um microsserviço em forma de abstração para dois módulos de um STI. O projeto desenvolvido apresentou o comportamento esperado em termos de integração e em regras de negócio.

Palavras-chave: Sistema Tutor Inteligente. Microsserviço. Representational State Transfer (REST). Árvore de decisão.

ABSTRACT

Intelligent Tutoring Systems are educational software perfected with artificial intelligence techniques to provide a more adaptable and exclusive teaching to the student. Software development brings difficulties in architecture as it grows and these difficulties have led to the microservice pattern. However, no records have yet been found of ITS built on a microservice architecture. In view of this, it was seeking to provide support in the construction and experimentation of new STIs by third parties in this architecture. An abstraction microservice was developed for two modules of an STI. The developed project presented the expected behavior in terms of integration and in business rules.

Palavras-chave: Intelligent Tutoring System. Microservice. Representational State Transfer (REST). Decision Tree.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

STI	Sistema Tutor Inteligente
CAI	*Computer-Assisted Instruction*
IA	Inteligência Artificial
TOEFL	*Test of English as a Foreign Language*
ISO	*International Standards Organization*
REST	*Representational State Transfer*
JSON	*JavaScript Object Notation*

SUMÁRIO

1 INTRODUÇÃO

Softwares cumprem um papel transformador no processo de ensino e aprendizagem, atuando como ferramentas potencializadoras, capazes de promover práticas pedagógicas inovadoras. Esses recursos permitem estimular o aprendiz em seu próprio desenvolvimento, tornando-o um ser mais ativo nesse processo através do uso de novas metodologias educacionais (FALKEMBACH, 2005).

Segundo (FIALHO, 2010), vários são os fatores que dificultam o aprendizado tradicional por parte dos alunos, tais como falta de motivação, interesse e atenção. Sendo importante citar que há também uma seleção de conteúdo por parte do aluno que define quais conteúdos o mesmo tem mais interesse e afinidade, impactando diretamente a eficácia do ensino. O uso de softwares educativos aumentam essa dinâmica de ensino, flexibilizando novas apresentações de conteúdos e tornando o ensino algo mais envolvente, motivador e lúdico.

Sistemas Tutores Inteligentes são softwares educacionais aperfeiçoados com técnicas de Inteligência Artificial, que permitem ao estudante aprender com um tutor artificial, onde esse tutor cumpre um papel de guia no processo de aprendizagem. Um Sistema Tutor Inteligente permite aplicar estratégias pedagógicas individuais para cada aprendiz, devendo adaptar tais estratégias de acordo com o ritmo de cada estudante (GUELPELI *et al.*, 2004).

O desenvolvimento de sistemas é desafiador, entre eles, sistemas voltados à aprendizagem. O caminho natural para o desenvolvimento de um sistema é uma abordagem monolítica, onde toda a lógica é centralizada em um único processo. Normalmente, aplicações monolíticas são compostas de três partes principais:

1. Uma interface de usuário;
2. Um banco de dados e;
3. uma camada de negócio, onde são tratadas as requisições, acesso ao banco e regras da aplicação.

Essa abordagem traz dificuldades em momentos, onde por exemplo, uma pequena modificação em uma parte na aplicação, exige que a aplicação seja completamente atualizada - *build* e *deploy* - para uma nova versão e assim refletir essa modificação para o usuário final (FOWLER; LEWIS, 2014).

Essa e outras dificuldades presentes em manter aplicações monolíticas fizeram com que houvesse um salto nos últimos anos para construções de sistemas usando estilo de arquitetura em microsserviços. Arquitetura de microsserviços visa a construção de uma aplicação através de

suítes de serviços menores e independentes que se comunicam entre si através de mecanismos leves, permitindo o desenvolvimento e escalabilidade independente, diferentes linguagens de programação e alocação de serviços para times diferentes (FOWLER; LEWIS, 2014).

Entretanto, as pesquisas realizadas à trabalhos voltados ao desenvolvimento de Sistema Tutor Inteligente (STI) e à sistemas educativos no geral, não apresentou resultados de exemplos de construções dos mesmos em uma arquitetura de microsserviços. Os resultados mostraram aplicações monolíticas, que mesmo tendo resultados satisfatórios em suas propostas, limitavam o uso da plataforma apenas no ambiente em que havia sido originalmente desenvolvida, seja ela Mobile, Web, ou Desktop.

Um STI comumente é apresentado na literatura como uma composição de quatro principais módulos: especialista, diagnóstico do estudante, instrução e interface. Esses módulos definem contextos e responsabilidades que podem ser construídos em forma de microsserviços independentes, aumentando a coesão e dando mais flexibilidade para a construção. De forma que um serviço possa ser substituído ou mude seu algoritmo e isso seja indiferente para os outros serviços, desde que mantenha o mesmo contrato.

A partir disso, o trabalho busca auxiliar o desenvolvimento de novos Sistemas Tutores Inteligentes em uma arquitetura de microsserviços. Oferecendo apoio para terceiros através de base de código, arquitetura e integrações. Busca-se facilitar experimentos de novos algoritmos de diagnóstico, construção de novos serviços ou módulos e estudos de casos.

O trabalho tem o objetivo de desenvolver um microsserviço semelhante aos módulos de Especialista e Instrução, com funcionalidades essenciais para uma plataforma de ensino, capaz de integrar-se à um segundo serviço inteligente, onde o mesmo pode classificar e inferir melhores apresentações de conteúdos para o serviço principal.

O trabalho está organizado da seguinte forma: a Sessão 2 apresenta a fundamentação teórica dos assuntos bases; a Sessão 3 apresenta os principais estudos relacionados que foram encontrados, mostrando suas semelhanças e diferenças; na Sessão 4 é apresentado a metodologia de trabalho, o microsserviço desenvolvido e detalhes de sua construção; a Sessão 5 mostra a forma e teste e seus resultados, por fim, a Sessão 6 apresenta uma conclusão e definição de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistema Tutor Inteligente

Um Sistema Tutor Inteligente (STI) é uma evolução de sistemas *Computer-Assisted Instruction* (*CAI*), aperfeiçoado com técnicas de Inteligência Artificial (IA). Possibilitam ao estudante a capacidade de aprender com um tutor artificial, que serve como guia no processo. Este último deve se adaptar ao aprendiz, e não o contrário, como acontece em métodos tradicionais. Com isso, é necessário uma modelagem do aprendiz, para que o STI possa saber o que ensinar, a quem ensinar e como ensinar (GUELPELI *et al.*, 2004).

Houve um grande aprimoramento dos STIs desde seu surgimento nos anos de 1970. Uma ampla variedade de multimídia e técnicas proporcionam motivação e avaliação ideal para os alunos. Os últimos crescimentos nesses sistemas comprovou que os mesmos podem melhorar o desempenho em áreas e habilidades precisas (AKKILA *et al.*, 2019).

Xu *et al.* (2019) realizou uma análise da eficácia de STIs para compreensão da leitura dos alunos de ensino médio e fundamental, constatando que esses sistemas são uma forma escalável e acessível de auxiliar o ensino, visto que a tutoria humana pode ser cara, inconsistente e difícil de escalar. Além disso, economiza-se tempo do aluno e professor e o aluno não tem medo de cometer erros, pois trata-se de um ensino individualizado (AL-SHAWWA *et al.*, 2019).

Devido a flexibilidade e a multidisciplinaridade de um Sistema Tutor Inteligente, o mesmo pode ser aplicado para o ensino de línguas estrangeiras. Bakeer e Abu-Naser (2019) apresenta um STI que prepara alunos para realizar o *Test of English as a Foreign Language* (*TOEFL*), que é um teste aceito por muitas instituições acadêmicas e profissionais de língua inglesa para medir a capacidade de falar inglês de não-nativos. O sistema apresenta palestras, exemplos e questões a serem respondidas pelo aluno, abrangendo os quatro pilares do aprendizado: escuta, fala, leitura e escrita.

Polson e Richardson (2013) apresenta uma arquitetura clássica de um sistema tutor inteligente que ainda se mantém atual, como por exemplo do trabalho de Alshawwa *et al.* (2019). A arquitetura básica é composta por quatro módulos que serão detalhados a seguir, sendo eles: módulo especialista, módulo de diagnóstico do estudante, módulo de instrução e módulo de interface.

Módulo especialista, também encontrado na literatura como módulo de conhecimento, ou módulo de domínio, refere-se ao componente do sistema responsável por armazenar

os conteúdos e disciplinas que devem ser apresentadas ao aluno, em linhas gerais, é o domínio do conhecimento.

Módulo diagnóstico do estudante, também encontrado na literatura módulo do professor, nesse componente é onde se encontram as técnicas e algoritmos de IA que avaliam o desempenho do aluno durante as respostas dos exercícios e atividades. A partir desse desempenho o sistema recomenda questões e conteúdos correspondentes ao nível cognitivo do aprendiz, de forma que apresentação não fique muito fácil ou muito difícil.

Enquanto o módulo de diagnóstico, como próprio nome sugere, serve para diagnosticar o desempenho do aluno em uma dada forma de apresentação de conteúdo, o módulo de instrução armazena a individualidade do usuário do STI, guardando a didática, ordem de apresentação dos conteúdos e todas as informações relacionadas à individualidade do aprendiz.

A interface do usuário é o meio de comunicação entre o sistema com o usuário final, responsável por apresentar todas as funcionalidades tanto para o aprendiz quanto para um tutor externo que queira ver relatórios ou adicionar mais conteúdos ao módulo especialista.

Erümit *et al.* (2019) apresenta uma avaliação de usabilidade que de um STI e faz sugestões para processos de avaliação de usabilidade de sistemas similares. A avaliação é medida por critérios de eficácia, eficiência e satisfação incluídos na definição de usabilidade pela *International Standards Organization* (*ISO*). O estudo é útil para orientar pesquisadores que desejam trabalhar com sistemas semelhantes, que é o caso do presente trabalho.

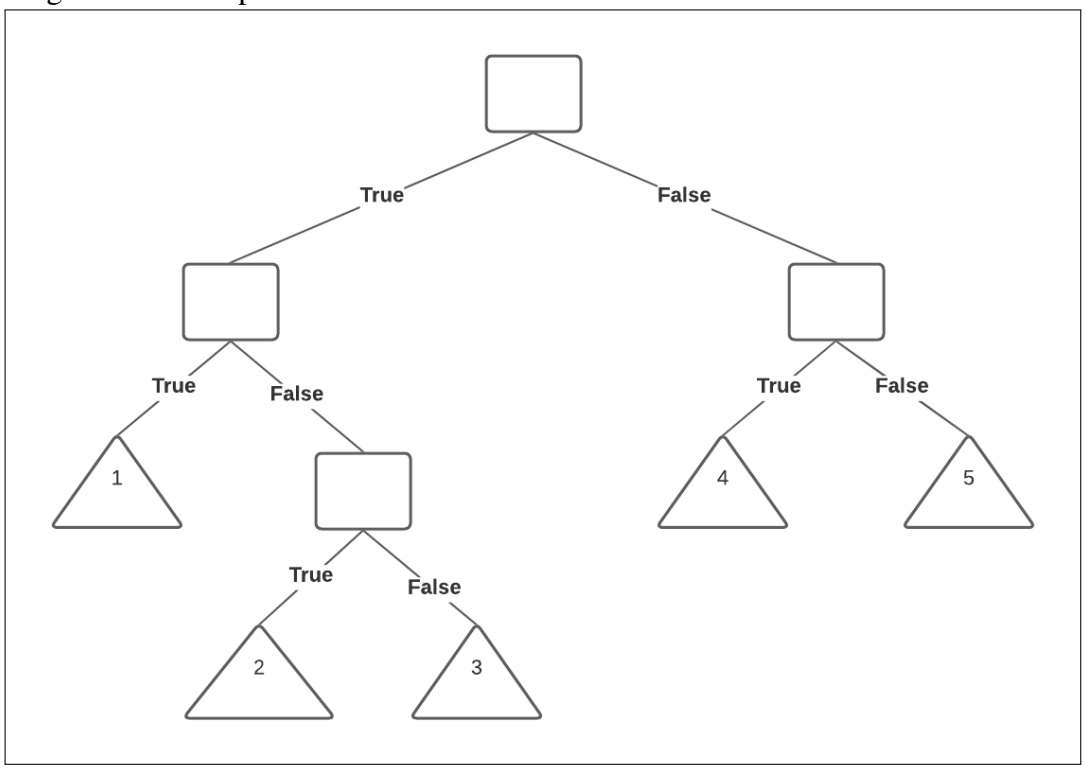
2.2 Árvore de decisão

Uma árvore de decisão é um formalismo para representar o mapeamento de objetos ou situações em classes. Classes são rótulos mutualmente exclusivos, como diagnósticos médicos, categorias de imagens ou projeções qualitativas econômicas, usados para classificação do objeto de estudo (QUINLAN, 1996).

Uma árvore de decisão é formada por folhas, nós e ramos: os nós são elementos responsáveis por realizar as funções lógicas para separação dos dados. Um nó é ligado à outro através de um ramo. O primeiro nó é chamado de nó raiz, enquanto os nós que ficam abaixo são chamados se nós filhos. Por fim, as folhas são os últimos nós da árvore que representam um rótulo ou valor (SATO *et al.*, 2013).

Árvores de decisão possuem vantagens como a interpretação dos seus resultados, pois a classificação é obtida de forma explícita. Geralmente a eficiência computacional apresentada

Figura 1 – Exemplo de árvore de decisão



Fonte: Elaborado pelo próprio autor (2020)

por essa técnica também contribui para uma apresentação rápida dos resultados, dependendo também do problema e da forma como será representado (SATO *et al.*, 2013).

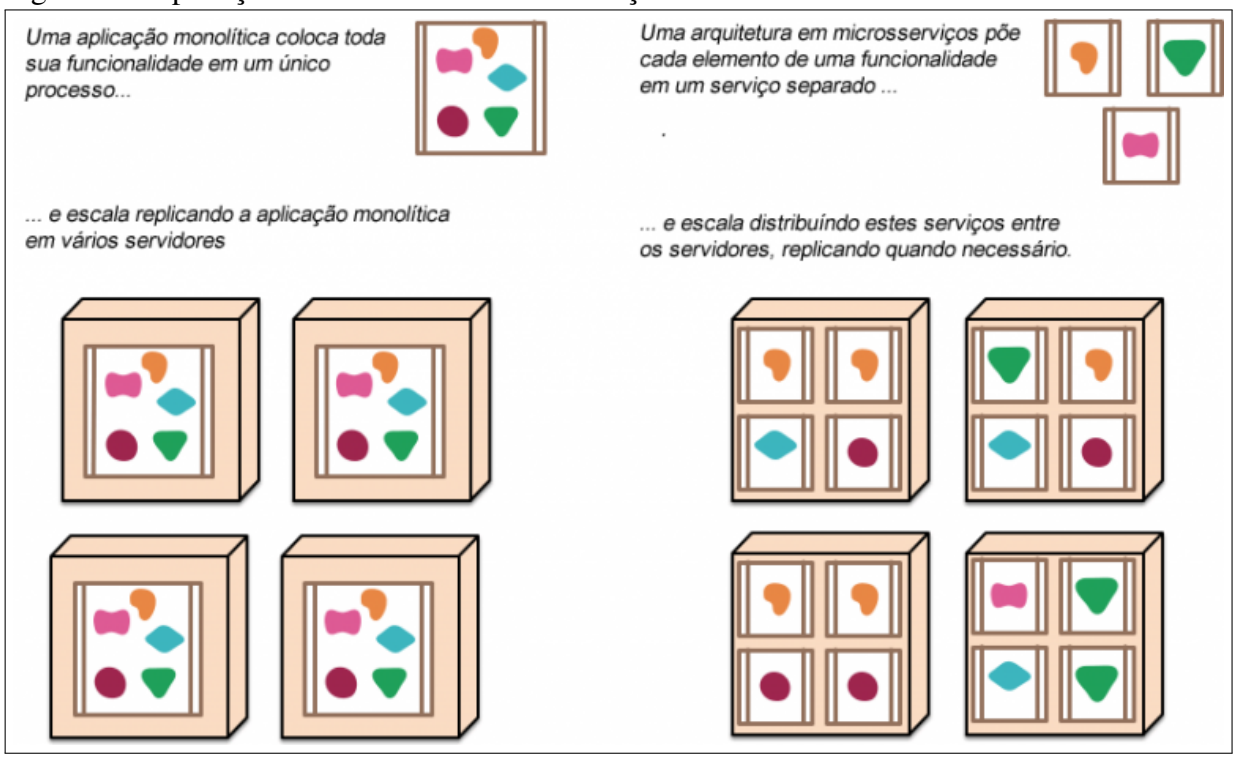
2.3 Arquitetura de Microsserviços

Microsserviço é uma abordagem para o desenvolvimento de software onde uma única aplicação é desenvolvida como uma suíte de serviços. Cada serviço é construído a partir de pequenas responsabilidades, rodando em processos diferentes e sendo publicados em produção de maneira independente. Diferente de uma abordagem monolítica, onde toda a aplicação é construída sobre uma única unidade lógica executável e roda em um único processo (FOWLER; LEWIS, 2014).

A abordagem de desenvolvimento monolítica é uma forma natural para o desenvolvimento, onde toda a aplicação se encontra em um único processo, porém, a medida que o sistema cresce, essa abordagem começa a apresentar desafios. Uma pequena mudança em uma parte do código exige que toda a aplicação seja publicada em produção novamente. Se alguma parte precisar de maiores recursos, toda a aplicação deverá ser escalada, além das limitações de uma única linguagem para toda a aplicação (FOWLER; LEWIS, 2014).

Abordagem de microsserviços buscam resolver estes problemas, com serviços que

Figura 2 – Aplicações monolíticas e microsserviços



Fonte: (FOWLER; LEWIS, 2014) traduzido por (MENDES, 2016).

podem ser implantados e escalados de maneira independente. Cada serviço define fronteiras bem definidas entre os módulos, podendo ser definidos pelas áreas de negócio da aplicação. Além de poderem ser construídos com linguagens diferentes (FOWLER; LEWIS, 2014).

Segundo Fowler e Lewis (2014) o termo microsserviço ganhou um significado um pouco infeliz, devido ao prefixo "Micro", podendo ser assimilado à ideia de um serviço muito pequeno, embora não seja esse o objetivo. Por esse motivo, neste trabalho, os termos Serviços e Microsserviços são usados como sinônimos.

2.4 REST

Transferência Representacional de Estado, do inglês *Representational State Transfer* (*REST*), é um estrito estilo de arquitetura para construção de serviços web. O REST dispõe de uma interface com uma semântica uniforme para manipulação de recursos, basicamente manipulação de criação, busca, atualização e exclusão dos mesmos. Uma outra característica fundamental do REST é sua natureza sem estado - do inglês, *Stateless* - onde uma requisição para manipulação de um recurso não tem conhecimento e não depende do estado da conversa, isto é, das requisições anteriores (W3C, 2004).

Os recursos são disponibilizados através de URIs únicas (Uniform Resource Identifi-

ers) e podem ser representados em diferentes formatos de dados, como por exemplo: HTML, XML, CSS, PNG, JPG (W3C, 2004).

Como um exemplo de disponibilização de recurso, pode-se analisar a URI a seguir, que representa um recurso de Aluno em um sistema hipotético através de seu número de identificação:

http://www.sistemaufc.edu.br/alunos/353535

O presente trabalho usa *JavaScript Object Notation* (*JSON*), que em português significa Notação de Objetos JavaScript, como formato de dado para comunicação. Se trata de uma formatação simples para representação de dados, em forma de texto, independente de linguagem de programação (JSON.ORG, 2015).

Um objeto JSON é delimitado por duas chaves, cada atributo do objeto é definido dentro das suas duas chaves por: *String*, dois pontos e o seu valor. O valor pode adotar vários tipos, como *String*, *Number*, *Boolean*, um outro objeto ou *null*. *Arrays* também são permitidos, sendo definidos por dois colchetes com o seu conteúdo entre eles (JSON.ORG, 2015). Como exemplo, segue uma representação de um recurso de aluno:

Código-fonte 1 – Representação Aluno em formato JSON

```
{
    "codigoAluno": 353535,
    "nome": "Vanderlei Leite Loureiro",
    "ira": 8.0,
    "nomeCurso": "Engenharia de Software"
    "dataNascimento": "1997-03-29",
    "campus": "Russas"
}
```

2.5 Spring Boot

Spring Boot é um *framework* que facilita a criação de aplicações no ecossistema Spring, adicionando servidor embutido, dependências iniciais configuradas e funcionalidades opcionais de monitoramento da aplicação com métricas e dados sobre a saúde da aplicação. Além disso, o *framework* engloba outros do mesmo ecossistema, como o Spring Framework para configuração de aplicações corporativas modernas e Spring MVC para tratamento de requisições web, delegando para seus componentes responsáveis (SPRING, 2018).

Ao gerar um projeto Spring Boot, ele rodará por padrão usando a URI:

http://localhost:8080

A seguir há um exemplo de uma classe controladora de alunos, intitulada *Alunos-Controller*. A classe tem como objetivo tratar requisições REST referente ao recurso de Aluno.

Código-fonte 2 – Controllador de Aluno com framework Spring

```
1 @RestController
2 @RequestMapping("/aluno")
3 public class AlunoController {
4
5     @GetMapping("/{codigoAluno}")
6   public String obterAluno(@PathVariable long codigoAluno)
      {
7     return "Bem vindo, aluno!";
8   }
9 }
```

O Spring fornece a anotação *@RestController* para informar que o retorno dos métodos do controlador será em um objeto de dados HTTP. Com a anotação *@ResquestMapping*, é informado ao Spring que a classe tratará requisições para o caminho passado dentro das aspas. A anotação *@GetMapping* informa que o método abaixo atenderá a uma requisição HTTP com o verbo GET para a respectiva URI. Por fim, a anotação *@PathVariable* significa que o *codigoAluno* será um valor que pode ser variado na URI da requisição, sendo usado como um parâmetro (SPRING, 2018). Com isso, o método *obterAluno()* seria consultado através da seguinte URI:

http://localhost:8080/aluno/codigoAluno

Sendo *codigoAluno* um valor numérico. É possível perceber que apesar do código do aluno passado como parâmetro, o método retorna sempre a mensagem "Bem vindo, aluno!", pois trata-se apenas um exemplo superficial para demonstração de anotações importantes do framework.

3 TRABALHOS RELACIONADOS

Devido ao objetivo do trabalho em desenvolver um microsserviço como abstração de módulos de um STI, o assunto mais coerente encontrado para realização de pesquisa de trabalhos relacionados foi o de Sistemas Tutores Inteligentes, buscando exemplos construídos em arquitetura de microsserviços ou semelhante. A busca encontrou diversos resultados em diferentes áreas de ensino.

Foi observado nos resultados mais relevantes da pesquisa um certo padrão nos trabalhos: sua grande maioria relatava algum contexto de experimentação de um Sistema Tutor Inteligente, seguindo de uma análise de seus resultados e apenas uma pequena parte mostrava de forma detalhada a construção de um STI, abordando contextos de arquitetura de software, algoritmo utilizado e estratégias de didáticas.

Schaab *et al.* (2015) apresenta uma uma avaliação experimental com 46 alunos através do uso de um Sistema Tutor Inteligente para a prática de *mindfulness*. Dividindo em dois grupos onde um ouvia a sessão de *mindfulness* enquanto outro ouvia aulas de álgebra. Mesmo sem muita diferença entre os grupos, observou-se uma melhora da atenção, uma possível justificativa seria que as aulas funcionaram também como uma prática indireta de *mindfulness*. O trabalho não apresenta informações precisas sobre a construção do STI em si, mantendo o foco do trabalho em relação a sua aplicação em uma situação prática.

Ferreira *et al.* (2018) apresenta um trabalho sobre uma construção de uma API REST voltada à educação no Instituto Federal de Alagoas, também utilizando Java, entretanto, trata-se de uma API de integração com um sistema acadêmico presente na universidade em questão. A API tem como objetivo prover uma interface fácil para que outros desenvolvedores ou pesquisadores acessem informações presentes da base de dados do sistema acadêmico implantado na universidade, visto que o mesmo não possui esse recurso.

Um sistema de recomendação educacional é abordado no trabalho de Vargas *et al.* (2017). No trabalho é usado uma técnica de *ranking* usada em jogos chamada ELO, abstraindo a apresentação de objetos de aprendizagem como se fossem duelos entre os problemas de programação e os estudantes. Observou-se que problemas apresentados que tinham um ELO próximo ao do aluno geraram um aprendizado potencializado. O estudo utilizou a plataforma de programação URI Online Judge, a plataforma disponibiliza um repositório de objetos de aprendizagem, as submissões são encaradas como os duelos e a aplicação proposta no trabalho armazena os objetos de aprendizagem e o retorno recebido pela submissão da atividade, com

isso, os ELOs dos alunos e desafios são calculados.

Fazendo parte do grupo dos trabalhos que demonstravam a construção de um STI, o trabalho de Martins *et al.* (2007) aborda o desenvolvimento de um Sistema Tutor Inteligente usando aprendizado por reforço com o algoritmo Softmax. A análise através de estatística demonstrou resultados promissores com a aplicação do STI. O trabalho não apresenta termos referentes a tecnologia de desenvolvimento e não cita se o sistema tem algum meio de integração, como uma API REST ou Web Service SOAP.

Valeriano *et al.* (2019) apresenta o MAZK, um sistema tutor inteligente para o ensino no fundamental I, como uma alternativa para enfrentar os desafios da rede pública de ensino, proporcionando uma metodologia mais adaptável ao nível de habilidade de cada aluno. O estudo foi realizado em quatro turmas de duas escolas da rede municipal de Araranguá, com turmas que iam do primeiro ao quinto ano. O trabalho apresentou o MAZK como sendo uma ferramenta potente para o ensino e aprendizagem.

Abaixo segue uma tabela com os principais pontos de diferença entre os trabalhos relacionados e o presente trabalho. Vale ressaltar que campos preenchidos como *Não*, não significam necessariamente que o sistema desenvolvido ou utilizado no respectivo trabalho não possui determinada característica, significa apenas que essa característica não foi citada no trabalho em questão.

Tabela 1 – Diferença entre os trabalhos relacionados

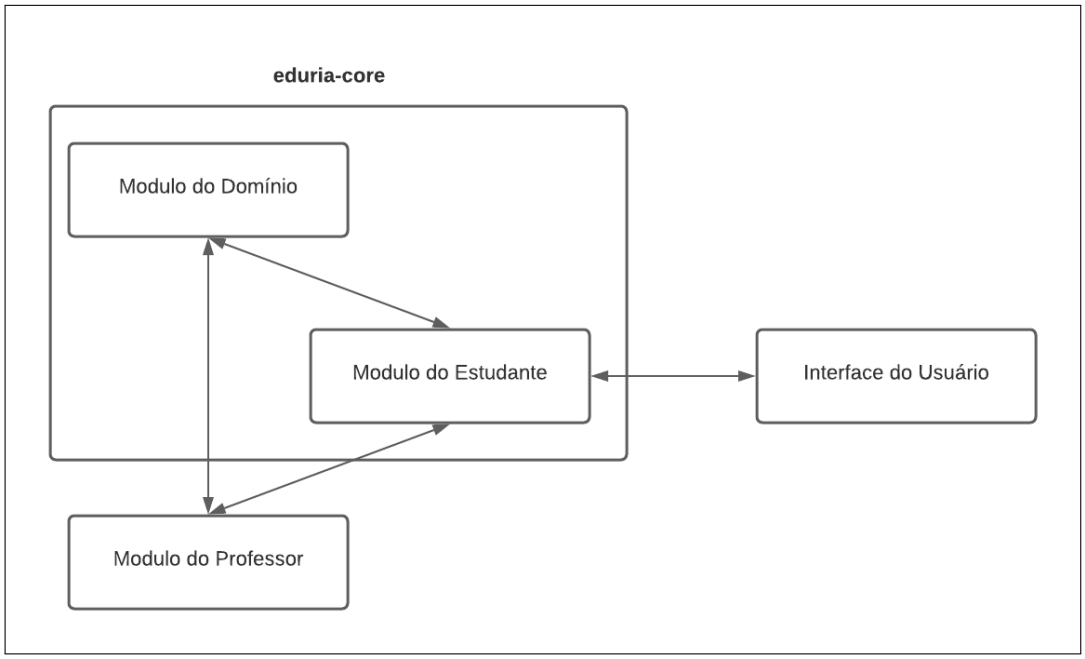
Autor	STI	REST API
Schaab *et al.* (2015)	Sim	Não
Ferreira *et al.* (2018)	Não	Sim
Vargas *et al.* (2017)	Sim	Não
Martins *et al.* (2007)	Sim	Não
Valeriano *et al.* (2019)	Sim	Não
Este Trabalho	Sim	Sim

Fonte: elaborado pelo autor (2020).

4 SERVIÇO DESENVOLVIDO

O serviço desenvolvido buscou ser uma ferramenta utilitária para outros desenvolvedores e pesquisadores no estudo e desenvolvimento de novos Sistemas Tutores Inteligentes. O serviço web intitulado *Eduria-core* é uma abstração para o módulos especialista e instrução descritos do capítulo dois.

Figura 3 – Módulos do serviço desenvolvido



Fonte: Elaborado pelo próprio autor (2020)

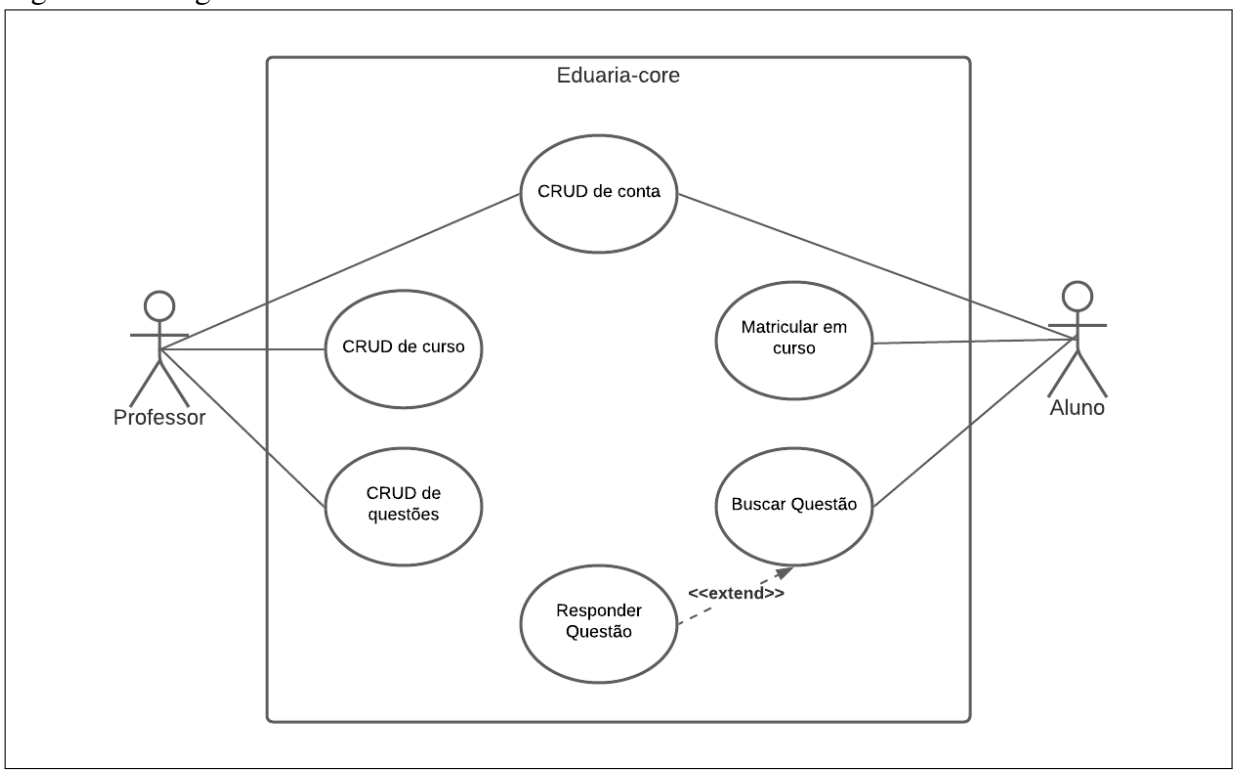
O módulo aqui desenvolvido permite uma integração aos demais serviços e módulos, como módulo de interface, pois o mesmo opera sobre a arquitetura *REST*, assim como permite integração com algum serviço de inteligência.

4.1 Requisitos

O levantamento dos requisitos ocorreu através de arqueologia de sistemas e revisão de trabalhos relacionados. A partir disso, foram percebidas e levantadas questões comuns e necessárias para esse formato de aplicação. O usuário final no sistema pode assumir dois papéis: como um professor, torna-se responsável para prover o ensino, assim como em um ensino tradicional fora do ambiente virtual; já como um aluno, o mesmo pode interagir com o ensino proposto pelo professor, ingressando em turmas e respondendo os materiais cadastrados.

O desenvolvimento da aplicação foi gerenciada com a plataforma Trello e utilizando

Figura 4 – Diagrama de casos de uso



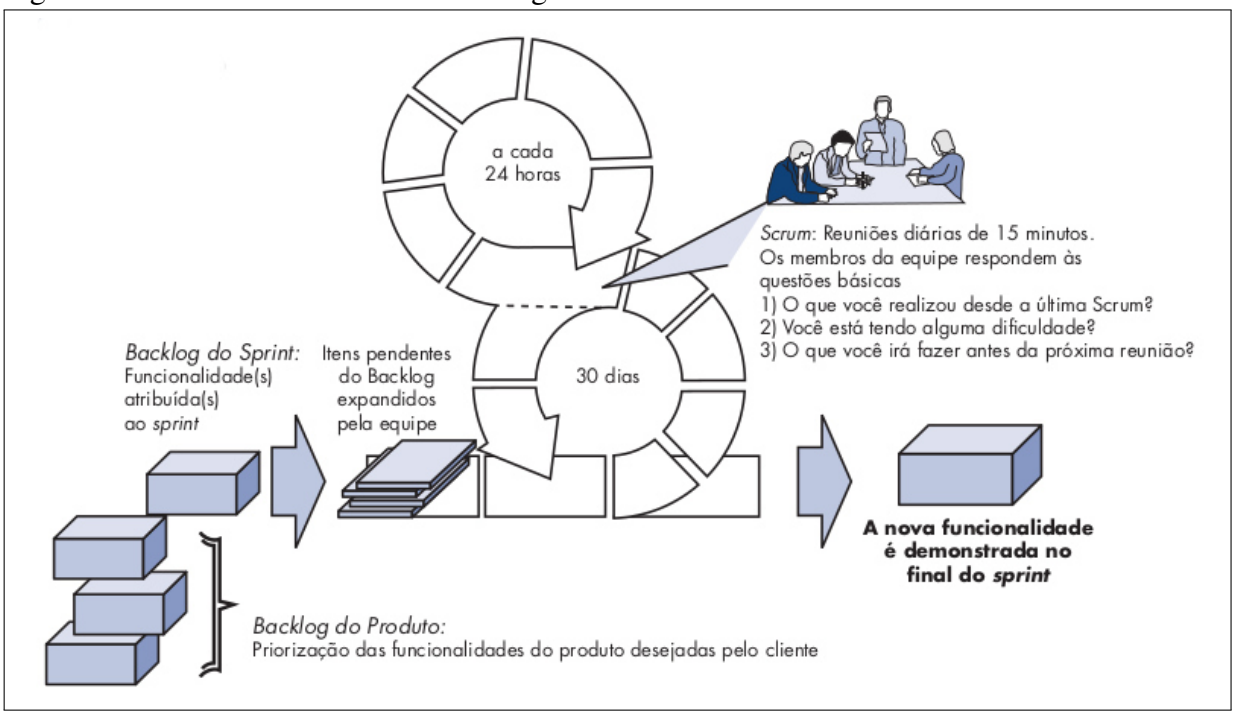
Fonte: Elaborado pelo próprio autor (2020).

Scrum, um método ágil de desenvolvimento de software. As atividades são desenvolvidas dentro de um padrão de processo chamado *Sprint*, que consiste de uma unidade de trabalho em um prazo pré-definido. As *sprints* se comportam em ciclos, realizando as atividades e requisitos, buscando concluir todas que estão presentes na lista de registro pendente de trabalhos, também conhecido como *Backlog* (Pressman, 2011).

As Sprints do projeto foram dimensionadas em intervalos semanais devido ao tamanho das atividades e uma melhor mensuração de progresso em curto prazo. Os requisitos foram planejados em um modelo de História de Usuário e depois quebradas em atividades com teor mais técnico. Histórias de Usuário (User Stories) são incrementos funcionais que contribuem para o valor do produto, sempre enfatizando a atomicidade dessas atividades (Agile Alliance). Abaixo seguem as histórias de usuário levantadas:

1. Como um professor, quero realizar meu cadastro de usuário para poder utilizar as funcionalidades permitidas ao meu perfil de professor.
2. Como um aluno, quero realizar meu cadastro de usuário para poder utilizar as funcionalidades permitidas ao meu perfil de aluno.
3. Como um professor, quero cadastrar, alterar, excluir e visualizar todas as minhas turmas para poder gerenciá-las.

Figura 5 – Modelo de desenvolvimento ágil Scrum



Fonte: Pressman (2011).

4. Como um professor, quero cadastrar conteúdos em minhas disciplinas para que os alunos possam ler e responder.
5. Como um aluno, quero me matricular na turma para que possa interagir com os conteúdos cadastrados.
6. Como um professor, quero definir nível de dificuldade das questões, para que cada aluno veja questões do seu nível.
7. Como um aluno, quero responder questões apresentadas em uma turma.

Um requisito não funcional importante é a operabilidade do serviço principal *Eduria-core* mesmo sem o serviço classificador. Caso o classificador eventualmente pare de responder o seja desativado, a aplicação principal deve continuar seu funcionamento de uma forma menos danosa possível ao cliente final, mesmo perdendo desempenho na acurácia de escolha de nível de questão apresentada. O nível das questões será escolhido de forma aleatória.

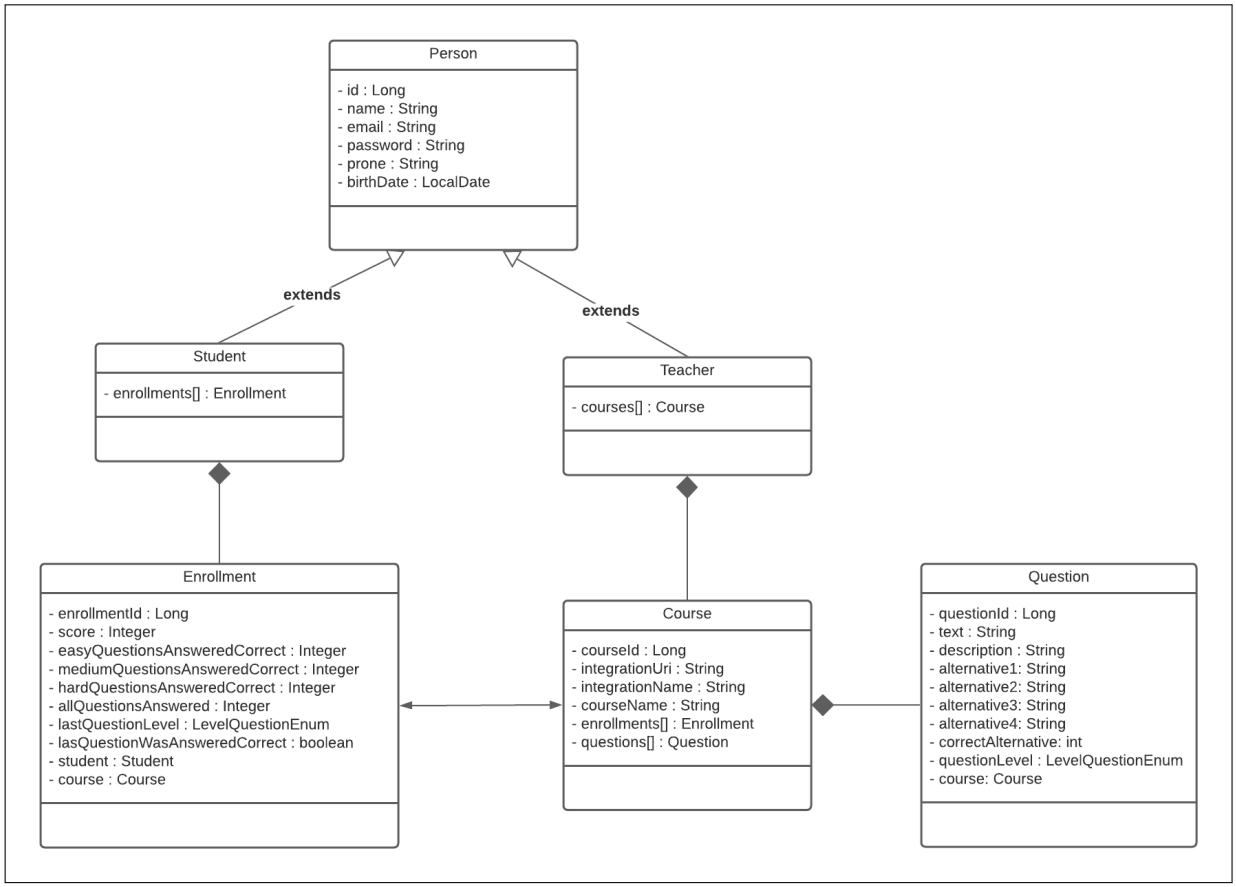
4.2 Serviço de domínio e especialista

O objeto de estudo encontra-se nesse serviço. Tendo em vista as histórias de usuários levantadas, foram levantadas classes principais para a aplicação, como: *Teacher, Student, Course e Question*, após levantamento das primeiras classes, as mesmas foram detalhadas e adicionadas outras mais.

O modelo de entidade implicitamente demonstra algumas das regras de negócios presentes na aplicação. Um aluno, por exemplo, pode se matricular em vários cursos, essa vinculação fica armazenada dentro dos objetos da classe Enrollment. Com isso é possível ter níveis de habilidades diferentes para cada um desses cursos, dependendo apenas do desempenho que o aluno apresenta em cada um deles.

Uma outra regra de negócio é o fato de que o professor pode ter muitos cursos e cadastrar questões específicas e diferentes para cada um dos cursos. As questões seguem um padrão contendo um texto, uma descrição da pergunta e quatro alternativas possíveis, assim como um atributo que especifica qual dessas quatro alternativas é a correta. A figura 10 mostra a representação das entidades:

Figura 6 – Modelo de entidades



Fonte: Elaborado pelo próprio autor (2020).

Uma das principais vantagens do serviço é a capacidade de integração com um serviço de inteligência classificador, capaz de classificar o nível da questão que deve ser apresentada ao aluno a cada iteração. O serviço de inteligência é uma abstração para o módulo de disgnóstico de estudante, visto do capítulo dois.

Essa integração é registrada no cadastro do curso, onde o professor informa o campo *integrationUri*, que significa qual é a URI do serviço web que contem o *endpoint* para consulta do nível e o campo *integratioName*, que serve basicamente como um rótulo para dar um nome particular ao serviço.

Desta forma, cada curso terá que informar seu serviço de integração. Isso abre a possibilidade de colocar classificadores diferentes para cada curso, ou criar dois cursos idênticos mas usando classificadores diferentes e comparar seus desempenhos. Entretanto, se o serviço de inteligência permitir, pode-se usar apenas um para todos os cursos, informando apenas a mesma URI.

O Apêndice A apresenta a tela de documentação Swagger com o *endpoint* para buscar uma questão. Na imagem é possível ver o campo de matrícula do aluno *(enrollmentId)* que deve ser informado e um modelo de JSON em caso de sucesso na requisição.

4.3 Contrato de integração

Como citado em tópicos anteriores, a aplicação principal *eduria-core* usa a arquitetura REST para comunicação entre os serviços. Dessa forma, essa comunicação exige uma interface para determinar o contrato de comunicação.

Para integrar com a aplicação principal, o serviço de inteligência deve dispor de um *endpoint* com o verbo POST que receba um JSON e retorne um segundo JSON com *StatusCode* da requisição com código *200 OK*. O caminho para esse *endpoint* deve ser informado no cadastro do curso.

A cada solicitação de apresentação de questão, uma requisição é feita ao serviço de inteligência. Nessa requisição é enviado um objeto contento várias informações sobre o estado do aluno, do curso e da matrícula. Essas informações tem o objetivo de serem genéricas de forma que possam ser usadas por diferentes algoritmos, estratégias ou heurísticas.

Código-fonte 3 – JSON enviado ao serviço de inteligência

```
1 {
2     "registerUuid": string,
3     "studentId": number,
4     "studentAge": number,
5     "courseName": string,
```

```
    "easyQuestionsAnsweredCorrect": number,
    "mediumQuestionsAnsweredCorrect": number,
    "hardQuestionsAnsweredCorrect": number,
    "qttAllQuestionsAnswered": number,
    "qttAllCourseQuestions": number,
    "score": number,
    "lastQuestionLevel": string,
    "lastQuestionWasAnsweredCorrect": string
}
```

Abaixo segue uma lista com a explicação de cada atributo enviado no objeto JSON da requisição.

registerUuid: Identificador gerado para cada requisição.

studentId: Código do registro do estudante no banco de dados.

studentAge: Idade do estudante.

courseName: Nome do curso.

easyQuestionsAnsweredCorrect: Quantidade de questões fáceis respondidas corretamente pelo aluno no curso.

mediumQuestionsAnsweredCorrect: Quantidade de questões intermediárias respondidas corretamente pelo aluno no curso.

hardQuestionsAnsweredCorrect: Quantidade de questões difíceis respondidas corretamente pelo aluno no curso.

qttAllQuestionsAnswered: Quantidade de questões respondidas pelo aluno no curso, independente de acertos ou erros.

qttAllCourseQuestions: Quantidade de questões cadastradas no curso.

score: Pontuação atribuída à matrícula do aluno.

lastQuestionLevel: Nível da última questão apresentada, representa um enumerável que pode pode assumir os valores EASY (fácil), MEDIUM (médio) ou HARD (difícil).

lastQuestionWasAnsweredCorrect: Se a última questão apresentada foi respondida corretamente.

O JSON de resposta é mais simples que o primeiro, tendo apelas dois atributos:

Código-fonte 4 – JSON retornado pelo serviço de inteligência

```
{
    "registerUuid": string,
    "selectedLevel": string
}
```

Sendo eles:

registerUuid: Identificador gerado para cada requisição.

selectedLevel: Nível da questão que o serviço selecionou para a requisição, representa um enumerável que pode pode assumir os valores EASY (fácil), MEDIUM (médio) ou HARD (difícil).

Não é de importância do serviço de domínio entender o comportamento do serviço de inteligência, por isso, pouco importa se a aplicação inteligente irá usar todos os atributos passados como entrada em seu algoritmo ou qual algoritmo será utilizado, o importante é apenas que o retorno da chamada obedeça o contrato.

5 RESULTADOS

Foram realizados testes para avaliar a integração do microsserviço *Eduria-core* com o microsserviço de inteligência. O Apêndice B apresenta mais detalhes sobre o o serviço de inteligência desenvolvido. Para o teste foi desenvolvido um novo microsserviço intitulado *Eduria-test*.

5.1 Testes de integração

O serviço de teste realiza uma sequência de sessenta requisições para a aplicação principal, intercalando uma chamada que busca de uma questão com outra que responde a questão buscada anteriormente. O teste é iniciado a partir de um método *POST*, onde é informado o código da matrícula do aluno no curso *(enrollmentId)* e um campo enumerável informando o padrão de comportamento.

Os padrões de comportamento, ou níveis do aluno, são modelos de formas de respostas que representam comportamentos simulados de alunos interagindo com a aplicação. Os padrões de comportamentos foram definidos como:

BEGINNER: representa um comportamento de aluno iniciante, onde o aluno acerta 4 questões iniciais e depois se mantêm com respostas intercaladas entre certas e erradas.

MEDIUM: representa um comportamento de aluno intermediário, similar ao iniciante, porém, acertando 17 questões iniciais e depois se mantêm com respostas intercaladas entre certas e erradas.

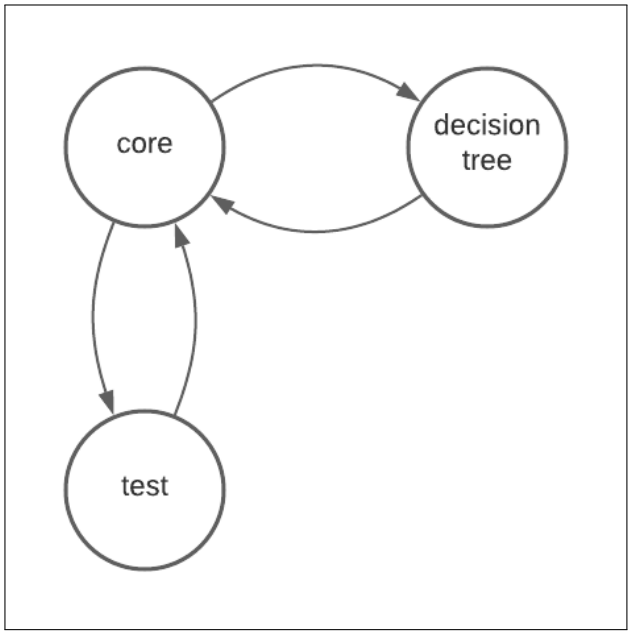
ADVANCED: representa comportamento do aluno avançado, no qual acerta todas as questões.

HOUSE ROOF: comportamento telhado de casa, representa um aluno que começa acertando todas as questões e subindo de nível, dado um certo momento, começa a errar todas as questões e a regredir.

A Figura 7 apresenta a comunicação entre os serviços. O serviço de teste dispara a requisição para a aplicação principal, a aplicação principal, por sua vez, consulta a aplicação de inteligência *eduria-decision-tree* para que esta terceira classifique o nível da questão a ser apresentada, o nível é retornado para aplicação principal e uma questão qualquer desse nível e curso é apresentada. Após apresentada, o serviço de teste reponde esta questão com base no modelo de comportamento do aluno selecionado. Esse ciclo de pergunta e resposta é repetido

trinta vezes.

Figura 7 – Relacionamento entre serviços



Fonte: Elaborado pelo próprio autor (2020).

5.2 Resultados dos testes

A aplicação *eduria-core* exibiu por linha de comando o objeto JSON enviado para aplicação de inteligência à cada iteração do teste. Para cada teste, foi cadastrado uma nova matrícula para o aluno no curso, de forma que todos os valores relacionados à desempenho começassem iguais.

As tabelas a seguir mostram o comportamento na apresentação de níveis de perguntas e resultados que são gerados a partir de cada iteração. É importante ver os valores das duas últimas colunas, pois representam o nível da questão apresentada na última consulta e se a mesma foi respondida corretamente, respectivamente. A partir disso é possível ter uma noção mais clara dos comportamentos.

Por questões de tamanho de exibição dos campos na tabela de resultados, muitos atributos tiveram os nomes reduzidos, segue abaixo uma legenda para os campos abreviados e seus respectivos significados:

req (request) : contagem da requisição, de 1 à 30.
stdId (studentId) : código do registro do estudante no banco de dados.

age (studentAge) : idade do estudante.

course (courseName) : nome do curso.

easy (easyQuestionsAnsweredCorrect) : quantidade de questões fáceis respondidas corretamente pelo aluno no curso.

medium (mediumQuestionsAnsweredCorrect) : quantidade de questões intermediárias respondidas corretamente pelo aluno no curso.

hard (hardQuestionsAnsweredCorrect) : quantidade de questões difíceis respondidas corretamente pelo aluno no curso.

qttAwr (qttAllQuestionsAnswered) : quantidade de questões respondidas pelo aluno no curso, independente de acertos ou erros.

qttC (qttAllCourseQuestions) : quantidade de questões cadastradas no curso.

score (score) : pontuação atribuída à matrícula do aluno.

lastLevel (lastQuestionLevel) : nível da última questão apresentada.

lastAwrCrt (lastQuestionWasAnsweredCorrect) : se a última questão apresentada foi respondida corretamente.

O atributo *registerUuid* apresentado no capítulo anterior, embora enviado em todas as requisições, não está apresentado nas tabelas, pois se trata apenas de uma sequência de caracteres em forma de chave identificadora da requisição, não sendo algo que influencie diretamente o comportamento.

5.2.1 Comportamento iniciante

É apresentado na Tabela 2 a tabela de nível iniciante, onde o aluno começa acertando algumas questões iniciais e após isso acerta de forma intercalada, mantendo-se com um *score* abaixo de 40.

Tabela 2 – Comportamento iniciante

req	stdId	age	course	easy	medium	hard	qttAwr	qttC	score	lastLevel	lastAwrCrt
1	1	23	IA	0	0	0	0	30	0	null	null
2	1	23	IA	1	0	0	1	30	3	EASY	true
3	1	23	IA	1	0	1	2	30	6	HARD	true
4	1	23	IA	2	0	1	3	30	9	EASY	true
5	1	23	IA	2	0	2	4	30	12	HARD	true
6	1	23	IA	3	0	2	5	30	15	EASY	true
7	1	23	IA	3	0	2	6	30	12	EASY	false
8	1	23	IA	4	0	2	7	30	15	EASY	true
9	1	23	IA	4	0	2	8	30	12	MEDIUM	false
10	1	23	IA	5	0	2	9	30	15	EASY	true
11	1	23	IA	5	0	2	10	30	12	EASY	false
12	1	23	IA	6	0	2	11	30	15	EASY	true
13	1	23	IA	6	0	2	12	30	12	EASY	false
14	1	23	IA	7	0	2	13	30	15	EASY	true
15	1	23	IA	7	0	2	14	30	12	EASY	false
16	1	23	IA	8	0	2	15	30	15	EASY	true
17	1	23	IA	8	0	2	16	30	12	EASY	false
18	1	23	IA	9	0	2	17	30	15	EASY	true
19	1	23	IA	9	0	2	18	30	12	EASY	false
20	1	23	IA	10	0	2	19	30	15	EASY	true
21	1	23	IA	10	0	2	20	30	12	EASY	false
22	1	23	IA	11	0	2	21	30	15	EASY	true
23	1	23	IA	11	0	2	22	30	12	EASY	false
24	1	23	IA	12	0	2	23	30	15	EASY	true
25	1	23	IA	12	0	2	24	30	12	EASY	false
26	1	23	IA	13	0	2	25	30	15	EASY	true
27	1	23	IA	13	0	2	26	30	12	EASY	false
28	1	23	IA	14	0	2	27	30	15	EASY	true
29	1	23	IA	14	0	2	28	30	12	EASY	false
30	1	23	IA	15	0	2	29	30	15	EASY	true

Fonte: elaborado pelo autor (2020).

5.2.2 Comportamento intermediário

Na Tabela 3 é apresentado a tabela que representa o comportamento de nível intermediário. A diferença do teste intermediário para o iniciante encontra-se em que o intermediário responde bem mais questões corretas até chegar ao ponto onde intercala respostas certas e erradas. Mantendo-se assim com um *score* abaixo de 70, porém, superior ou igual à 40.

Assim como em todas as outras simulações, é importante lembrar dos critérios da árvore de busca, onde as requisições com menos de doze questões respondidas podem ter níveis de questões sorteados de forma aleatória, como uma forma de exploração. Essa aleatoriedade ocorre sempre que a quantidade de questões respondidas for um número ímpar. Relembrando disso, pode-se ter uma observação mais atenta aos números pares menores que doze, ou todos, sem distinção, a partir de doze.

Tabela 3 – Comportamento intermediário

req	stdId	age	course	easy	medium	hard	qttAwr	qttC	score	lastLevel	lastAwrCrt
1	1	23	IA	0	0	0	0	30	0	null	null
2	1	23	IA	1	0	0	1	30	3	EASY	true
3	1	23	IA	2	0	0	2	30	6	EASY	true
4	1	23	IA	3	0	0	3	30	9	EASY	true
5	1	23	IA	4	0	0	4	30	12	EASY	true
6	1	23	IA	5	0	0	5	30	15	EASY	true
7	1	23	IA	6	0	0	6	30	18	EASY	true
8	1	23	IA	7	0	0	7	30	21	EASY	true
9	1	23	IA	7	1	0	8	30	24	MEDIUM	true
10	1	23	IA	8	1	0	9	30	27	EASY	true
11	1	23	IA	8	1	1	10	30	30	HARD	true
12	1	23	IA	9	1	1	11	30	33	EASY	true
13	1	23	IA	9	1	2	12	30	36	HARD	true
14	1	23	IA	10	1	2	13	30	39	EASY	true
15	1	23	IA	11	1	2	14	30	42	EASY	true
16	1	23	IA	11	2	2	15	30	45	MEDIUM	true
17	1	23	IA	11	3	2	16	30	48	MEDIUM	true
18	1	23	IA	11	4	2	17	30	51	MEDIUM	true
19	1	23	IA	11	5	2	18	30	54	MEDIUM	true
20	1	23	IA	11	6	2	19	30	57	MEDIUM	true
21	1	23	IA	11	6	2	20	30	54	MEDIUM	false
22	1	23	IA	11	7	2	21	30	57	MEDIUM	true
23	1	23	IA	11	7	2	22	30	54	MEDIUM	false
24	1	23	IA	11	8	2	23	30	57	MEDIUM	true
25	1	23	IA	11	8	2	24	30	54	MEDIUM	false
26	1	23	IA	11	9	2	25	30	57	MEDIUM	true
27	1	23	IA	11	9	2	26	30	54	MEDIUM	false
28	1	23	IA	11	10	2	27	30	57	MEDIUM	true
29	1	23	IA	11	10	2	28	30	54	MEDIUM	false
30	1	23	IA	11	11	2	29	30	57	MEDIUM	true

Fonte: elaborado pelo autor (2020).

5.2.3 Comportamento avançado

O comportamento de aluno avançado é o mais simples dos aqui apresentados, seus resultados refletem respostas corretas para todas as perguntas apresentadas e são apresentados na Tabela 4.

Sendo assim, pode-se notar um crescimento tanto no *score*, como também no nível das questões apresentadas ao aluno. Saindo do nível fácil, passando para o nível médio até atingir a apresentação de questões difíceis até o fim da execução.

Tabela 4 – Comportamento avançado

req	stdId	age	course	easy	medium	hard	qttAwr	qttC	score	lastLevel	lastAwrCrt
1	1	23	IA	0	0	0	0	30	0	null	null
2	1	23	IA	1	0	0	1	30	3	EASY	true
3	1	23	IA	1	0	1	2	30	6	HARD	true
4	1	23	IA	2	0	1	3	30	9	EASY	true
5	1	23	IA	2	1	1	4	30	12	MEDIUM	true
6	1	23	IA	3	1	1	5	30	15	EASY	true
7	1	23	IA	3	1	2	6	30	18	HARD	true
8	1	23	IA	4	1	2	7	30	21	EASY	true
9	1	23	IA	5	1	2	8	30	24	EASY	true
10	1	23	IA	6	1	2	9	30	27	EASY	true
11	1	23	IA	7	1	2	10	30	30	EASY	true
12	1	23	IA	8	1	2	11	30	33	EASY	true
13	1	23	IA	8	1	3	12	30	36	HARD	true
14	1	23	IA	9	1	3	13	30	39	EASY	true
15	1	23	IA	10	1	3	14	30	42	EASY	true
16	1	23	IA	10	2	3	15	30	45	MEDIUM	true
17	1	23	IA	10	3	3	16	30	48	MEDIUM	true
18	1	23	IA	10	4	3	17	30	51	MEDIUM	true
19	1	23	IA	10	5	3	18	30	54	MEDIUM	true
20	1	23	IA	10	6	3	19	30	57	MEDIUM	true
21	1	23	IA	10	7	3	20	30	60	MEDIUM	true
22	1	23	IA	10	8	3	21	30	63	MEDIUM	true
23	1	23	IA	10	9	3	22	30	66	MEDIUM	true
24	1	23	IA	10	10	3	23	30	69	MEDIUM	true
25	1	23	IA	10	11	3	24	30	72	MEDIUM	true
26	1	23	IA	10	11	4	25	30	75	HARD	true
27	1	23	IA	10	11	5	26	30	78	HARD	true
28	1	23	IA	10	11	6	27	30	81	HARD	true
29	1	23	IA	10	11	7	28	30	84	HARD	true
30	1	23	IA	10	11	8	29	30	87	HARD	true

Fonte: elaborado pelo autor (2020).

5.2.4 Comportamento house roof

A Tabela 5 mostra os resultados coletados no comportamento intitulado telhado de casa, ou *House Roof*. A partir do estado em que as questões param de serem escolhidas de forma aleatória para índices ímpares de requisição, há um aumento no nível das questões, a partir da décima nona requisição, todas as questões são respondidas de forma errada propositalmente, mostrando uma queda no nível das questões a medida que o aluno diminui seu *score*.

Tabela 5 – Comportamento telhado de casa (House Roof)

req	stdId	age	course	easy	medium	hard	qttAwr	qttC	score	lastLevel	lastAwrCrt
1	1	23	IA	0	0	0	0	30	0	null	null
2	1	23	IA	1	0	0	1	30	3	EASY	true
3	1	23	IA	1	1	0	2	30	6	MEDIUM	true
4	1	23	IA	2	1	0	3	30	9	EASY	true
5	1	23	IA	2	1	1	4	30	12	HARD	true
6	1	23	IA	3	1	1	5	30	15	EASY	true
7	1	23	IA	4	1	1	6	30	18	EASY	true
8	1	23	IA	5	1	1	7	30	21	EASY	true
9	1	23	IA	6	1	1	8	30	24	EASY	true
10	1	23	IA	7	1	1	9	30	27	EASY	true
11	1	23	IA	7	1	2	10	30	30	HARD	true
12	1	23	IA	8	1	2	11	30	33	EASY	true
13	1	23	IA	8	1	3	12	30	36	HARD	true
14	1	23	IA	9	1	3	13	30	39	EASY	true
15	1	23	IA	10	1	3	14	30	42	EASY	true
16	1	23	IA	10	2	3	15	30	45	MEDIUM	true
17	1	23	IA	10	3	3	16	30	48	MEDIUM	true
18	1	23	IA	10	4	3	17	30	51	MEDIUM	true
19	1	23	IA	10	5	3	18	30	54	MEDIUM	true
20	1	23	IA	10	5	3	19	30	51	MEDIUM	false
21	1	23	IA	10	5	3	20	30	48	MEDIUM	false
22	1	23	IA	10	5	3	21	30	45	MEDIUM	false
23	1	23	IA	10	5	3	22	30	42	MEDIUM	false
24	1	23	IA	10	5	3	23	30	39	MEDIUM	false
25	1	23	IA	10	5	3	24	30	36	EASY	false
26	1	23	IA	10	5	3	25	30	33	EASY	false
27	1	23	IA	10	5	3	26	30	30	EASY	false
28	1	23	IA	10	5	3	27	30	27	EASY	false
29	1	23	IA	10	5	3	28	30	24	EASY	false
30	1	23	IA	10	5	3	29	30	21	EASY	false

Fonte: elaborado pelo autor (2020).

5.2.5 Comportamento intermediário sem classificador

Um dos pontos citados no capítulo anterior é referente à operabilidade da aplicação principal *eduria-core* mesmo sem uma aplicação capaz de prover uma classificação para o estado atual.

Para o teste desse requisito não-funcional, foi realizado uma nova execução do modelo de comportamento do aluno intermediário, porém, dessa vez, com o serviço responsável pela árvore de decisão desligado.

É possível notar que mesmo com o serviço de inteligência desligado, a aplicação principal conseguiu realizar todas as iterações. Porém, sem o auxílio de um classificador inteligente, a aplicação escolhe o nível da questão de forma aleatória. A Tabela 6 apresenta os resultados desse teste.

Tabela 6 – Comportamento intermediário sem IA

req	stdId	age	course	easy	medium	hard	qttAwr	qttC	score	lastLevel	lastAwrCrt
1	1	23	IA	0	0	0	0	30	0	null	null
2	1	23	IA	0	1	0	1	30	3	MEDIUM	true
3	1	23	IA	1	1	0	2	30	6	EASY	true
4	1	23	IA	1	2	0	3	30	9	MEDIUM	true
5	1	23	IA	1	2	1	4	30	12	HARD	true
6	1	23	IA	1	2	2	5	30	15	HARD	true
7	1	23	IA	2	2	2	6	30	18	EASY	true
8	1	23	IA	2	2	3	7	30	21	HARD	true
9	1	23	IA	2	2	4	8	30	24	HARD	true
10	1	23	IA	2	2	5	9	30	27	HARD	true
11	1	23	IA	2	2	6	10	30	30	HARD	true
12	1	23	IA	3	2	6	11	30	33	EASY	true
13	1	23	IA	3	3	6	12	30	36	MEDIUM	true
14	1	23	IA	3	3	7	13	30	39	HARD	true
15	1	23	IA	3	3	8	14	30	42	HARD	true
16	1	23	IA	4	3	8	15	30	45	EASY	true
17	1	23	IA	5	3	8	16	30	48	EASY	true
18	1	23	IA	5	4	8	17	30	51	MEDIUM	true
19	1	23	IA	6	4	8	18	30	54	EASY	true
20	1	23	IA	6	5	8	19	30	57	MEDIUM	true
21	1	23	IA	6	5	8	20	30	54	HARD	false
22	1	23	IA	7	5	8	21	30	57	EASY	true
23	1	23	IA	7	5	8	22	30	54	EASY	false
24	1	23	IA	8	5	8	23	30	57	EASY	true
25	1	23	IA	8	5	8	24	30	54	HARD	false
26	1	23	IA	9	5	8	25	30	57	EASY	true
27	1	23	IA	9	5	8	26	30	54	MEDIUM	false
28	1	23	IA	10	5	8	27	30	57	EASY	true
29	1	23	IA	10	5	8	28	30	54	HARD	false
30	1	23	IA	10	6	8	29	30	57	MEDIUM	true

Fonte: elaborado pelo autor (2020).

6 CONCLUSÕES E TRABALHOS FUTUROS

O trabalho mostrou-se eficiente em aspectos primários e secundários. Primários que se referem a aplicação *eduria-core* em si, apresentando os resultados como sendo um serviço integrável à um outro serviço classificador externo e também sendo um serviço independente, capaz de realizar suas atividades mesmo sem a integração citada.

É possível ver o comportamento do serviço de inteligência que mantêm a árvore de decisão. Embora tendo uma estrutura simples, com objetivo de ser apenas um meio para o teste de integração da aplicação principal, viu-se que a mesma apresentou os resultados esperados.

Uma proposta de trabalho futuro é o desenvolvimento de uma aplicação de interface, ou módulo de interface, como é descrito no capítulo dois, assim também como a implementação de uma camada de autorização e autenticação na aplicação, através de *tokens* passados a cada requisição.

O principal trabalho futuro é o teste do serviço *eduria-core* com outros modelos de inteligência. O presente trabalho fez uso de arvore de decisão, mas é interessante o estudo e avaliação em outras formas, através de algoritmos de aprendizado por reforço ou outras técnicas e algoritmos de inteligencia computacional.

O projeto foi documentado em um site para acesso mais fácil e abrangente por outros desenvolvedores e pesquisadores. O site está disponível em: https://vanderloureiro.github.io/eduria e o Apêndice C apresenta uma captura de tela desta documentação online.

REFERÊNCIAS

AKKILA, A. N.; ALMASRI, A.; AHMED, A.; AL-MASRI, N.; SULTAN, Y. S. A.; MAHMOUD, A. Y.; ZAQOUT, I. S.; ABU-NASER, S. S. Survey of intelligent tutoring systems up to the end of 2017. In: . [*S. l.*]: IJARW, 2019.

AL-SHAWWA, M. O.; ALSHAWWA, I. A.; ABU-NASER, S. S. An intelligent tutoring system for learning java. IJARW, 2019.

ALSHAWWA, I. A.; AL-SHAWWA, M. O.; ABU-NASER, S. S. An intelligent tutoring system for learning computer network ccna. IJARW, 2019.

BAKEER, H. M. S.; ABU-NASER, S. S. An intelligent tutoring system for learning toefl. 2019.

ERÜMIT, A. K.; ÇETIN, İ.; KOKOÇ, M.; KÖSA, T.; NABIYEV, V.; AYGÜN, E. S. Designing a usability assessment process for adaptive intelligent tutoring systems: A case study. **Turkish Online Journal of Qualitative Inquiry**, v. 10, n. 1, p. 141–179, 2019.

FALKEMBACH, G. A. M. Concepção e desenvolvimento de material educativo digital. **RENOTE-Revista Novas Tecnologias na Educação**, v. 3, n. 1, 2005.

FERREIRA, A.; NICACIO, J.; FERREIRA, G.; FILHO, G. S.; RODRIGUES, V. Em busca de uma api rest para um sistema acadêmico de terceiros. In: SBC. **Anais do XVIII Escola Regional de Computação Bahia, Alagoas e Sergipe**. [*S. l.*], 2018. p. 153–158.

FOWLER, M.; LEWIS, J. Microservices a definition of this new architectural term. **URL: http://martinfowler. com/articles/microservices. html**, p. 22, 2014.

GUELPELI, M. V. C.; OMAR, N.; RIBEIRO, C. H. C. Aprendizado por reforço para um sistema tutor inteligente sem modelo explícito do aprendiz. **Revista Brasileira de Informática na Educação**, v. 12, n. 2, p. 69–77, 2004.

JSON.ORG. **ECMA-404 The JSON Data Interchange Standard.** 2015. Disponível em: http://json.org/json-pt.html.

MARTINS, W.; AFONSECA, U. R.; NALINI, L. E.; GOMES, V. M. Tutoriais inteligentes baseados em aprendizado por reforço: concepção, implementação e avaliação empírica. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [*S. l.*: *s. n.*], 2007. v. 1, n. 1, p. 550–559.

MENDES, P. Microsserviços, por martin fowler e james lewis. **http://www.pedromendes.com.br/2016/01/02/microservicos/**, 2016.

POLSON, M. C.; RICHARDSON, J. J. **Foundations of intelligent tutoring systems**. [*S. l.*]: Psychology Press, 2013.

QUINLAN, J. R. Learning decision tree classifiers. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 28, n. 1, p. 71–72, 1996.

SATO, L. Y.; SHIMABUKURO, Y. E.; KUPLICH, T. M.; GOMES, V. C. F. Análise comparativa de algoritmos de árvore de decisão do sistema weka para classificação do uso e cobertura da terra. **XVI Simpósio Brasileiro de Sensoriamento Remoto**, p. 2353–2360, 2013.

SCHAAB, B. L.; DUARTE, M.; AZEVEDO, O. B.; CRUZ, D. V. A. da; JAQUES, P. A. Aplicação do mindfulness em um sistema tutor inteligente: um estudo piloto. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [*S. l.*: *s. n.*], 2015. v. 26, n. 1, p. 1072.

SPRING. **Spring Boot**. Spring, 2018. Disponível em: https://spring.io/projects/spring-boot.

VALERIANO, E.; CORRÊA, A.; POZZEBON, E. O sistema tutor inteligente mazk no ensino fundamental i. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [*S. l.*: *s. n.*], 2019. v. 30, n. 1, p. 616.

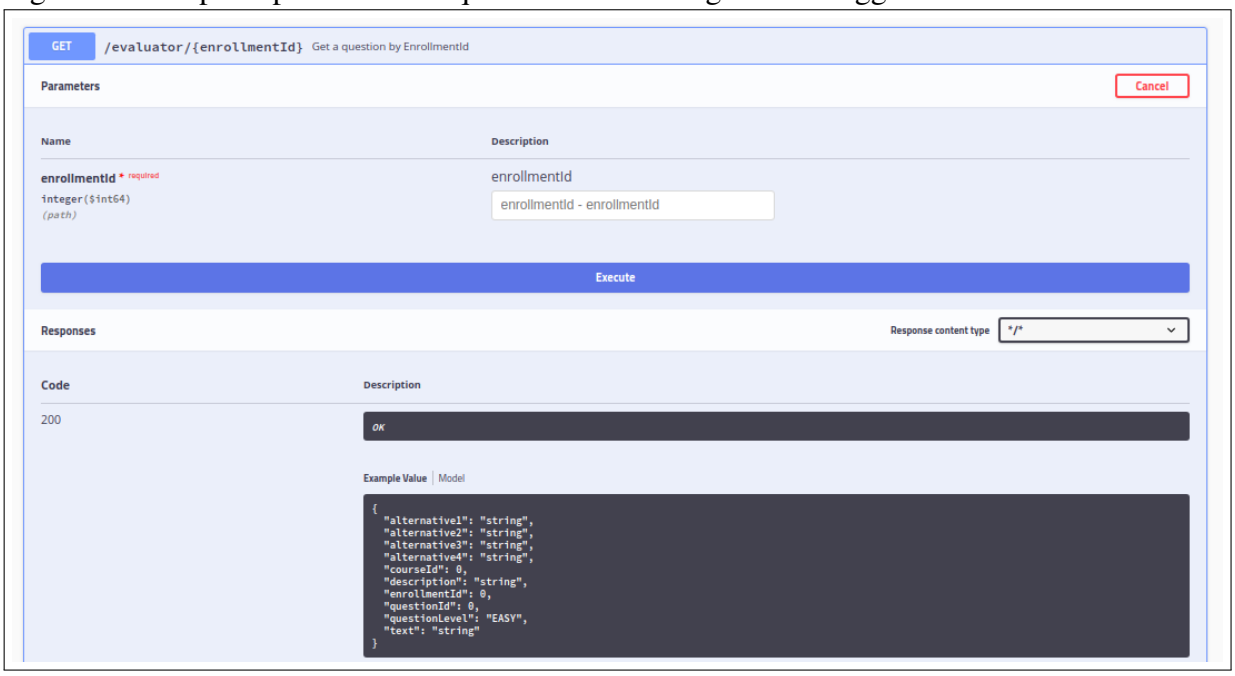
VARGAS, A. P.; SANTOS, R.; BOTELHO, S. S. da C.; TONIN, N.; BEZ, J. Um sistema de recomendação baseado em um modelo cognitivo de aprendizagem. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [*S. l.*: *s. n.*], 2017. v. 28, n. 1, p. 1667.

W3C. **Pattern: Microservice Architecture**. 2004. Disponível em: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest.

XU, Z.; WIJEKUMAR, K.; RAMIREZ, G.; HU, X.; IREY, R. The effectiveness of intelligent tutoring systems on k-12 students' reading comprehension: A meta-analysis. **British Journal of Educational Technology**, Wiley Online Library, v. 50, n. 6, p. 3119–3137, 2019.

APÊNDICE A – ENDPOINT PARA BUSCA DE QUESTÃO

Figura 8 – Endpoint para busca de questão na inteface gráfica Swagger



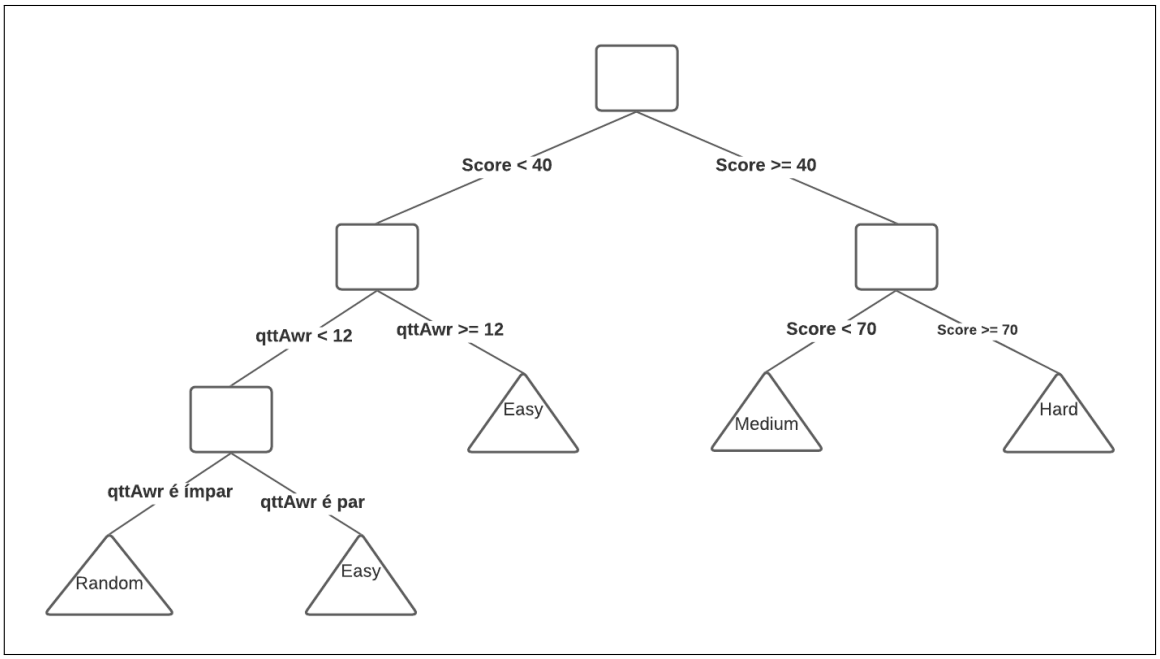
Fonte: Elaborado pelo próprio autor (2020).

APÊNDICE B – *SERVIÇO DE INTELIGÊNCIA*

Para realização de testes e avaliação da integração com o microsserviço principal que é o foco de estudo, foi desenvolvido um serviço de inteligência padrão. O serviço de inteligência é uma abstração para o módulo de diagnóstico apresentado no Capítulo 2. O serviço intitulado *eduria-decision-tree* foi desenvolvido com o uso de uma árvore de decisão para escolhas de níveis.

Por não ser o foco deste trabalho, e sendo apenas um meio para atingir a finalidade de testar a avaliação principal, a árvore de decisão foi construída de forma simples, com poucos níveis para busca. Todos os atributos do objeto JSON foram passados, mas apenas dois foram utilizados: *score* e *qttAllQuestionsAnswered*. Esse segundo está representado na árvore a seguir como *qttAwr*.

Figura 9 – Árvore de decisão em serviço de inteligência

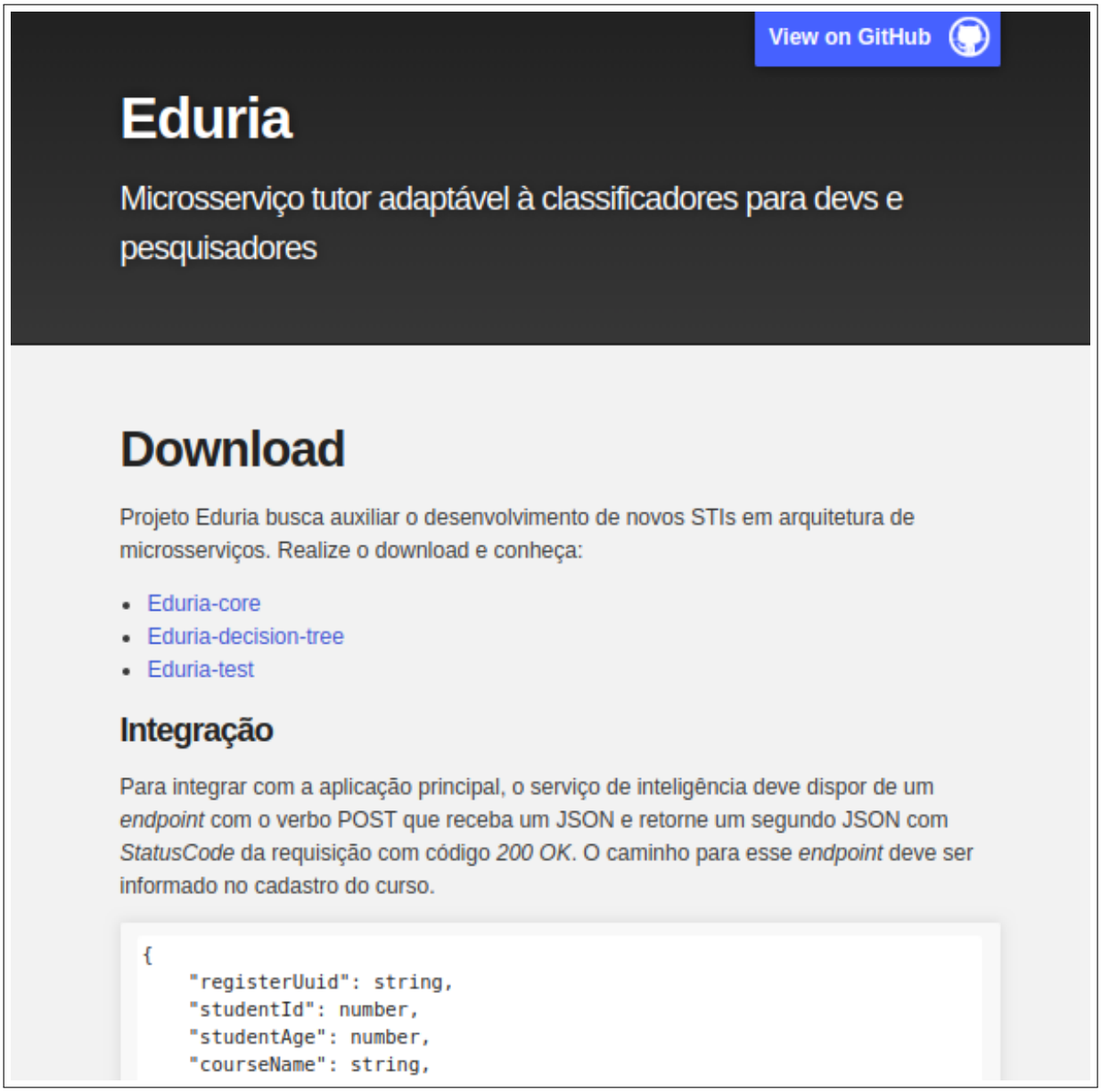


Fonte: Elaborado pelo próprio autor (2020).

Se o atributo *qttAllQuestionsAnswered* tiver um valor abaixo de doze, significa que o aluno acabou de realizar a matrícula no curso, e por isso, ainda não respondeu muitas questões. Foi definida a estratégia de sortear níveis aleatórios de questões para os alunos iniciantes, como uma forma de exploração. Essa estratégia ocorre apenas quando o *score* está abaixo de quarenta e a quantidade de questões respondidas for um número ímpar e abaixo de doze.

APÊNDICE C – SITE DE DOCUMENTAÇÃO DO PROJETO

Figura 10 – Site de documentação do projeto



Fonte: Elaborado pelo próprio autor (2020), disponível em: https://vanderloureiro.github.io/eduria