DropoutGuard: Especificação de Arquitetura

Eduardo Fillipe da Silva Reis[1]

[1]Departamento de Computação – Universidade Federal de Sergipe (UFS)
São Cristóvão – SE – Brasil

eduardo.reis@dcomp.ufs.br

Abstract. *School dropout in higher education is a problem that affects educational institutions worldwide. This document aims to specify the architecture of a decision support system named DropoutGuard. The system can be used by educational institutions to predict students at risk of dropping out of their respective courses, thus identifying causes and preparing strategies and actions that enable the retention of these students.*

Resumo. *A evasão escolar no ensino superior é um problema que afeta instituições de ensino em todo o mundo. Assim, este documento tem como objetivo especificar a arquitetura de um sistema de apoio à decisão, nomeado DropoutGuard. O sistema, então, poderá ser utilizado por instituições de ensino para prever alunos em risco de evasão de seus respectivos cursos, podendo assim identificar causas e preparar estratégias e ações que possibilitem a retenção desses alunos.*

1. Introdução

A evasão no ensino superior é tema de debate em todo o mundo e que acontece tanto em países desenvolvidos quanto em desenvolvimento, segundo dados do NCES [NCES 2020]. Além disso, é um problema que atinge instituições de ensino públicas e privadas, mostrando-se um obstáculo a ser superado por toda a sociedade.

É necessário que as instituições de ensino atuem de forma ativa afim de reduzir os níveis de evasão no sistema educacional brasileiro. Assim, uma das possíveis abordagens é investir em estratégias que antecipem o problema, identificando individualmente quais estudantes têm maior tendência a abandonar o curso ou a instituição, antes que isso ocorra.

Dessa forma, este projeto objetiva especificar a arquitetura de um sistema de software capaz de auxiliar instituições de ensino na tarefa de detectar alunos sob risco de evasão, denominado DropoutGuard. Para isso, este trabalho irá se basear na monografia ”Uso de Redes Bayesianas Multinível para prever a evasão estudantil no Departamento de Computação da Universidade Federal de Sergipe”[Reis 2023] que descreve um conjunto de abordagens que torna possível prever a probabilidade de evasão de alunos da UFS através do uso de Redes Bayesianas Multinível [Lappenschaar et al. 2013]. Além disso, a monografia explica como adquirir os dados relevantes para realizar essas previsões, fornecendo uma base para a implementação do sistema proposto. No entanto, não é escopo deste trabalho determinar as tecnologias exatas a serem utilizadas na implementação do sistema, como linguagens de programação, bibliotecas de software ou frameworks. As sugestões apresentadas aqui devem ser vistas como recomendações e podem ser adaptadas conforme necessário durante a futura implementação, considerando também a opinião e a experiência da equipe responsável.

Por fim, embora [Reis 2023] se limite à um escopo departamental, este documento visa propor um sistema que seja extensível em duas vertentes. Primeiro, o sistema deve ser capaz de atender toda a instituição, ampliando seu escopo para além do Departamento de Computação. Em segundo lugar, a aplicação deve ser expansível, permitindo que novos modelos ou atualizações de modelos de apoio à decisão existentes sejam facilmente integrados e utilizados pela comunidade acadêmica, permitindo a aplicação simplificada de trabalhos desenvolvidos dentro da Universidade.

2. Especificação de Requisitos

Esta seção visa listar os principais requisitos funcionais e não funcionais da aplicação descrita neste projeto. Também será apresentado o diagrama de Casos de Uso afim de demonstrar as possíveis interações entre os usuários e o sistema.

2.1. Requisitos Funcionais

1. O sistema deve prover uma interface gráfica que permite fácil acesso ao sistema.
2. A interface de usuário deve exigir que o usuário se identifique e retorne um erro caso o usuário não possua autorização para acessar o sistema.
3. A interface de usuário deve permitir que o usuário carregue um PDF contendo o histórico de um aluno emitido pelo departamento. Os dados dos históricos enviados devem ser extraídos e então analisados pelos mecanismos de apoio a decisão.
4. A interface de usuário deve permitir que o usuário visualize o histórico de relatórios gerados por ele.
5. O sistema deve permitir que professores e gestores consigam acessar relatórios sobre a probabilidade de evasão de um grupo de alunos ou de um aluno específico dado o seu histórico acadêmico.
6. O sistema deve armazenar os relatórios gerados e dados utilizados para tal, anonimizando os dados utilizados como entrada, inviabilizando a identificação de alunos.
7. O sistema deve permitir apenas que usuários autenticados e autorizados dentro do domínio da instituição acessem suas funcionalidades.
8. O sistema deve manter o histórico de ações tomadas por usuários gestores para fins de auditoria. Os mecanismos de apoio a decisão do sistema devem utilizar dados dos alunos matriculados na instituição para montar sua base de conhecimento.
9. O sistema deve permitir que os dados utilizados nos processos de previsão sejam atualizados sempre que necessário.
10. Os gestores do sistema devem ser capaz de solicitar atualização dos dados utilizados pelos mecanismos de apoio a decisão do sistema.
11. Os mecanismos de apoio a decisão internos devem sempre utilizar a versão mais atualizada dos dados.
12. Os dados utilizados pelos mecanismos de apoio a decisão devem ser anonimizados, impedindo a identificação de alunos envolvidos.

2.2. Requisitos Não Funcionais

1. O sistema deve utilizar o Google como provedor de autenticação e autorização
2. Os usuários autorizados a acessar o sistema devem estar no domínio *@dcomp.ufs.br*

3. O sistema deve autenticar o usuário com Single Sign-On utilizando o protocolo OAuth2 [Hardt 2012] e extendendo o processo de autorização através do protoloco OpenID Connect [Sakimura 2014].
4. O sistema deve permitir que os usuários possuam, inicialmente, dois papéis: gestor e professor
5. Inicialmente, o sistema deve prover uma interface WEB, que pode ser estendida para outras tecnologias posteriormente.
6. O sistema deve ser extensível quanto aos algoritmos utilizados para realizar a previsão de evasão estudantil.
7. O sistema deve ser flexível quanto ao número de usuários e tempo de resposta dos algoritmos de apoio a decisão, permitindo que solicitações sejam enfileiradas e que as operações de inferência e treinamento dos modelos utilizados sejam realizados de maneira assíncrona.
8. O sistema deve implementar, inicialmente, um mecanismo de apoio a decisão baseado em Redes Bayesianas Multinível, mas deve ser flexível à adição de novos modelos.

2.3. Diagrama de Casos de Uso

A Figura 1 apresenta o diagrama de casos de uso do sistema. Observam-se dois atores: um *Professor*, interessado em visualizar os índices de probabilidade de um determinado aluno, e um *Administrador*, que também é professor, mas que, no sistema, poderá realizar tarefas de atualização e configuração do sistema de apoio à decisão.

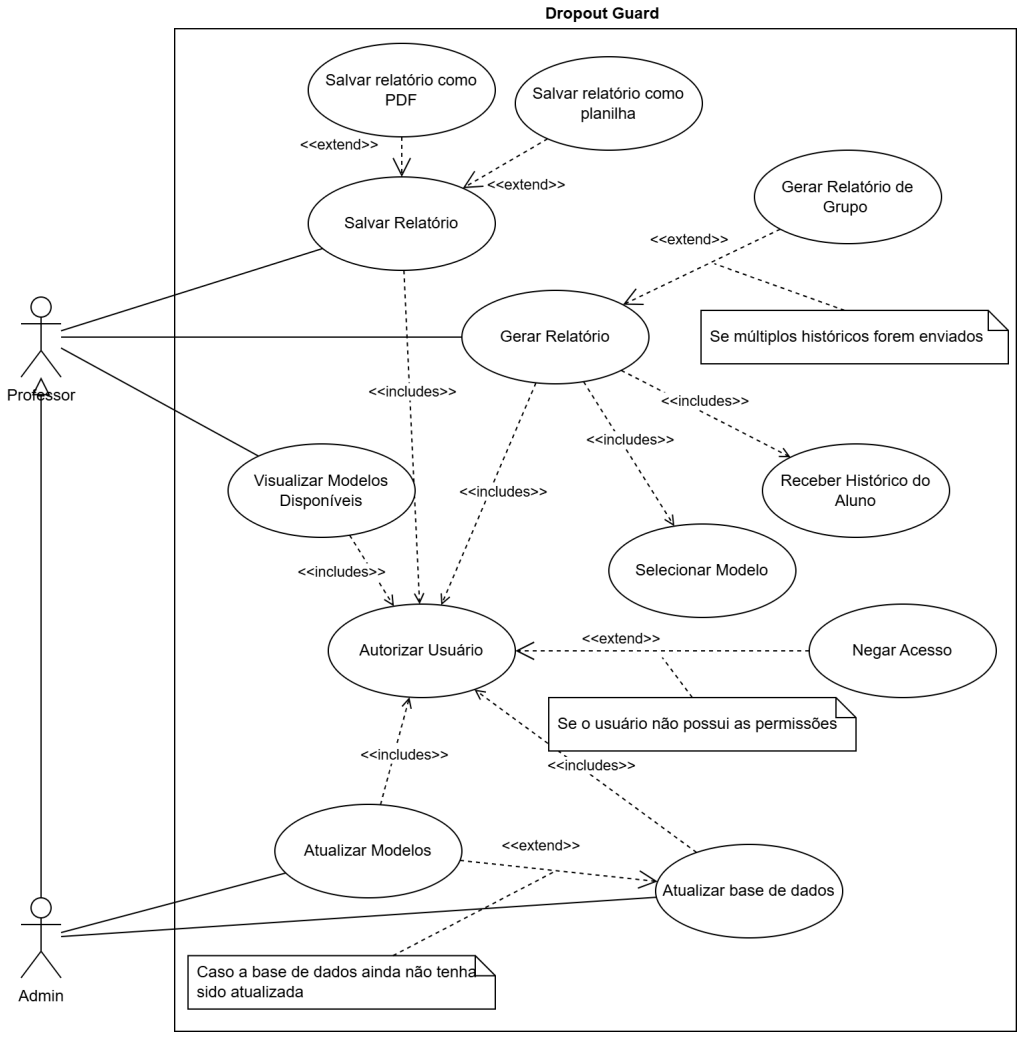
O caso de uso "Autorizar Usuário"é central para o sistema por dois motivos. Primeiro, porque os usuários têm a capacidade de enviar dados sensíveis ao sistema e solicitar atualizações de configurações que alteram seu comportamento. Segundo, as previsões realizadas pelo sistema de apoio podem ser consideradas sensíveis quando analisadas em larga escala. Portanto, é necessário que as interações entre os usuários e o sistema possuam um mecanismo de autenticação e autorização para garantir a segurança dos dados e dos usuários.

O caso de uso "Gerar Relatório"serve como a interface de comunicação entre o Professor e o sistema de apoio à decisão, através do qual os processos de inferência são executados. Por meio desse caso de uso, o professor obterá um relatório em um formato legível por humanos, como PDF, contendo o resultado da análise realizada sobre a base de dados do sistema de apoio à decisão.

O caso de uso "Visualizar Modelos Disponíveis"refere-se à capacidade do usuário de escolher qual modelo de apoio à decisão utilizará durante a inferência. Idealmente, ao fazer essa escolha, o usuário deve focar na eficiência e eficácia do modelo, em vez de se preocupar com a técnica aplicada. Portanto, todos os modelos disponíveis devem apresentar informações que auxiliem o usuário na escolha, como acurácia e tempo de inferência, por exemplo.

Por fim, os casos de uso *Atualizar Modelos* e *Atualizar Base de Dados* são as duas funções que um *Administrador* estende de um *Professor*. Esses casos de uso são necessários para que seja possível atualizar a base de dados conforme a passagem dos períodos letivos e atualizar os modelos utilizados com esses novos dados. Idealmente, esse é um trabalho realizado apenas ao final de cada ciclo escolar, mas é caro em termos

Figura 1. Diagrama dos principais casos de uso da aplicação



de processamento computacional.

3. Especificação Arquitetural

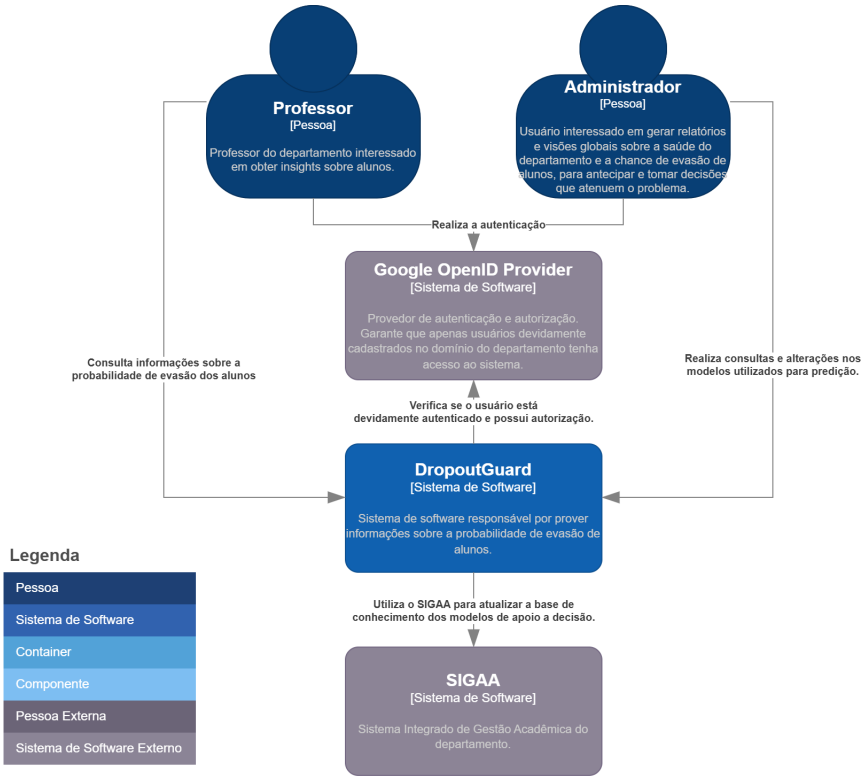
Esta seção visa expor a especificação de arquitetura desenvolvida para atender as necessidades do DropoutGuard no modelo C4[Ford and Richards 2019]. Este modelo visa simplificar o processo de especificação, tornando-o menos prolixo e mais fácil de entender. Dessa forma, busca-se criar documentações concisas que possam delinear os principais pontos do sistema, permitindo que pessoas de diferentes áreas e níveis de conhecimento compreendam mais facilmente o escopo e a arquitetura do sistema, com foco na resolução do problema. Em um ambiente diverso como uma universidade, essa característica é altamente desejável, pois incentiva discussões entre pessoas com diferentes perspectivas, ampliando o leque de soluções possíveis.

3.1. Diagrama de Contexto

Do ponto de vista do usuário o Dropout Guard é uma aplicação Web que centraliza as capacidades de extrair relatórios sobre a saúde escolar de alunos. Os únicos sistemas

externos com os quais existe comunicação é o Google, utilizado para autenticar e autorizar os usuários e o SIGAA utilizado como plataforma para obter dados sobre a performance dos estudantes. A Figura 2 mostra a relação entre esses usuários e sistemas.

Figura 2. Diagrama de Contexto da aplicação DropoutGuard



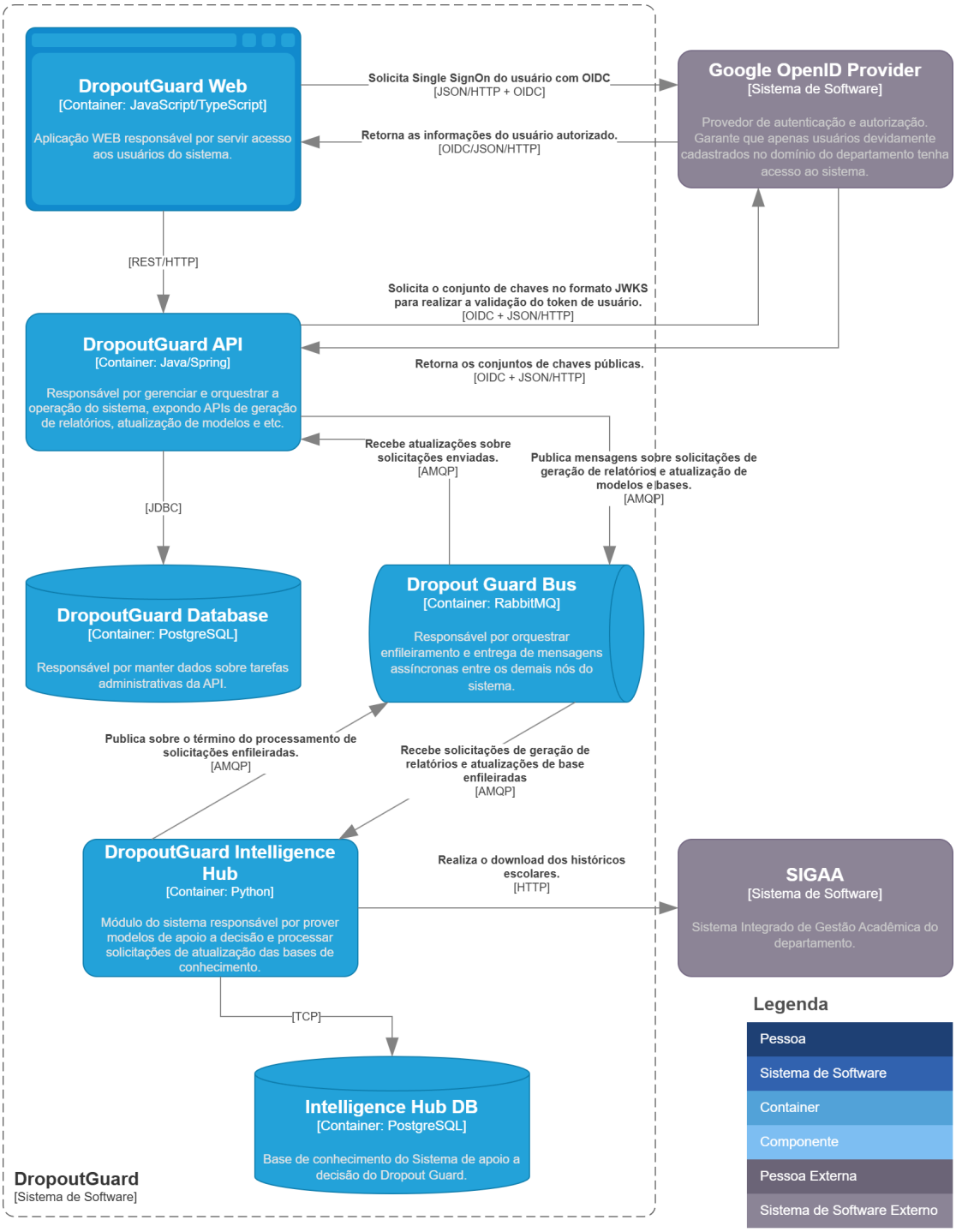
3.2. Diagrama de Contêineres

A Figura 3 expõe o Diagrama de Contêineres do sistema proposto. Por sua vez, os contêineres apresentados foram divididos em dois tipos: Centrais e Periféricos, de acordo com a sua responsabilidade.

3.2.1. Contêineres Centrais

Os requisitos funcionais e não funcionais apresentados anteriormente descrevem duas grandes facetas que devem ser previstas na arquitetura do sistema. A primeira faceta é administrativa, o DropoutGuard pode ser visto como um catálogo de modelos capazes de construir relatórios a partir da performance acadêmica de estudantes, onde cada um desses modelos pode ser configurado e executado de diferentes maneiras. A segunda faceta diz respeito à execução desses modelos e como mantê-los atualizados, tendendo a ser computacionalmente custosa, pois envolve o treinamento e a execução de modelos de aprendizado de máquina.

Figura 3. Diagrama de contêineres da aplicação



Por isso, a arquitetura especificada neste documento foi distribuída em dois grandes módulos, que serão executados em contêineres diferentes. Um módulo responsável pelas configurações de modelos e solicitações de processos, chamado *DropoutGuard API*, e outro módulo, chamado *DropoutGuard Intelligence Hub*, responsável pelo processamento das requisições realizadas pelo usuário no primeiro módulo.

Distribuir o sistema dessa maneira traz alguns benefícios:

1. **Separação de contextos**: Enquanto os sub-módulos da *DropoutGuard API* se preocuparão com fatores administrativos, os sub-módulos do *Intelligence Hub* focarão na execução eficaz e eficiente de inferência e treinamento de modelos.
2. **Flexibilidade quanto ao uso de tecnologias**: Como os módulos resolvem partes diferentes do sistema, é possível evoluir ambos com tecnologias diferentes, empregando aquelas mais adequadas para cada caso. Isso também facilita a colaboração entre grupos de pessoas com conhecimentos variados, permitindo que trabalhem juntos, mas ainda assim em suas zonas de conforto.
3. **Flexibilidade quanto à escalabilidade**: Durante o uso normal do sistema, o *Intelligence Hub* será o módulo que mais consumirá capacidade computacional. Ao distribuir o sistema fisicamente, é possível adicionar mais nós computacionais para processar as requisições de inferência e atualização de modelos, conforme a necessidade.

A comunicação entre esses módulos seguirá os estilos Cliente-Servidor e Publisher-Subscriber. Dependendo da ação realizada, os módulos trocarão de papéis, de forma que o responsável pelo contexto sempre será o servidor. A comunicação poderá ser tanto síncrona quanto assíncrona, a depender da necessidade.

1. **Comunicação Síncrona:** Quando os módulos precisarem fazer consultas sobre o estado do outro ou realizar tarefas de curto prazo e baixo custo computacional.
2. **Comunicação Assíncrona:** Via de regra, toda comunicação que parte do *DropoutGuard API* para o *Intelligence Hub* será assíncrona, pois serão solicitações de trabalho de longo prazo ou de alto custo computacional. Para que essa comunicação ocorra, será utilizado o padrão *Publisher-Subscriber*, com um canal de comunicação por assunto, utilizando um *broker* de mensagens como middleware que garanta a entrega e a integridade da comunicação. As mensagens trocadas entre os módulos serão Eventos de Domínio[Richardson 2018]. É recomendado que o middleware utilizado possua alta disponibilidade, pois, em caso de falha desse componente, toda a aplicação falha. Sugere-se a utilização do RabbitMQ, que permite a implantação em cluster, garantindo alta disponibilidade, persistência e replay de mensagens.

3.2.2. Contêineres Periféricos

Além dos contêineres que comportam os dois principais módulos do sistema, a Figura 3 também mostra o contêiner *DropoutGuard Web*, que interage com o usuário final, e os contêineres de bancos de dados. Alterações nesses contêineres não devem causar impacto sobre as regras de negócio e interfaces já estabelecidas nos contêineres centrais. Essa característica isola os contêineres centrais dos periféricos, protegendo as regras de negócio contra possíveis alterações no modelo de dados ou na interface do usuário, por exemplo.

Esses contêineres não serão especificados neste documento, pois, idealmente, a implementação da interface de usuário deve ser o mais leve possível, respeitando as APIs dos contêineres/módulos centrais. Da mesma forma, as bases de dados devem agir como repositórios que respondem às consultas especificadas nas camadas de domínio dos contêineres/módulos centrais, independentemente de serem relacionais ou não relacionais, por exemplo.

3.3. Diagramas de Componentes

3.3.1. DropoutGuard API

A Figura 4 mostra as relações entre componentes da aplicação. Os componentes de autorização verificam a validade do token OAuth2. Os 'Controladores' atuam como a camada de apresentação. O 'Use Case Component' orquestra as funcionalidades do sistema, dependendo de abstrações do domínio, como 'FileStorageComponent', 'MessagePublisherComponent', 'MessageListenerComponent' e 'RepositoryComponent'.

3.3.2. IntelligenceHub

Por sua vez, a Figura 5 mostra o diagrama de Componentes do Contêiner *IntelligenceHub*. A nomeclatura dos componentes também se baseia nos padrões sugeridos por [Martin 2017]. Entretanto, não há uma camada de *Controladores*, como no Contêiner *DropoutGuard Api*. Como a comunicação entre esses dois contêineres é assíncrona, os pontos de entradas do IntelligenceHub são *listeners* de filas que entregam atualizações sobre as entidades e agregações do domínio.

3.4. Limitações

Um dos principais possíveis pontos de falha do sistema do sistema descrito é o Broker de mensagens que atua para comunicar os contêineres DropoutGuardApi e Intelligence Hub. No caso de indisponibilidade desse broker toda a aplicação irá parar de funcionar. Como já mencionado anteriormente, uma das formas de contornar e reduzir esse risco é utilizar brokers resilientes e dotados de alta disponibilidade. Entretanto, esse problema ainda pode vir a ocorrer.

Uma forma de contornar essa situação é aplicar o padrão *Outbox* [Richardson 2018]. Dessa forma, utilizando esse padrão, mesmo que o broker esteja indisponível, o sistema continuará em pleno funcionamento porque as mensagens serão enviadas quando o broker ficar disponível novamente.

4. Considerações Finais

De acordo com o conjunto de diagramas apresentados, é possível afirmar que a arquitetura especificada neste documento é capaz de atender os requisitos funcionais e não funcionais da aplicação DropoutGuard. Cabe ressaltar a escalabilidade e resiliência da arquitetura sugerida, que se utiliza de mecanismos de comunicação síncrona e assíncrona para garantir a integridade e a eficiência das operações, através de estilos e padrões bem definidos na literatura.

Sobre o modelo de documentação utilizado, C4 [Ford and Richards 2019], é possível afirmar que ele se demonstrou uma ferramenta versátil e que permite o entendimento e a evolução da documentação do sistema de maneira simplificada, tanto por arquitetos quanto por programadores. Por outro lado, a relação intrínseca entre o modelo e os padrões e tecnologias utilizadas na arquitetura pode torná-lo menos acessível a leitores menos experientes, uma vez que alguns detalhes da arquitetura se tornam implícitos.

Figura 4. Diagrama de componentes do contêiner DropoutGuard API

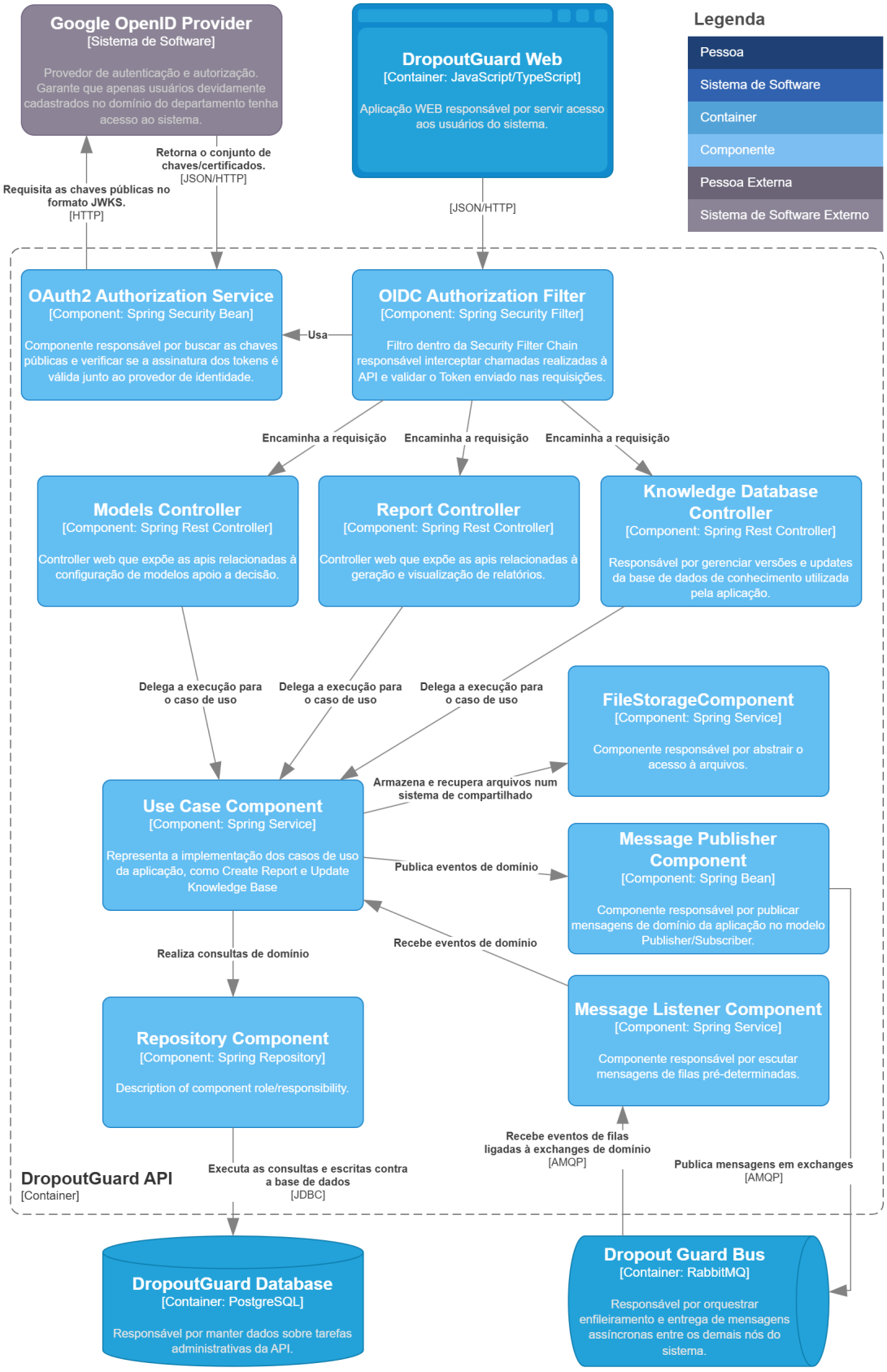
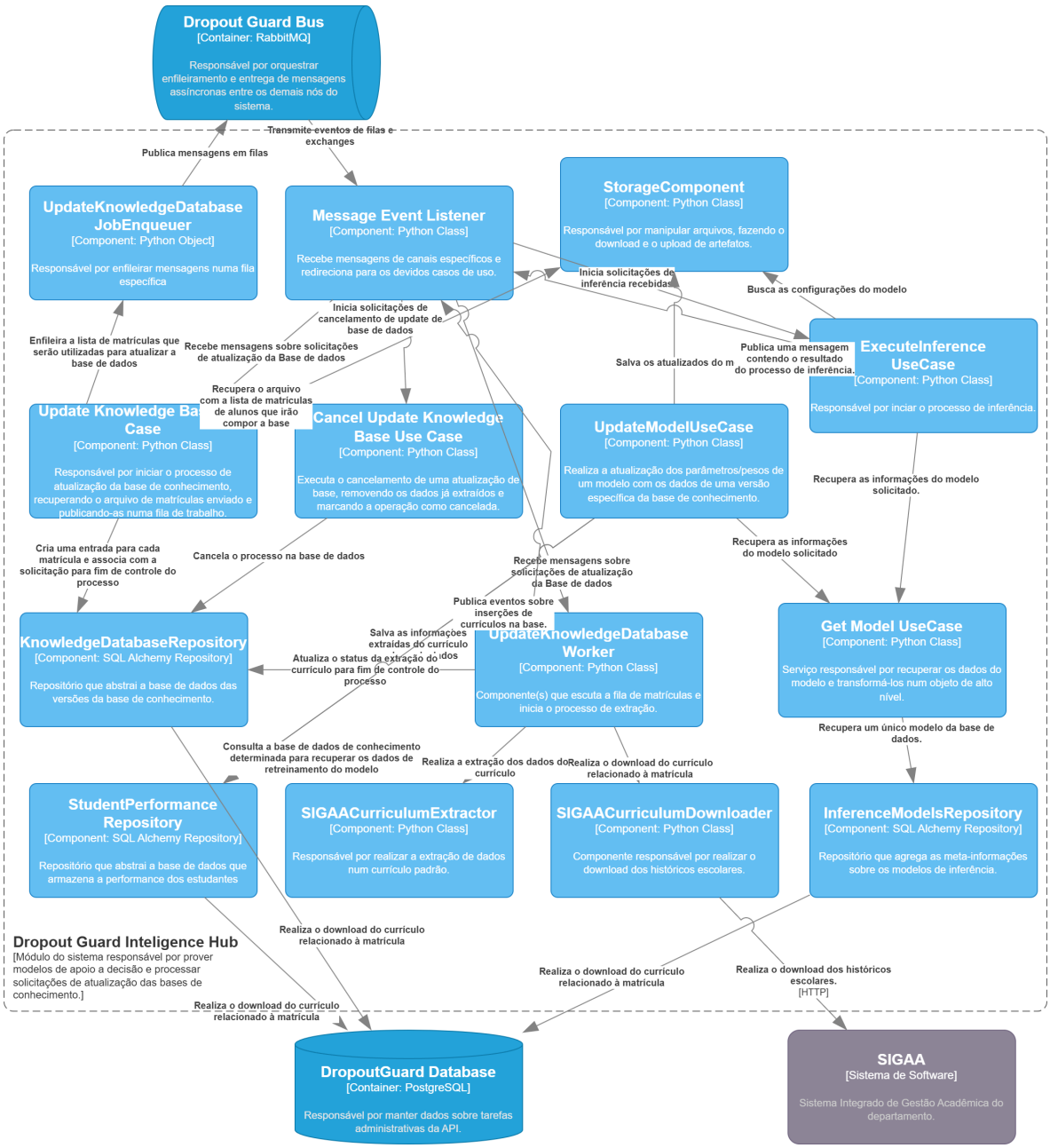


Figura 5. Diagrama de componentes do contêiner Intelligence Hub



Assim, faz-se necessário verificar a validade dessa afirmação quando os leitores não possuem conhecimento e/ou experiência técnica suficiente para identificar e reconhecer como os componentes dos diagramas irão se traduzir durante a implementação do sistema.

Referências

Ford, N. and Richards, M. (2019). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, Sebastopol, CA.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749.

Lappenschaar, M., Hommersom, A., Lucas, P. J., Lagro, J., and Visscher, S. (2013). Multilevel bayesian networks for the analysis of hierarchical health care data. *Artificial Intelligence in Medicine*, 57:171–183.

Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, Boston, MA.

NCES (2020). What are the dropout rates of high school students?

Reis, E. F. S. (2023). Uso de redes bayesianas multinível para prever a evasão estudantil no departamento de computação da universidade federal de sergipe.

Richardson, C. (2018). *Microservices Patterns: With Examples in Java*. Manning Publications, Shelter Island, NY.

Sakimura, N. (2014). Openid protocol. `https://openid.net/specs/openid-connect-core-1_0.html`. Acesso em 14 de novembro de 2024.