MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

SOFTWARE PARA GERENCIAMENTO ESTUDANTIL BASEADO EM MICRO SERVIÇOS

Lucas Alves de Faria

Orientador: Prof. Dr. Bruno Ferreira

FORMIGA- MG
2018

LUCAS ALVES DE FARIA

SOFTWARE PARA GERENCIAMENTO ESTUDANTIL BASEADO EM MICRO SERVIÇOS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Minas Gerais - *Campus* Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Bruno Ferreira.

FORMIGA- MG
2018

Faria, Lucas Alves de.
004 Software Para Gerenciamento Estudantil Baseado em Micro Serviços / Lucas Alves de Faria . -- Formiga : IFMG, 2018.
84p. : il.

Orientador: Prof. Dr. Bruno Ferreira.
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Angular. 2. Spring 3. Micro-Serviços.4. EducaSys. 5. Website. I. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Me..Naliana Dias Leandro CRB6-1347

LUCAS ALVES DE FARIA

DESENVOLVIMENTO DE WEBSITE PARA GERENCIAMENTO ESCOLAR COM MICRO SERVIÇOS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais-Campus Formiga, como Requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: 05 de junho de 2018.

BANCA EXAMINADORA

Prof.º Bruno Ferreira

Prof.º Fernando Paim Lima

Prof.º Manoel Pereira Júnior

AGRADECIMENTOS

Agradeço primeiramente a Deus e aos meus pais que me deram forças para continuar nessa longa jornada.

Ao meu orientador por compartilhar seus conhecimentos e tornar esta tarefa menos árdua.

A todos os professores pelo empenho e incentivo para desenvolver novas ideias e fundamentá-las.

Sou grato a todos amigos que sempre estiveram presente para auxiliar todas minhas decisões e tornar os momentos difíceis superáveis.

Depois de muito trabalho árduo, erros e acertos, foi possível superar todas as dificuldades devido ao empenho e apoio prestado por todos que estiveram presentes nessa caminhada.

RESUMO

Este projeto tem como proposta o desenvolvimento de um *software* que possa validar a utilização da arquitetura de micro serviços. Esta arquitetura visa a divisão de uma aplicação em um conjunto de módulos que podem funcionar de forma independente, proporcionando assim uma maior modularidade e persistência da aplicação. Para validar esta arquitetura de *software* foi desenvolvido um site denominado EducaSys que tem como intuito proporcionar uma experiência simples, eficaz e intuitiva para seu usuário, baseando-se em uma concepção de aplicação diferente dos modelos monolíticos já conhecidos. Sua ideia partiu do princípio que poucas instituições de ensino públicas do estado de Minas Gerais possuem uma ferramenta que possibilita tanto alunos quanto professores compartilhar informações essenciais como número de faltas, notas e conteúdo para estudo. Além de suas funcionalidades, o *software* tem como intuito utilizar tecnologias atuais que possam maximizar sua utilidade e eficácia. Dentre as tecnologias utilizadas é necessário destacar a importância de alguns *frameworks* que foram cruciais para o bom funcionamento do mesmo. Através de *frameworks* como Angular 5, Spring Framework e Bootstrap foi possível modularizar as partes do projeto. Isso fez com que cada uma pudesse ser analisada separadamente e tratada de uma forma mais específica e usual. A aplicação pode vir a se mostrar uma opção útil e viável, uma vez que nenhuma instituição pública estadual de ensino da região utiliza sistemas similares. Esse projeto pode agregar diversos benefícios tanto para funcionários quanto para alunos.

Palavras-chave: *Angular*.*Spring*.Micro Serviços.EducaSys.*Website*

ABSTRACT

This project aims to develop a software that can validate the use of the microservices architecture. This architecture aims to divide an application into a set of modules that can work independently, providing greater modularity and application persistence. To validate this software architecture a website called EducaSys that aims to provide a simple, effective and intuitive experience for its user was developed based on a different application concept of monolithic models already known. its idea was to assume that a few public education institutions in the state of Minas Gerais have a tool that allows both students and teachers to share essential information such as number of absences, grades, and content to study. In addition to its functionalities, the software intends to use current technologies that can maximize its usefulness and effectiveness. Among the technologies used it is necessary to highlight the importance of some frameworks that were crucial for the proper functioning of it. Through frameworks like Angular 5, Spring Framework and Bootstrap it was possible to modularize the parts of the project. This make each one of them being analyzed separately and treated in a more specific and usual way. The application may prove to be a useful and viable option, since no public state educational institution in the region uses similar systems. This project can add several benefits for both employees and students.

Keywords: *Angular*.*Spring*.*Microservices*.EducaSys.*Website*

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CRUD	*Create, Read, Update e Delete*
CSS	*Cascading Style Sheets*
FOSS	*Free/Open Source Software*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
MIT	*Massachusetts Institute license*
REST	*Representational State Transfer*
SO	Sistema Operacional
SPA	*Single Page Application*
SQL	*Structured Query Language*
TAD	Tipo Abstrato de Dados
TCC	Trabalho de Conclusão de Curso
TS	*TypeScript*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

Atualmente grande parte das aplicações utilizam a arquitetura monolítica como padrão para desenvolvimento de *software*. Porém, em determinados casos esta arquitetura se mostra inviável e propensa a falhas. Em resposta a isso surgiu uma nova arquitetura de *software* chamada de micro serviços. Esta arquitetura visa a modularização do projeto em serviços menores, responsáveis por determinada área da aplicação e que podem funcionar de forma independente.

Este modelo de arquitetura, apesar de muito vantajoso em determinados tipos de aplicação, ainda é pouco abordado. Porém, devido ao alto poder de encapsulamento, independência e resistência a falhas, ele vem sendo utilizado pelas principais empresas de *software* do mundo em suas aplicações. Alguns exemplos são Amazon, Netflix, Ebay e Uber, demonstrando o verdadeiro potencial desta nova abordagem.

Para validar a utilização desta nova arquitetura foi desenvolvido o **EducaSys**, que é um sistema educacional que proporciona uma série de funcionalidades com o intuito de auxiliar a interação entre a escolas públicas e suas partes, seja elas alunos ou professores. Tais funcionalidades são de vital importância para a persistência de dados, agilidade de processos e interação dentre os membros da instituição.

Uma vez que aplicações similares raramente são disponibilizados pelo governo e sua implantação, por parte de terceiros, se torna muitas vezes inviável. A ideia de um *software* livre com licença MIT (*Massachusetts Institute of Technology*) [1] que possibilita a troca e consulta de informações entre professores e alunos pode se torna um empreendimento vantajoso e extremamente necessário, além do fato de possibilitar futuras expansões e melhorias, já partindo das funcionalidades mostrada nesse projeto. Pode vir a ser implementando uma série de utilidades para o gerenciamento da instituição, facilitando tarefas que muitas vezes são realizadas manualmente.

Todo o *software* foi desenvolvido para a plataforma WEB que possibilita o acesso remoto de qualquer dispositivo com acesso a rede, possibilitando maior alcance e área de abrangência do mesmo, independente do SO ou *hardware* utilizado.

Com base nesta necessidade foi possível modularizar este *software* baseado no modelo de micro serviços com o intuito de demonstrar as principais vantagens e diferenciais desta arquitetura.

Ao decorrer deste documento serão apresentados mais detalhadamente informações que justificam o desenvolvimento de uma aplicação de gerência escolar gratuita de código fonte aberto com base em micro serviços.

[1] Disponível em: <https://tlo.mit.edu/> Acesso em Junho de 2018

1.1 Objetivos

O **objetivo principal** deste trabalho de conclusão de curso é validar a utilização de micro serviços para desenvolver um *software* livre de código-fonte aberto (FOSS - *free open source software*) para o gerenciamento de um ambiente escolar, ou seja, fornecer ferramentas que agreguem e automatize processos antes feitos manualmente. Isso torna tais informações transparentes a ambas as partes.

São **objetivos específicos** deste trabalho:

- Desenvolver uma aplicação baseada na arquitetura de micro serviços;
- Utilizar tecnologias, metodologias e ferramentas atuais para o desenvolvimento do *software*;
- Criar um *software* livre para gerenciamento escolar;
- Fornecer um ambiente que permita ao usuário consultar informações relevantes em relação às disciplinas ministradas em sua escola;
- Prover um ambiente que possibilite a postagem de material de apoio para o estudo de tais disciplinas;
- Possibilitar aos alunos realizar entrega de atividades através do *website* de forma digital.

1.2 Justificativa

O processo de desenvolvimento de *softwares* WEB podem se tornar exorbitantemente grandes, englobando diversas áreas de atuação e um conjunto variado de funcionalidades. Porém, ao se desenvolver uma aplicação utilizando a arquitetura monolítica padrão, a aplicação se torna propensa a falhas que podem resultar na quebra de toda aplicação.

Este tipo de falha se torna cada vez mais crítica uma vez que impossibilita o funcionamento de todas as áreas da aplicação. O que torna definitivamente inaceitável em determinados casos.

Em contrapartida a este tipo de falha o modelo de micro serviços foi desenvolvido. Com o intuito de prevenir a falha geral da aplicação, esta arquitetura foca seus esforços para prover a modularização do *software*. Uma vez modularizada a aplicação diminui a área de impacto de possíveis falhas, aumentando a segurança e funcionalidade do projeto nos mais adversos ambientes.

Para isso, foi desenvolvido o site EducaSys que tem como intuito validar esta nova arquitetura, além de se mostrar extremamente importante para que haja uma boa interação entre professores, secretariado e alunos. Uma vez que todas as necessidades de ambas as partes são

feitas por meios manuais, sem a possibilidade de acesso e requisição remota, o qual torna tais processos mais trabalhosos e demorados. Tarefas que demandam mão de obra e matéria prima podem ser automatizadas evitando o gasto dos mesmos. Além de poderem ser efetuadas a distância a partir de qualquer dispositivo com acesso a rede, Isso evita que tais tarefas sejam feitas de forma presencial.

Este *software* visa também disponibilizar à comunidade acadêmica uma aplicação que utiliza de diversos recursos atuais para possibilitar uma boa funcionalidade ao utilizar ferramentas poderosas que possibilitam a persistência do *website* em situações adversas.

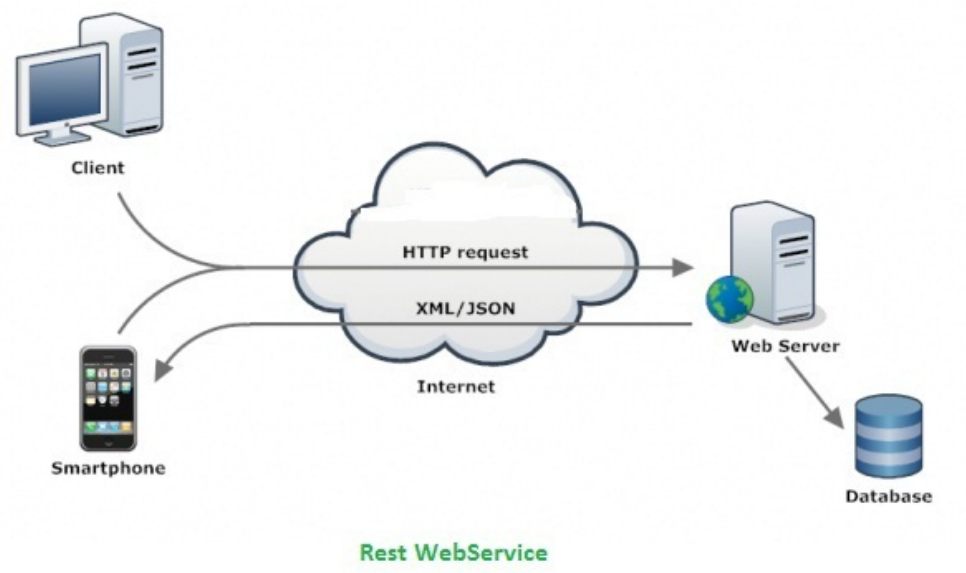
Ao desenvolver o *software* a partir da lógica dos micro serviços é possível modularizar o mesmo em pequenos serviços que são responsáveis por cada área do site., assim seu desenvolvimento, alteração e manutenção não afeta os demais.

2 REFERENCIAL TEÓRICO

O desenvolvimento de uma aplicação para o ambiente WEB fornece uma série de vantagens em relação às demais áreas de desenvolvimento. Este ambiente possibilita o acesso a plataforma independente do dispositivo e sistema operacional, desde que o mesmo possua acesso à rede. Desta forma uma aplicação desenvolvida para este ambiente abrange uma quantidade extremamente grande de dispositivos, que aumenta a área de alcance do *software* e torna o mesmo mais acessível independente de onde ou de qual dispositivo é utilizado.

Para entender melhor o trabalho deste tipo de aplicação é necessário entender como é a comunicação entre o servidor e as máquinas clientes. O esquemático da Figura 1 exemplifica a forma que a arquitetura cliente/servidor funciona.

Figura 1 – Comunicação usuário com servidor



Fonte: (WLCONSULTORIA, 2017)

A Figura 1 acima demonstra a função intermediadora exercida pela internet responsável por oferecer uma interface que direciona as requisições ao servidor e retorna as respostas as maquinas clientes. Desta forma, o processo de comunicação com os servidores se torna transparente ao usuário, que na visão dele é apenas conhecido a figura da internet.

Ao optar por esta plataforma não é necessário realizar a manutenção, assistência e atualização do *software* de forma presencial, uma vez que toda a aplicação está disponível e sendo executada em um servidor que pode ser acessado remotamente, no qual quaisquer alterações

no mesmo são refletidas em todos os dispositivos que usufruem de tais serviços. Isso torna o processo de implantação e manutenção mais fácil e prático.

Também deve ser levado em conta a possibilidade de expansões e melhorias da aplicação. Uma vez que a mesma é gerenciada de forma centralizada, tais *updates* se tornam mais fáceis de serem implementados e postos em prática.

Outro fator extremamente relevante que valida a escolha de um ambiente WEB é a não necessidade de desenvolver para hardwares diferentes. Uma vez que a aplicação não será armazenada na máquina local e sim em um servidor especializado, o qual possui um poder de processamento conhecido que proporciona um *hardware* voltado para a armazenagem e execução de tal aplicação.

Apesar de por muito tempo as aplicações WEB terem se baseado em uma arquitetura monolítica para o seu desenvolvimento, novas tecnologias vêm surgindo e possibilitando diferentes formas de se arquitetar uma aplicação. Um grande exemplo disso é a arquitetura de micro serviços, que vem ganhando cada vez mais espaço no mercado. E é esta arquitetura que será desenvolvida neste trabalho.

2.1 Micro Serviços

2.1.1 O que são Micro Serviços

Por muito tempo o desenvolvimento de *softwares* se deu por meio de aplicações monolíticas. Esta arquitetura tem como principal característica o desenvolvimento de uma grande aplicação responsável por todas as funções oferecidas pelo sistema.

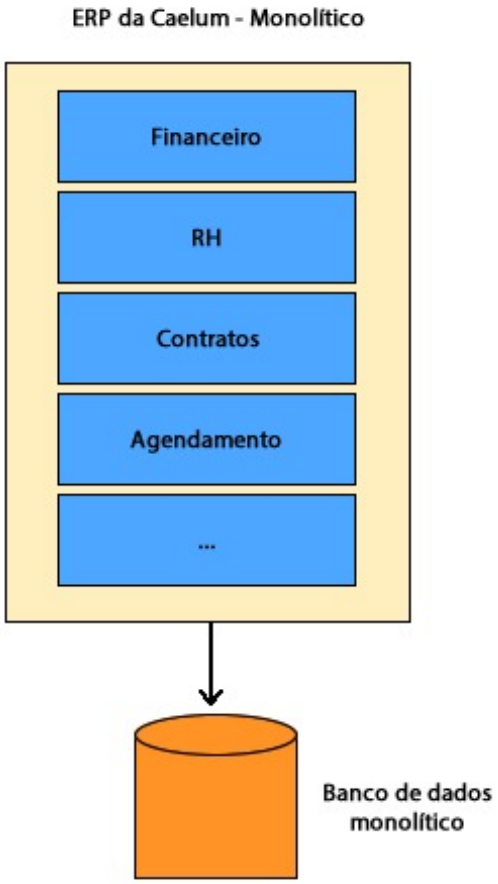
Porém, com o decorrer do tempo este modelo de desenvolvimento começou a apresentar falhas e se tornar inviável para alguns tipos de programas, pois ao desenvolver uma grande aplicação, responsável por todas as tarefas oferecidas pelo *software*, pode deixar o mesmo passível a determinados tipos de falhas. Como afirmado por Namiot (2014),

> "A aplicação monolítica pode ser difícil de se manter e modificar. Isso é especialmente verdadeiro quando o aplicativo está ficando maior. Com o crescimento da aplicação, é difícil adicionar novos desenvolvedores ou substituir membros da equipe." (DMITRY; MANFRED, 2014, p. 24).

Automaticamente ao adotar esta arquitetura o projeto se torna um ponto único de falha. Caso alguma de suas áreas de atuação apresente problemas, toda a aplicação será afetada impossibilitando a utilização de funcionalidades que independem da que apresenta um mau funcionamento.

A Figura 2 exemplifica o esquemático de um *software* desenvolvido com uma arquitetura monolítica:

Figura 2 – Modelo de aplicação monolítica



Fonte: (CAELUM, 2015)

Como pode-se observar, a aplicação é ilustrada como um grande bloco contendo outros pequenos blocos responsáveis por setores do projeto. Assim, caso algum setor deste projeto apresente inconsistências ou falhas, todos os demais serão afetados, pois pertencem a mesma estrutura.

Outro fator que pode tornar esta arquitetura ineficaz em sua execução é possuir uma base de desenvolvimento muito extensa, uma vez que todas as funcionalidades são desenvolvidas juntas, gerando uma grande quantidade de código.

Este tipo de desenvolvimento pode dificultar a abstração e separação dos problemas. Alm de atrelar problemas que não necessariamente se relacionam, podendo tornar inviável a modularização da aplicação além de dificultar a manipulação do código.

É possível notar a necessidade de fazer com que certas áreas da aplicação funcione de forma independente, uma vez que a comunicação entre as mesmas é extremamente fraca ou até

mesmo inexistente. Para atender a esta necessidade, foi desenvolvido um modelo de arquitetura baseado em micro serviços.

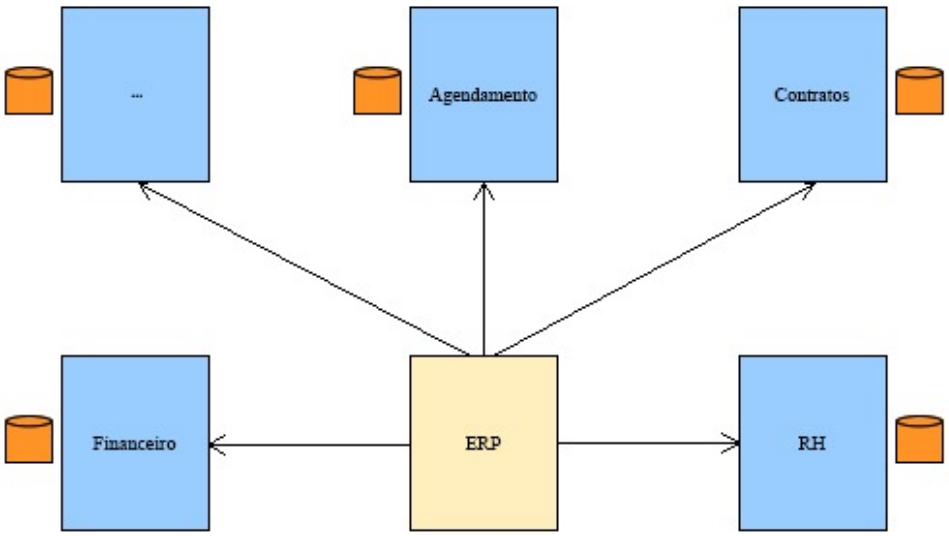
Podemos nos basear na definição de micro serviços proposta por Thönes (2015):

> "Um micro serviço, na minha opinião, é um pequeno aplicativo que pode ser implantado independentemente, dimensionado e testado de forma independente. É uma responsabilidade única no sentido original de que ela tenha uma única razão para a mudança e/ou uma única razão para ser substituída." (THÖNES, 2015).

Desta forma, uma arquitetura baseada em micro serviços propõe a divisão de um grande problema em pequenos problemas, que possuem pouca ou nenhuma interação entre si, com ênfase no fato de que caso um desses pequenos processos venha apresentar mau funcionamento os demais não são afetados.

Com base nesta lógica é necessário abstrair os requisitos de *software* e separá-los em áreas de atuação. Tais áreas devem ser analisadas à parte, a fim de detalhar seus objetivos e requisitos específicos. Esses módulos descentralizam as funcionalidades da aplicação e o desenvolvedor passa a pensar nelas separadamente.

Figura 3 – Modelo de aplicação com micro serviços



Fonte:(CAELUM, 2015)

A Figura 3 exemplifica uma abordagem, referente ao projeto da Figura 2, porém utilizando micro serviços. É possível observar que todas as subáreas identificadas foram divididas

em seus respectivos serviços, possuindo seus respectivos bancos de dados. Nota-se que ambas pertencem ao mesmo projeto porém são totalmente independentes em relação às demais áreas.

Desta forma, o foco do desenvolvedor é voltado a cada serviço possibilitando minimizar a complexidade do problema e focando seus esforços em uma área específica. Essa arquitetura possibilita abordar os modulos de uma forma mais detalhada, sem a necessidade de haver conhecimento dos demais serviços.

Portanto, para utilizar a arquitetura de micro serviços é necessário mudar a visão com relação a projeto e idealizar uma forma de dividi-lo em pequenos serviços, ao invés de planejá-lo como uma grande aplicação abordando diversas áreas e funções. Desta forma é possível focalizar todo o esforço de desenvolvimento em uma área específica, além de tornar esta área independente das demais.

2.1.2 Comunicação

Ao detalhar o conceito de micro serviço é necessário entender sua interação com o *frontend* e os demais micro serviços, além do seu papel em uma aplicação WEB. Para isso é necessário entender alguns conceitos.

Os micro serviços trabalham como *backend* e são responsáveis por fornecer informações para a interface final com o usuário. É neles que todas as tarefas do projeto são desenvolvidas e executadas além de retornar os dados a serem consumidos pelo *frontend*.

Já o *frontend* é a parte do projeto responsável por consumir as informações do *backend* e exibi-las ao usuário, possibilitando assim sua interação com tais informações. Desta forma, a medida que o cliente interage com as informações o *frontend* informa aos micro serviços tais interações para que a manipulação destes dados possa ser feita.

Apesar de um dos principais objetivos da utilização dos micro serviços ser o fato de que possam funcionar independentemente dos demais, pode existir situações nas quais é necessária a troca de comunicação entre os módulos.

Seja na comunicação com outros micro serviços ou com o *frontend*, todo o processo de interação entre eles é feito através de requisições que possibilitão a interação de forma independente da linguagem de programação ou plataforma, pois este tipo de requisição oferece uma padronização dos dados.

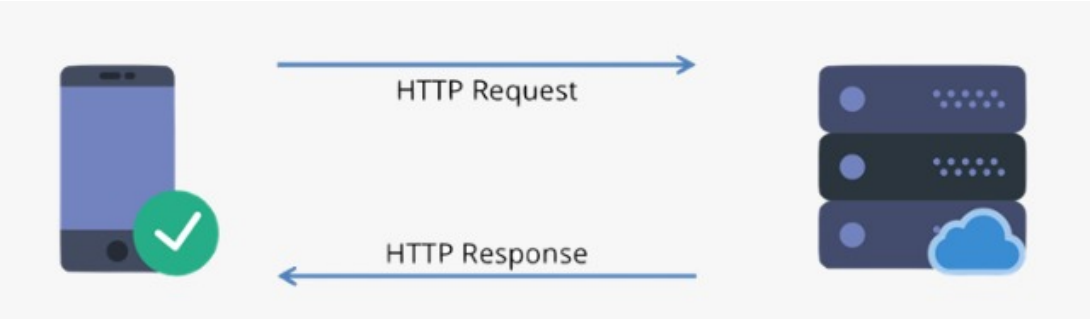
Estas requisições fazem parte do protocolo HTTP, que foi definido por Berners-Lee como:

> “Um protocolo em nível de aplicativo com a leveza e a velocidade necessárias para sistemas de informações hipermídia distribuídos e colaborativos. É um protocolo genérico, sem estado e orientado a objetos, que pode ser usado para muitas tarefas, como servidores de nomes e sistemas de gerenciamento

> de objetos distribuídos, através da extensão de seus métodos de solicitação (comandos)". (BERNERS-LEE; FIELDING; FRYSTYK, 1996)

As requisições funcionam da seguinte forma. O cliente abre um *socket* para se comunicar com o servidor e envia requisições (*requests*), estes são recebidas pelo servidor, tratadas para retornar as respostas(*responses*) que possui o conteúdo requisitado pelo cliente. A Figura 4 abaixo ilustra esta comunicação.

Figura 4 – Modelo de comunicação cliente servidor



Fonte:(UBIDOTS, 2017)

Esta requisição é composta por três partes. O *request line* que é responsável por especificar o tipo de requisição e a URL referente a requisição, os *headers* que informam as opções referentes a aquela requisição e o *body*, que é mais usado em requisições do tipo POST e é usado para informar os dados referentes a requisição.

Os dados contidos nestas respostas são padronizados utilizando o padrão de estrutura JSON (*JavaScript Object Notation*) definido por Crockford como:

> "JSON é um formato de texto para a serialização de dados estruturados. É derivado dos objetos literais do JavaScript, conforme definido no *ECMAScript Programming Language Standard, Third Edition* [ECMA]. O JSON pode representar quatro tipos primitivos (cadeias, números, booleanos e nulos) e dois tipos estruturados (objetos e matrizes)." (CROCKFORD, 2006, p. 1)

Desta forma, independente da linguagem e do ambiente que recebe tais dados, é possível interpretá-los de acordo com seu formato padronizado que possibilita a comunicação dos micro serviços entre si e com o *frontend*.

2.1.3 Arquitetura

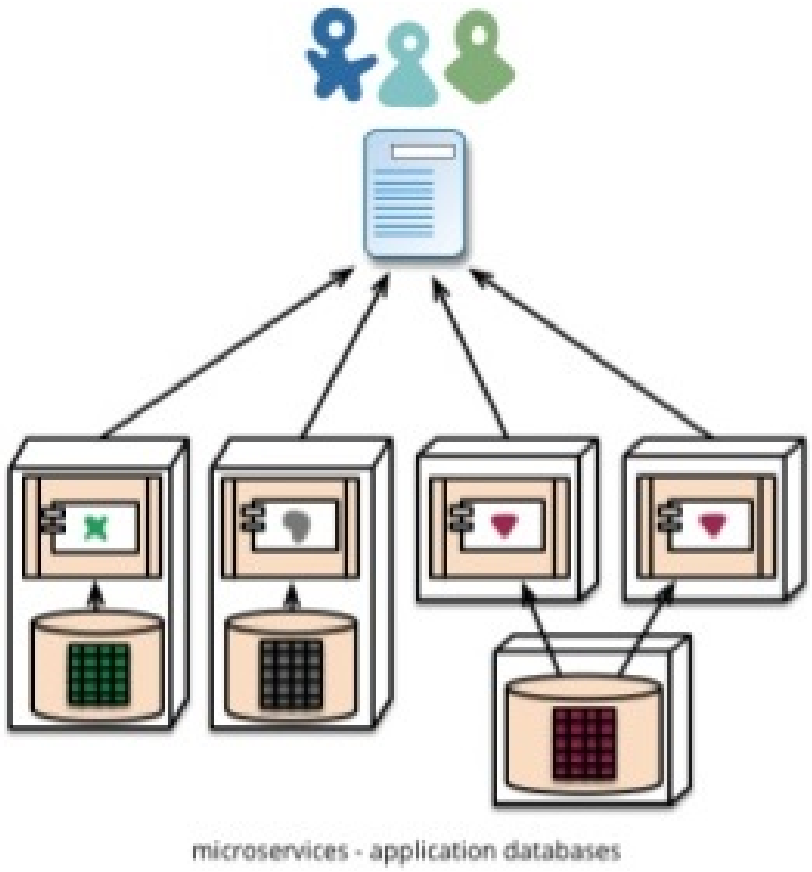
Como citado anteriormente a arquitetura de micro serviços possui uma abordagem diferente da monolítica. Ela propõe a divisão do projeto em módulos menores, responsáveis por

fornecer serviços que possuem semelhanças, idealizando-os de forma que os mesmos dependam o mínimo dos demais. Como reforçado por Júnior,

> "Micro Serviço é uma parte, específica e independente, de uma aplicação maior, como uma parte de outra aplicação, ela tem responsabilidades únicas dentro da arquitetura. Em outras palavras, a arquitetura de micro serviços é o conjunto de aplicações menores e independentes que juntas formam uma aplicação completa." (JÚNIOR, 2017)

A Figura 5 abaixo demonstra um esquemático desta arquitetura. Como é possível observar, vários módulos fornecem dados a uma mesma interface, que podem usar bancos de dados paralelos ou um banco de dado compartilhado de acordo com a necessidade.

Figura 5 – Estrutura básica de aplicação com micro serviços



Fonte:(APPDYNAMICS, 2016)

A Figura 5 destaca a autonomia dos serviços, demostrando que independentemente dos demais, um serviço fornece dados a uma interface de *frontend*. Esse serviço possibilita que a sua

área de atuação funcione mesmo que as demais áreas venham a falhar. O que acarreta também em troca de mensagens do *frontend* com diversos módulos simultaneamente.

Esta arquitetura provê uma série de vantagens. Dentre elas fornece uma melhora considerável na prevenção de falhas, de forma que a falha de um módulo do projeto não afeta os demais.

2.1.3.1 Vantagens

Uma vez que os micro serviços são planejados para funcionarem de forma **independente**, sua manutenção, modificação e atualização pode ser feita separadamente, enquanto os demais módulos se mantêm funcionando. O que torna a detecção de falhas mais fácil, devido ao fato do problema estar em apenas um módulo. Logo a área de abrangência do erro é mais específica.

Ao dividir o projeto em módulos, é possível levantar os requisitos e detalhar as etapas de desenvolvimento de forma específica para cada módulo, tornando assim o levantamento de requisitos e a análise do problemas mais aprofundados. Dando foco especificamente a cada módulo e os desenvolvendo de forma mais detalhada.

As etapas de desenvolvimento se tornam mais fáceis devido ao fato do problema ser menor e suas funcionalidades serem mais específicas, além de existir uma relação entre elas. Assim é possível ter um maior compreendimento do módulo e definir melhor suas funcionalidades.

Facilita também a adição de novos membros a equipe de desenvolvimento, devido ao fato da ideia do serviço ser mais específica e menos abrangente. O que torna a adaptação deste novo membro menos complexa e sua curva de aprendizado menos íngreme.

A modularização permite realizar melhorias e atualizações de forma mais prática. O que possibilita planejar, modelar e implementar atualizações referentes a um módulo específico. O processo de implementação pode ser realizado com os demais módulos em execução. Desta forma os impactos ao se planejar, desenvolver e implementar atualizações se tornam menores, não comprometendo o funcionamento da aplicação como um todo. Assim como D'Agostino afirma:

> "Eles são componentes fracamente acoplados que pode ser atualizado de forma independente e fornecer melhor escalabilidade com relação a outras soluções." (D'AGOSTINO et al., 2017, p. 4)

É possível realizar a entrega do produto através de etapas. Uma vez que um micro serviço referente a uma área do projeto esteja implementado, é possível realizar a implantação deste módulo e torná-lo funcional. Na medida que os demais módulos são desenvolvidos, sua implantação é realizada, além de possibilitar a implementação de novos módulos em um projeto já existente.

Outro importante ponto a ser exaltado é o ganho com relação a segurança. Caso haja uma falha de segurança em um determinado serviço os demais não são afetados. Essa divisão permite isolar a falha e tratá-la em um módulo específico.

2.1.3.2 Desvantagens

Porém em contrapartida existem fatores que devem ser considerados ao iniciar a utilização desta arquitetura, como por exemplo em relação a segurança. Apesar da possibilidade de isolar uma falha de segurança em apenas um dos módulos, a utilização de micro serviços requer uma maior atenção neste quesito. Diferente de um sistema monolítico, é necessário se preocupar individualmente com a segurança de cada módulo. Essa preocupação faz com que esta atividade se torne mais complexa e demanda mais mão de obra.

Apesar de o fato de dividir um projeto em vários módulos se mostrar uma ideia válida em diversos casos, esta abordagem possui ressalvas, pois é necessário que haja muita cautela ao dividir os serviços. Esses módulos devem possuir o mínimo ou nenhuma dependência dos demais.

Desta forma caso algum serviço necessite solicitar informações dos demais, é necessário haver uma prevenção caso o mesmo não responda a sua solicitação. Isso se deve principalmente ao fato de uma das principais propostas da utilização de micro serviços ser o funcionamento independente. Logo é necessário torná-lo persistente a este tipo de situação.

A modularização faz com que cada micro serviço seja tratado de forma separada. Isso pode acarretar problemas como citado por Koskinen,

> “Se cada micro serviço é muito original do aspecto de desenvolvimento e uma pessoa está desenvolvendo um certo microsserviço, então essa pessoa se torna insubstituível. Se ele deixar a equipe de desenvolvimento, encontrar outro desenvolvedor com a mesma competência pode ser difícil.” (KOSKINEN et al., 2016, p. 19)

É possível que ocorra uma centralização de mão de obra em um pequeno grupo, onde somente aqueles membros estão aptos a lidar com determinado serviço. Desta forma pode se tornar difícil substituir um grupo.

Porém, isto ocorre principalmente quando não existe uma preparação para a entrada de novos membros. Pois caso um novo membro seja treinado pelos membros já pertencentes a equipe, a abstração do projeto se torna mais rápida e fluida.

Logo é possível deduzir que esta tecnologia imprime uma série de vantagens. Porém, não deve ser aplicada deliberadamente, uma vez que o projeto necessita ter áreas de atuação específicas que possibilitam sua modularização.

É necessário ponderar que esta arquitetura demanda mais mão de obra específica. De forma que cada módulo deve ser tratado como um projeto que terá de passar por todas etapas de desenvolvimento de um projeto monolítico, fazendo com que haja um tratamento específico focado em cada módulo.

É de suma importância que o desenvolvedor saiba definir se um projeto pode ser ou não modularizado. Uma vez que a modularização gera mais complexidade no sentido de desenvolver vários módulos, a divisão deve ser de realizada de forma consciente, para que não haja a criação de módulos desnecessários.

2.1.4 Desenvolvimento

Para realizar o processo de implementação dos micro serviços, é necessário ter os módulos do projeto bem definidos. Ao definir quais serão os serviços a serem implementados e suas respectivas funcionalidades é necessário verificar se haverá necessidade de comunicação entre eles.

Com estes pré requisitos básicos definidos, levando em conta as necessidades difundidas pela utilização de tal arquitetura, é necessário escolher o ambiente ou *framework* que provê as ferramentas necessárias para o desenvolvimento.

Jaques propõe uma definição de *framework* da seguinte forma,

> "Framework é um conjunto de códigos abstratos e/ou genéricos, geralmente classes, desenvolvidos em alguma linguagem de programação, que relacionam-se entre si para disponibilizar funcionalidades específicas ao desenvolvedor de software. Em outras palavras, é como uma caixa de ferramentas, um kit que possui diversas funcionalidades devidamente implementadas, testadas e prontas para serem utilizadas no construção de softwares, poupando ao desenvolvedor tempo e trabalho na elaboração de operações básicas como acesso a banco de dados, sistema de templates, mapeamento de rotas e validação de dados." (JAQUES, 2016)

É necessário a utilização de um *framework* que provê ferramentas que auxiliam no desenvolvimento de micro serviços e que atue principalmente em sua comunicação tanto com os demais serviços, quanto com o *frontend* e o banco de dados. Ao analisar estas necessidades é possível encontrar diversas ferramentas que provê tais funcionalidades.

Uma das opções é o **Spring Boot Framework**. Uma plataforma Java que provê uma infraestrutura para o desenvolvimento de aplicações possibilitando assim uma abrangente gama de recursos que visam facilitar o desenvolvimento de *software* como micro serviços.

O *framework* propõe maior agilidade na publicação e gerência de uma aplicação. Ele utiliza uma forma simples de configuração e adição de bibliotecas esse *framework* se responsabi-

liza por realizar todo o processo de instalação e acoplamento das bibliotecas ao projeto. No qual é necessário apenas que o usuário informar suas dependências em um arquivo de configuração.

Tais afirmações são validadas pelo texto de Antonov (ANTONOV, 2015, p. 1) onde o mesmo exalta a sua flexibilidade:

> "Spring Boot é exatamente o tipo de framework que lhe dará potência combinada com a flexibilidade que lhe permitirá produzir software de alta qualidade em um ritmo rápido." (ANTONOV, 2015, p. 1)

O Spring Boot fornece uma abrangente gama de ferramentas que dão suporte ao desenvolvimento de micro serviços, desde ferramentas para a configuração de execução até anotações que são utilizadas para tratar os mais diversos tipos de situações que podem vir a ocorrer no projeto.

Estas anotações possibilitam controlar o comportamento do *framework*, que fornece as informações para os métodos e objetos, para que os mesmos possam lidar com suas funções da forma esperada. Assim é possível estabelecer uma configuração a determinada função ou objeto, afim de tornar as interações como ferramentas externas mais triviais, além de fornecer uma poderosa ferramenta de manipulação dos dados trabalhados pelo sistema.

É importante salientar a atuação por parte do *framework* com relação aos *Web Services* REST (*Representational State Transfer*). Que é definido da seguinte forma por Paliari,

> "REST é baseado no design do protocolo HTTP, que já possui diversos mecanismos embutidos para representar recursos como código de status, representação de tipos de conteúdo, cabeçalhos, etc. Ele aproveita os métodos HTTP para se comunicar. Porém, não existe um padrão obrigatório, pode ser implementado somente o que é necessário em seu contexto." (BALLEM, 2012)

O Spring Boot oferece uma série de funcionalidades que tem como objetivo auxiliar na implementação desta interface de forma rápida e ágil. Para isso é utilizado uma série de anotações que disponibilizam um comportamento necessário para lidar com as solicitações definidas por padrão.

Estas anotações provem às aplicações que utilizam o *framework* lidar com requisições HTTP de uma forma simples e funcional. Elas tratam estas com agilidade e praticidade, oferecendo ferramentas que auxiliam na manipulação de seu conteúdo a fim de disponibilizar os dados de uma forma mais concisa.

O *framework* também faz uso da JPA (*Java Persistence* API), que é definido por Bellem da seguinte forma:

> "O Spring Data JPA é responsável pela criação das classes Data Access Object (DAO) que possuem os métodos concretos para comunicação com a base de

> dados. Assim, fica abstraída a necessidade de se criar classes concretas para os repositórios de dados, sendo necessário apenas criar uma interface específica para cada classe de entidade, e nelas, estender a interface JpaRepository." (PALIARI, 2012)

A utilização desta ferramenta possibilita uma manipulação em mais alto nível das entidades presentes no banco de dados. Essa manipulação torna a interação entre o serviço e o banco mais trivial. Devido a fato da mesma já implementar todos os parâmetros padrões necessários para a comunicação e interação com a base de dados alem de possibilitar definir novos parâmetros de acesso que não seja os padrões.

Como reforçado acima o Spring Boot possui um conjunto completo de ferramentas que tornam o processo de desenvolvimento dos micro serviços mais ágil, que automatiza uma série de tarefas que possibilita ao desenvolvedor ter maior foco no desenvolvimento do serviço em si. Sem a necessidade de se passar grande parte do desenvolvimento tratando situações não referentes ao problema proposto.

2.2 Ferramentas auxiliares

É evidente que o desenvolvimento de um ambiente WEB não é definido apenas pelo desenvolvimento de seu *backend*. Por isso é necessário atrelar a essa poderosa tecnologia um conjunto de ferramentas que tornam a aplicação mais robusta, competente e usual.

Para isso foi utilizado um conjunto de tecnologias atuais e poderosas que fornecem uma vasta gama de funcionalidades que agregam ao projeto como um todo. O foco do projeto é utilizar das mais robustas e eficientes tecnologias do mercado, para o desenvolvimento de um protótipo resistente que possibilita a utilização de tecnologias que agregam no desenvolvimento e uso.

2.2.1 Bootstrap Framework

O Bootstrap é um *framework*, de código fonte aberto, desenvolvido inicialmente por Mark Otto e Jacob Thornton até então funcionários da Twitter. A necessidade de tal ferramenta se válida segundo Albino,

> "Havia uma crescente necessidade de padronizar os conjuntos de ferramentas de front-end (desenvolvimento de interfaces) de engenheiros em toda a empresa."(ALBINO et al., 2015, p. 152)

A ferramenta disponibiliza componentes que possibilitam desenhar telas WEB de forma responsiva. O que possibilita uma visualização agradável em diferentes tipos de dispositivos, com diferentes resoluções, tamanhos e formatos de tela.

Antes do desenvolvimento de plataformas como essa, era utilizado quaisquer bibliotecas, o que gerava inconsistências entre as mesmas acarretando em falhas, inviabilizando a utilização de um conjunto aleatório de bibliotecas.

Desta forma, o *framework* disponibiliza um conjunto completo de componentes necessários para o desenvolvimento de uma tela. Essas ferramentas facilitam a sua implementação e evitam a necessidade de busca de bibliotecas externas, uma vez que todos os componentes necessários já estão disponíveis.

Dentre as diversas opções existentes, optou-se pela utilização do Bootstrap neste projeto, devido ao fato de o mesmo ser o mais popular *framework* HTML, CSS, JS para desenvolvimento *frontend*. Desta forma o mesmo possui uma ampla documentação e compatibilidade com os mais diversos navegadores e dispositivos.

Além de sua compatibilidade um fator que reforça sua escolha é a agilidade. Pois segundo Bortoli (BORTOLI; RUFINO, 2016),

> "O Bootstrap, foi criado para agilizar o desenvolvimento front-end. Proporcionando agilidade e qualidade no desenvolvimento de interfaces. Essa framework, oferece componentes de interfaces pronto, onde o desenvolvedor não precisará ficar perdendo tempo na criação de componentes de interfaces, fazendo que o tempo de desenvolvimento da aplicação seja otimizada." (BORTOLI; RUFINO, 2016)

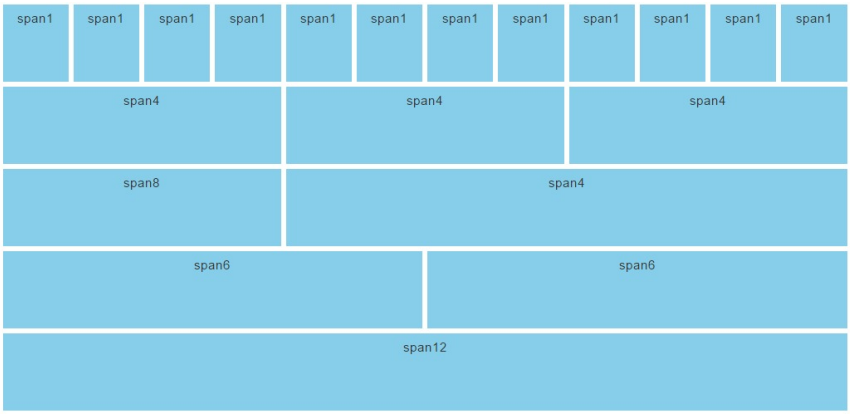
O Bootstrap nada mais é que CSS criado de forma pré-processada gerando uma série de vantagens em relação aos não processados. Isso se deve ao fato de que possui mais flexibilidade, possibilidade de gerar declarações aninhadas, variáveis para valores de propriedades CSS e uma série de outras funcionalidades.

O *framework* trabalha com um sistema de grid que divide a tela em 12 colunas (Figura 6). Tais colunas se rearranjam de acordo com o tamanho da tela, possibilitando assim alterar o modo de exibição da interface em determinados dispositivos. O que torna a mesma auto ajustável a tela na qual é exibida.

A Figura 7 mostra a estrutura da biblioteca Bootstrap. Na imagem é possível observar todos os arquivos necessários para seu funcionamento.

Sua utilização consiste em adicionar a pasta na raiz do projeto e referenciá-la no mesmo. No projeto em questão é utilizado a biblioteca Bootstrap específica para aplicações Angular. Para sua utilização é necessário instalá-la via Node.Js através do comando **npm install –save @ng-bootstrap/ng-bootstrap**.

Figura 6 – Grid Bootstrap



Fonte: (KILOBYTE, 2017)

Figura 7 – Estrutra Bootstrap

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    └── glyphicons-halflings-regular.woff
```

Fonte:(JAVABEAT, 2014)

2.2.2 Angular 5

Para atribuir funções aos componentes fornecidos pelo Bootstrap pode-se utilizar o **Angular Framework 5**, que possibilita a sua manipulação e gerenciamento atribuindo funcionalidades e realizando a comunicação, quando necessário, com processos externos.

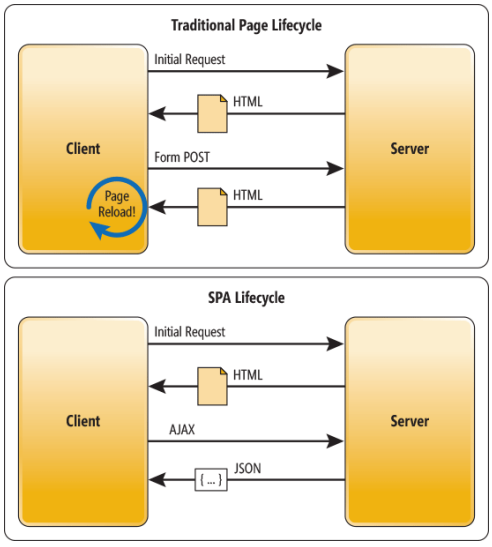
O Angular é um *framework* WEB de código fonte aberto, desenvolvido e mantido pela Google que disponibiliza ferramentas para auxiliar no desenvolvimento de *websites*. Ele possibilita a injeção de dependências, ferramentas *end to end* e boas práticas integradas para solucionar desafios. Além de utilizar o conceito de *Single Page Application* que pela definição de Mikowski é:

> "é uma aplicação entregue ao navegador que não necessita ter suas páginas recarregadas durante o uso." (MIKOWSKI; POWELL, 2013)

Isso é possível pois o esqueleto de todas páginas é carregado no início da utilização. Desta forma o cliente passa a requisitar apenas os dados, uma vez que toda a estrutura do site já está disponível em sua máquina. Isto torna a transição de páginas mais fluida e proporcionando uma sensação mais agradável ao usuário.

Uma vez que toda a estrutura do site já está disponível em sua máquina, os processos de requisição de dados obtém respostas mais rápidas pois como não é necessário retornar o código da página, a requisição retorna irá menos conteúdo, contendo apenas as informações solicitadas. A Figura 8 ilustra o funcionamento de uma página WEB tradicional e o de uma SPA.

Figura 8 – Single Page VS Modelo Tradicional



Fonte: (MICROSOFT, 2013)

É possível observar que para cada requisição realizada pela página WEB tradicional é retornado um novo HTML com os dados requisitados, realizando assim o reload para a exibição do mesmo. Já no modelo *single page*, todo o código HTML é provido pela primeira requisição. Desta forma as próximas requisições são referente apenas aos dados que serão exibidos nestes modelos de tela.

2.2.2.1 Componentes

Além disso, o Angular trabalha com a ideia de componentização de páginas, que visa dividi-las em módulos, os quais podem ser acoplados em outros componentes. Essa acoplação provê funcionalidades em várias áreas da aplicação sem a necessidade de duplicação de código. Estes blocos de código contém, no momento de sua criação, 3 arquivos base com as seguintes extensões CSS, HTML e TS. Estes arquivos são responsáveis respectivamentes pelas seguintes funções:

- CSS(Cascading Style Sheets): Responsável por descrever como os componentes HTML devem ser exibidos na tela, que controlam seu layout em diferentes situações;
- HTML(Hyper Text Markup Language): São arquivos contendo o código fonte da linguagem de marcação padrão para o desenvolvimento de páginas WEB;
- TS(Typescript File): Responsável pela manipulação dos componentes HTML, que trata suas ações e manipulando suas propriedades.

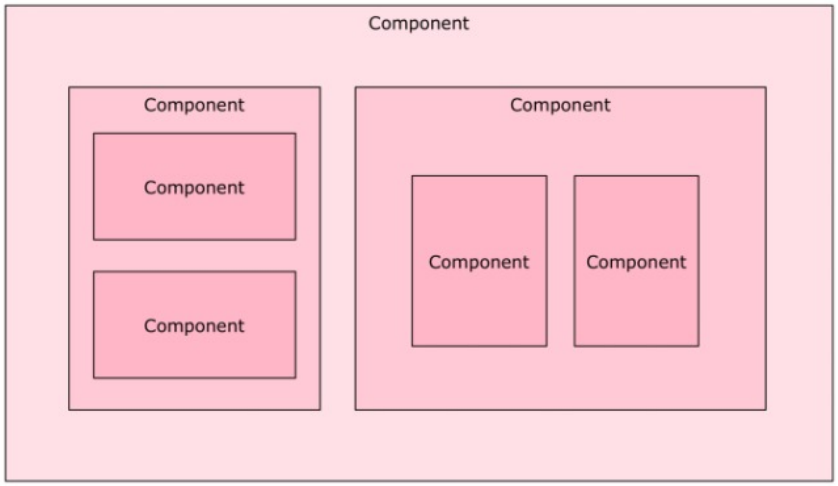
Tais componentes permitem dividir a aplicação em módulos menores. Cada módulo teria sua própria interface HTML, seus próprios estilos de CSS e sua própria lógica de funcionamento através do TS.

Desta forma é possível dividir problemas maiores em micro problemas, que possibilitam a implementação de tais componentes separadamente, unindo-os para formar uma página onde após um componente ser implementado, para utilizá-lo é necessário apenas inserir a TAG referente a aquele componente no HTML desejado.

A Figura 9 demonstra o funcionamento dos componentes. Ela representa uma página composta por diversos módulos. Assim esta tela consome os serviços de todos os componentes acoplados a ela, alem de possibilitar a junção de um conjunto de tarefas desenvolvidas paralelamente.

Esta modularização provê ferramentas para comunicar um conjunto de componentes e definir suas informações com base em dados pertencentes a outros módulos. Esta atribuição é feita através de anotações específicas para a interação dos objetos. Desta forma, é possível que haja comunicação entre os componentes, possibilitando a adaptação do mesmo ao ambiente que está acoplado.

Figura 9 – Angular Containers



Fonte: (TEROPA, 2015)

2.2.2.2 Diretivas

Para atribuir às funcionalidades ao componente, o *framework* dispõe de uma série diretivas que nada mais são que comandos que anexam comportamento aos elementos do HTML. Como citado por Simkhada

> "Basicamente, existem três tipos de diretivas em Angular: atributos, diretivas estruturais e de componentes. As diretivas de componente manipulam a parte de modelo para criar a interface do usuário, enquanto diretivas estruturais alteram o layout dos modelos de objetos de documento adicionando e desanexando o objeto DOM. Da mesma forma, os atributos desempenham um papel na alteração das aparências visuais de componentes e elementos." (SIMKHADA, 2017)

As diretivas podem atrelar condições para a exibição dos componentes. O que faz com que em determinadas ocasiões, o código HTML se comporte de acordo com as informações disponíveis na página. Este tipo de exigência pode variar desde condições para a visualização do item até condições para replicar tal componente um determinado número de vezes de acordo com uma lista de dados.

Um exemplo deste tipo de ação é quando existe um vetor de dados e deve ser criado um componente para cada objeto contido neste vetor. As diretivas possibilitam percorrer a lista de objetos e automaticamente gerar essa lista sem a necessidade de multiplicação de código.

Possibilitam também fornecer informações dos componentes HTML ao Typescript que fazem com que o mesmo possa usar os dados de cada componente e até mesmo atrelá-los a um

objeto. Desta forma qualquer alteração realizada no componente refletirá no objeto atrelado a ele.

Podemos exemplificar isso realizando a ligação de um componente a um objeto. Sempre que houver uma alteração no componente o valor do objeto atrelado a ele também irá ser alterado de forma que a recíproca é verdadeira.

Existe também um padrão de arquivos para a definição de objetos(TAD). Estas TADs (Tipos Abstratos de Dados) são definidas em uma arquivo com a denominação *module*. Após a criação de tais objetos é possível utilizá-los nos componentes. É usual utilizar um *module* para cada componente específico ou determina área de atuação. Eles podem receber os valores das requisições do backend e atribuí-los a componentes pertencentes ao HTML.

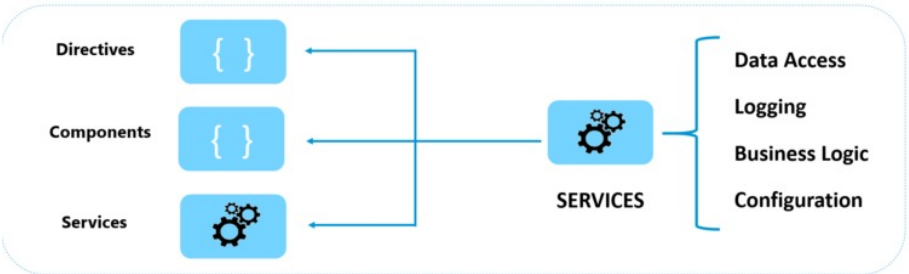
2.2.2.3 Comunicação

É usual desenvolver um arquivo responsável pela comunicação do *frontend* com os demais serviços. Este arquivo possui um modelo padrão que o define como um *service*. Ele é encarregado de todas as funções que realizam a comunicação com o *backend* e demais serviços externos e isso possibilita a existencia de um *service* para cada componente ou para uma determinada área de atuação do *frontend*.

Esta comunicação é feita através de uma biblioteca disponibilizada pelo Angular, denominada **HTTP Client**. Esta biblioteca disponibiliza as funções necessárias para realizar a comunicação de acordo com os padrões HTTP já citados que possuem funções para os métodos POST, GET, PUT entre outros. Assim é necessário informar apenas o caminho e os dados da requisição caso necessário.

Logo os serviços são classes encarregadas das requisições de dados. Neles são implementadas as funções responsáveis pelas solicitações HTTPs necessárias para o funcionamento dos componentes. Essas requisições são responsaveis por retornar ao componente uma resposta fornecida pelo servidor, na qual deve ser tratada da forma desejada pelo mesmo.

Figura 10 – Angular Service



Fonte: (EDUREKA, 2017)

A Figura 10 demonstra através de um esquemático a função de intermediador realizada pelo *service*. Ele atua como o ponto de conexão entre o *backend* e o *frontend*. Desta forma as todas as requisições de dados devem passar por um determinado *service*, tornando-o de vital importância para a aplicação.

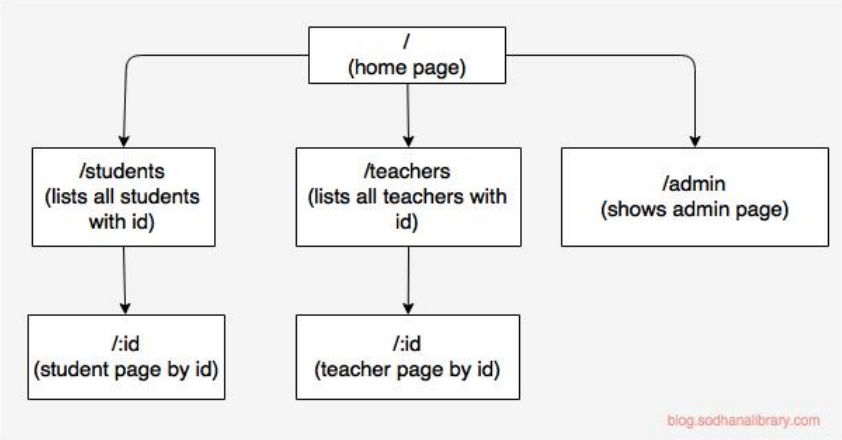
2.2.2.4 Rotas

Apesar do Angular fornecer uma experiência SPA (*Single Page Application*) como já mencionado. É necessário realizar a transição de páginas. Uma vez que cada página é composta de um ou mais componentes, sua alteração é feita através da troca do componente a ser exibido.

Essa substituição de páginas é feita através do **Angular Routes**. Sua principal função é interpretar uma URL como uma instrução para navegar entre os componentes. É possível também passar parâmetros adicionais ao novo componente, que podem ser de vital importância para o mesmo a buscar o seu conteúdo.

Podemos observar na Figura 11 um esquemático que representa o funcionamento das rotas. É possível observar o redirecionamento de rotas que apontam para outros componentes. Além de possibilitar a passagem dados por parâmetros para possibilitar a busca de informações referentes ao código especificado.

Figura 11 – Esquematico de rotas



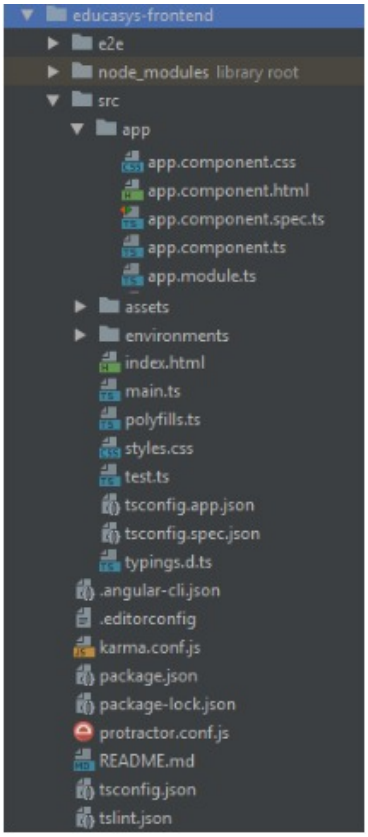
Fonte: (SODHANA, 2016)

Através das rotas é possível também definir ações para rotas inesperadas, como em ocasiões nas quais o usuário insere uma determinada rota inexistente. Nestes casos a biblioteca cuida para que esta rota incorreta seja tratada e retorne a uma página da aplicação.

Para que essas ferramentas possam funcionar é necessário um arquivo descrevendo todas as rotas e seus respectivos componentes. É possível também definir condições para que tais rotas sejam acessadas, que provê segurança e mais confiabilidade a aplicação.

A Figura 12 representa a estrutura de uma novo projeto. Para criar uma nova aplicação com Angular é necessário ter o Node.Js instalado na máquina. Atraves dele é possivel realizar a instalação do Angular CLI. Para isso é necessário executar o comando **npm i @angular/cli** no terminal. Após sua instalação o comando **ng new name** gera todos os arquivos iniciais da aplicação.

Figura 12 – Estrutura aplicação Angular



Fonte: O autor.

É importante salientar a funcionalidade de alguns destes arquivos, que são cruciais para o funcionamento da aplicação. São eles:

- **angular.cli.json**: Arquivo de configurações onde é informado dados vitais para o funcionamento da aplicação. Como por exemplo qual o arquivo HTML inicial, qual a pasta que contém os códigos dentre outras;
- **index.html**: É o HTML inicial da aplicação. Ele contém uma TAG que é responsável por exibir o conteúdo do componente denominado **app.component**. É possível também adicionar referências a bibliotecas neste arquivo;
- **app.module.ts**: É o arquivo de configuração no qual é declarado todas as bibliotecas a serem utilizadas na aplicação. Quando uma biblioteca é declarada neste arquivo toda

a aplicação tem acesso ao seu conteúdo. Além de ser declarado todos os componentes que fazem parte da aplicação, para permitir que haja um mapeamento dos componentes existentes. Também é responsável por conter os provedores de dados da aplicação, que são os arquivos responsáveis pela comunicação com o *backend*.

Com base nesta estrutura básica é possível inicia-se o desenvolvimento dos componentes e suas respectivas funcionalidades, usufruindo das diversas ferramentas fornecidas pelo Angular 5 que possibilitam o desenvolvimento de forma organizada através dos padrões estabelecidos pelo *framework*. Tornando o desenvolvimento mais eficaz e menos trabalhoso.

2.2.3 Firebase

O **Firebase** é uma plataforma online sustentada pela Google, que auxilia no desenvolvimento de aplicações *mobile* e WEB, disponibilizando um conjunto de ferramentas que possibilitam a substituição de um *backend* local. Kumar descreve seu funcionamento da seguinte forma:

> "O Firebase é uma tecnologia que nos permite criar aplicativos da Web sem programação do lado do servidor para que o desenvolvimento seja mais fácil e rápido. Usando o Firebase, não precisamos enfatizar o provisionamento excessivo de servidores ou a criação de APIs REST com apenas uma configuração muito pequena." (KUMAR et al., 2016)

O programa oferece uma base de dados que possibilita o armazenamento de informações por meio de Json que possibilita a consulta e armazenamento de dados de forma simples, eficaz. Além deste serviço a plataforma oferece outras funcionalidades de nuvem para o armazenamento de arquivos. Estes serviços podem ser incorporados em uma aplicação Angular após uma configuração básica.

Devido ao fato de ambos serem desenvolvidos pela mesma empresa, sua integração é feita de forma simplificada. Onde é necessário somente adicionar as bibliotecas referentes à aplicação e configurar a base de dados para a armazenagem dos arquivos.

A utilização do Firebase possibilita o armazenamento de arquivos e imagens utilizados pela aplicação. Uma vez que seu *upload* é efetuado, é possível ter acesso ao arquivo através de seu link. Desta forma é possível contornar as limitações de tamanho impostas pelo tipo **BLOB** do MySQL, que é usado para armazenamento de arquivos.

2.2.4 MySQL

Mantido atualmente pela Oracle o MySQL é um gerenciador de bancos de dados relacionais de código fonte aberto, que possibilita a criação, gerenciamento e atualização de bases de

dados. Tem como uma de suas principais características utilização da linguagem SQL (*Structure Query Lenguage*) que possibilita a manipulação de suas informações de forma simplificada. Este SGBD (Sistema de Gerenciamento de Banco de Dados) é responsável pelo armazenamento das informações utilizadas pelo *software* que possibilita o acesso remota ou localmente.

Em um banco de dados relacional o armazenamento de suas informações é feito através de tabelas. Cada coluna da mesma armazena um tipo de dado e suas linhas são responsáveis por os dados referentes a uma instância do mesmo. Além disso, é possível a criação de relacionamento entre suas tabelas que define um determinado tipo de interação entre elas.

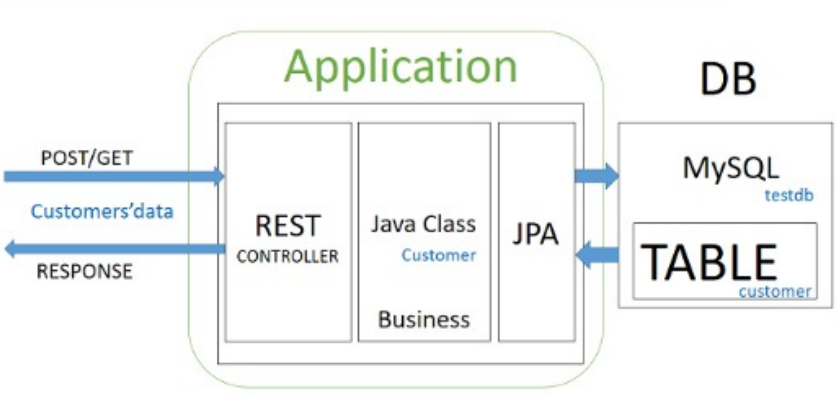
A ferramenta dispõe de uma interface gráfica denominada **MySQL Workbench**, que visa facilitar o desenvolvimento e manipulação de suas bases de dados, possibilitando realizar esta implementação de forma visual. Além de realizar operações de inserção, busca, remoção dentre outras.

Todas estas ferramentas são disponibilizadas de forma gratuita. Como ressalta Milani:

> "O MySQL é um banco de dados completo, robusto e extremamente rápido, com todas as características existentes nos principais bancos de dados pagos existentes no mercado. Uma de suas peculiaridades são suas licenças para uso gratuito, tanto para fins acadêmicos como para realização de negócios, possibilitando que na maioria dos casos as empresas o utilizem livremente" (MILANI, 2007, p. 21)

Além destas funcionalidades o *software* possui bibliotecas que podem ser acopladas as mais diversas aplicações e auxiliam na interação com o banco de dados, que torna sua manipulação mais didática e simplificada. Um exemplo é a própria JPA já citada neste documento. Podemos entender melhor seu funcionamento através da Figura 13

Figura 13 – Comunicação JPA - MySQL



Fonte: (JAVASAMPLEAPPROACH, 2018)

Esta imagem possibilita o compreendimento do funcionamento da JPA. Ela mostra que a mesma funciona como interface responsável pela comunicação entre o banco e a aplicação Java. Essa interface automatiza a interação e possibilita uma troca de dados com o MySQL de uma forma simplificada.

3 METODOLOGIA

Neste capítulo será destacado, passo a passo, as decisões e métodos utilizados para o desenvolvimento das funcionalidades propostas neste TCC seguindo a metodologia ágil do **Scrum**.

A ideia do *software* surgiu após uma série de entrevistas realizadas com professores e alunos das principais instituições de ensino público da região. Observou-se que nenhuma instituição contava com um sistema que permitisse a automatização de processos e transparência dos mesmos entre alunos e professores.

Foram consultadas um total de 6 escolas em Formiga e região. Dentre estas instituições um grupo de Escolas Estaduais utilizam o *software* disponibilizado pelo governo denominado **SIMADE**. As instituições que adotam esta aplicação são:

- Escola Estadual Padre Jose Sangali – Corrego Fundo;
- Escola Estadual Jalcira Santos Valadão – Formiga;
- Escola Estadual Dr Abílio Machado – Formiga;
- Escola Estadual Rodolfo Almeida – Formiga;
- Escola Estadual Prof Tonico Leite – Formiga.

O SIMADE é uma aplicação fornecida sem custos as escolas estaduais através do governo de Minas Gerais. Este *software* é responsável por armazenar o cadastro dos alunos e controlar dados como a frequência diária e a armazenagem de notas de atividades. Verificou-se que o *software* pode ser acessado remotamente, porém seu acesso é concedido apenas aos professores e o secretariado, desta forma não existe interação com o aluno. Ao questionar as instituições se existe possibilidade de disponibilizar o ambiente para a utilização dos alunos foi informado que a necessidade existe porém nenhum passo para implantá-la foi tomado, tornando a aplicação exclusiva aos funcionários da instituição.

Além disso, foi analisado as funcionalidades fornecidas pela aplicação, concluindo que o SIMADE não implementa um conjunto de funcionalidades como a disponibilização de material e a possibilidade de entregas *online*. Desta forma o *software* se mostra eficaz na armazenagem de informações que são úteis ao secretariado da instituição, deixando a desejar em relação ao conjunto de ferramentas para interação entre aluno e professor.

O SAGU também é uma opção de *software* livre para gerenciamento escolar. A ferramenta já está presente no mercado a muito tempo e possui um conjunto de funcionalidades

extenso. Devido a esse grande abrangência a aplicação se torna inviável para escolas publicas menores que não necessitam de uma aplicação tão complexa.

Assim o Educasys pode substituir ou **acrescentar** a experiência fornecida pelo *software* estadual, uma vez que pode ser planejado como projeto futuro a acoplamento do mesmo a base de dados do SIMADE, para que ambos trabalhem em conjunto para fornecer um conjunto completo de ferramentas para as escolas públicas estaduais.

Além disso, existem escolas que são geridas municipalmente, como é o caso da Escola Municipal Rafael José Alves de Córrego Fundo. Esta instituição utiliza outro *software* para o gerenciamento de seu diário escolar. Esta aplicação disponibiliza um *software* mobile que possibilita o acesso remoto de informações essenciais para o professor.

Novamente o *software* não disponibiliza uma interface de comunicação para os alunos. Desta forma não existe a transparência de dados proposta por este trabalho de conclusão de curso, mostrando que o Educasys pode ser uma opção e até mesmo um complemento as funcionalidades oferecidas pelos demais *softwares* existentes.

Com base nessas informações percebeu-se então a necessidade de uma aplicação que provesse a interação entre alunos e professores tornando os dados, referentes às disciplinas, mais transparentes e acessíveis.

Desta forma, é possível automatizar um conjunto de tarefas que antes eram feitas manualmente. Essa automatização possibilita a transparência destas informações em tempo real, tanto ao aluno quanto ao professor, oferece também a troca de informações entre as duas partes.

A modelagem e desenvolvimento do projeto foi realizada através do **Scrum**. Uma metodologia de desenvolvimento ágil que visa a elaboração do *software* através de interações, que nada mais são que ciclos nos quais um determinado conjunto de tarefas são desenvolvidos.

Os ciclos utilizados por esta metodologia são denominados *sprints*, que são definidas como intervalos semanais, quinzenais ou mensais nos quais um conjunto de funcionalidades deve ser implementado.

Além disso, esta metodologia propõe observações sobre o andamento do projeto ao final de cada dia com o intuito de atualizar o andamento da *sprint*, a fim de notar possíveis atrasos e definir quais os próximos passos a serem tomados para a conclusão do ciclo.

Para isso, é necessário realizar um levantamento de requisitos, listando todas as funcionalidades que a aplicação possuirá. No caso específico deste projeto, este levantamento foi realizado de forma extremamente cautelosa, pois os requisitos devem ser bem definidos e divididos em módulos, nos quais as funcionalidades contidas nestes devem se assemelhar de alguma forma, uma vez que estes módulos serão a base para o desenvolvimento dos **micro serviços**.

A análise de requisitos foi a primeira *sprint* realizada no projeto. Sua execução foi planejada para ser concluída em um mês. Ela foi feita por meio de entrevistas realizadas com alunos

e professores das instituições. Desta forma foi possível obter diversas funcionalidades para as partes. Os principais requisitos levantados para a construção do *software* foram:

- Possibilidade de postar e consultar notas;
- Possibilidade de postar e consultar faltas;
- Possibilidade de postar e baixar arquivos;
- Possibilidade de enviar e consultar atividades.

Após os requisitos serem levantados era necessário, no segundo ciclo de desenvolvimento, separar tais requisitos em módulos. De forma que as tarefas de cada módulo possuam relação de alguma forma. Contudo, chegou-se a conclusão que o mais viável para a aplicação seria a divisão do projeto em setores. Cada setor seria referente a um tipo de usuário (aluno, professor e secretariado).

O Scrum utiliza um conjunto de listas de atividades (a fazer, fazendo e feito) que visam tornar evidente o andamento da implementação de cada função, sendo que a partir da lista em que aquela funcionalidade pertence, é possível abstrair a que ponto de desenvolvimento a mesma está.

Com os setores já definidos, todas as funcionalidades são adicionadas a uma lista que possui a nomenclatura *Product Backlog*, que nada mais é que uma lista com todas as tarefas a serem cumpridas para o desenvolvimento do projeto. Assim sempre que uma nova *sprint* vai se inicializar, é realizado uma avaliação denominada *sprint planning meeting* é definido quais as funcionalidades serão desenvolvidas naquela sprint e é nesta etapa de avaliação que tais funcionalidades saem do *Product Backlog* e vão para uma nova lista chamada *Sprint Backlog*.

Esta lista representa todas as tarefas que serão desenvolvidas naquele ciclo. Logo, na medida que tais tarefas entram em fase de desenvolvimento, seu status muda para desenvolvimento (*doing*), assim é necessario desloca-la para uma nova lista que visa armazenar as funções que estão sendo produzidas naquele momento pela equipe.

Ao concluir uma funcionalidade, a mesma é enviada a etapa de testes e é realizado uma bateria de verificações que visam comprovar seu funcionamento nos mais diversos cenários de utilização. Após a conclusão desta etapa a funcionalidade pode ser adicionada a lista de tarefas encerradas.

Assim então se iniciou uma etapa de extrema importância para o desenvolvimento do *software*. A parte de documentação UML (*Unified Modeling Language*) é responsável por especificar, construir e documentar os artefatos de *software*. Ramos ressalta que:

> “A ênfase da UML é na definição de uma linguagem de modelagem padrão e,por conseguinte, independente de linguagens de programação, de ferramen-

> tas CASE, bem como dos processos de desenvolvimento. O objetivo da UML é, dependendo do tipo de projeto, da ferramenta de suporte ou da organização envolvida, poder adotar diferentes processos/metodologias, mantendo-se, contudo, a utilização de uma única linguagem de modelagem." (RAMOS, 2006)

Esta fase é de extrema importância para o desenvolvimento lógico da aplicação. Pois através da documentação gerada pela modelagem é possível compreender a estrutura e o funcionamento da aplicação dando início ao desenvolvimento com embasamento nessa documentação.

Dentre os diversos modelos de diagramas fornecidos pela UML, o **diagrama de caso de uso** possui extrema importância para o desenvolvimento do *software*. Ele é responsável por descrever a interação dos usuários do sistema com as funcionalidades oferecidas pelo mesmo.

Assim é possível abstrair como será a interação do usuário final com a aplicação, que provê uma maior compreensão das etapas de desenvolvimento e o modelo que as mesmas devem seguir, visando o desenvolvimento de um *software* funcional de acordo com os requisitos solicitados pelos usuários.

Este diagrama é composto pelos atores que são os usuários finais da aplicação, o caso de uso que representa uma funcionalidade que pode ser executada por um ator e a comunicação entre ator e caso de uso, assim é possivel determinar qual o tipo de interação possível entre os atores e os casos de uso.

O **diagrama de classes** é tão importante quanto o de casos de uso. Seu desenvolvimento tem como principal objetivo representar as entidades presentes na aplicação. Estas entidades possuem um conjunto de atributos que definem suas principais características.

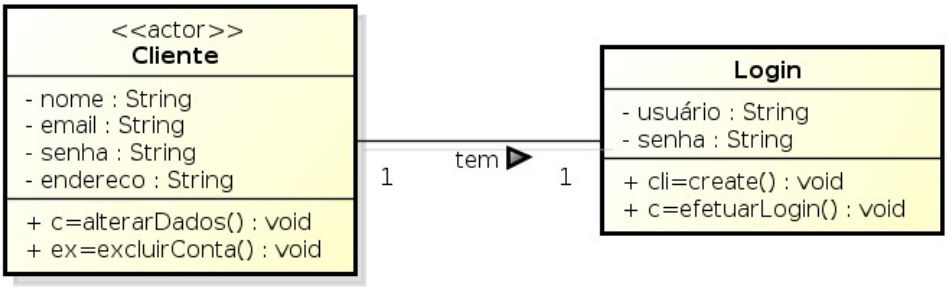
Além das entidades é necessário criar os relacionamentos entre as mesmas. Este relacionamento é representado por linhas que interligam as tabelas. Estas linhas representam a interação que existe entre estes esquemas. A ligação entre as tabelas pode indicar a quantidade de objetos que são associados naquela ligação.

A Figura 14 demonstra um exemplo deste diagrama. Nela cada tabela representa uma entidade (no caso Cliente e Login) e estas entidades possuem um conjunto de atributos e funções que manipulam estes atributos. Este diagrama serve de base para o desenvolvimento do diagrama de entidades e relacionamentos do banco de dados e cada entidade mapeada neste diagrama corresponde a uma tabela do banco de dados.

O desenvolvimento da documentação UML possibilita uma base de informações que deve ser seguida para o bom andamento do projeto. Desta forma é possível iniciar a codificação da aplicação.

Com base nesta documentação foi possível desenvolver um dicionário de dados que referencia as entidades presentes na aplicação. Este dicionário possibilita o melhor entendimento

Figura 14 – Diagrama de classes



Fonte: O autor

sobre as entidades e suas funções dentro da aplicação. Os dados presentes neste dicionário são:

- Aluno: Entidade que tem como intuito representar um estudante presente na instituição. Possui todas as informações pessoais necessárias referente ao indivíduo;
- Professor: Referência um docente. Seus campos são responsáveis por definir todas as informações básicas referente a determinado indivíduo que exerce tal função na instituição;
- Secretaria: Contém as informações referentes a um usuário do tipo secretaria;
- Disciplina: Tal entidade possui os dados básicos necessários sobre uma determinada disciplina. É necessário que exista uma ligação 1 para N com relação ao Professor. Uma vez que cada disciplina possui um professor que a ministra. E um professor pode ministrar diversas disciplinas;
- AluDis: Em alguns casos é necessário que haja uma entidade que faça a união entre duas outras. Este é o caso desta entidade. Seu intuito é atribuir alunos a disciplina. Uma vez que existem informações básicas referente a determinado aluno em uma disciplina, como por exemplo número de faltas, estas informações são armazenadas nesta tabela;
- Atividade: Usada na representação de atividades avaliativas referentes a determinada disciplina;
- Arquivos: Possui as informações básicas referente aos arquivo de auxílio aos estudos. Possui uma ligação com as disciplinas, pois cada disciplina possui sua lista de arquivos;
- AluAtivdade: A exemplo de AluDis os dados presentes nesta entidade são utilizados para interligar um aluno a uma atividade pertencente a determinada disciplina. Desta forma todas informações desta atividade referente ao aluno é armazenada neste objeto.

3.1 Secretaria

Como ressaltado anteriormente a aplicação foi dividida em três áreas, sendo elas aluno, professor e secretária. O primeiro módulo desenvolvido foi o de secretariado, pois é nele que é realizado o cadastro de alunos, professores e disciplinas que são de vital importância para o funcionamento dos demais módulos.

Desta forma através de um conjunto de *sprints* foi iniciado o desenvolvimento do primeiro serviço da aplicação, que teve como base desenvolver um ambiente eficaz e intuitivo para o cadastro, modificação, remoção e vínculo das principais entidades da aplicação.

O primeiro ambiente desenvolvido foi o de manipulação do aluno. Nesta *sprint* foi desenvolvido uma interface que realiza a listagem de todos os alunos pertencentes à instituição. Além de disponibiliza o acesso a operações básicas de inserção, alteração e remoção. A inserção e modificação do aluno são feitas em uma tela separada. Onde todos os dados referentes ao mesmo são disponibilizados para preenchimento.

Para o desenvolvimento destas funcionalidades foi necessário implementar um conjunto de funções que tratam os dados enviados pelo *frontend* e realizam sua manipulação no banco de dados. Estas funcionalidades utilizam de uma interface **REST** para obter os dados e através da JPA realiza sua manipulação no banco de dados.

Com base nesta mesma estrutura foi desenvolvido em uma nova *sprint* o ambiente de cadastro de professor. Tanto professor quanto aluno são ambientes com muitas similaridades. Basicamente possuem as mesmas funcionalidades, porém cada um faz referência a sua respectiva entidade. Mas assim como o aluno, ele possibilita o cadastro, alteração e remoção de professores.

Já a *sprint* referente ao desenvolvimento do cadastro de disciplina se mostrou relativamente diferente. Pois a entidade disciplina possui ligação com diversas outras entidades. Assim foi necessário o desenvolvimento de um conjunto de tabelas que interligam a tabela disciplina as demais.

Além de desenvolver toda uma lógica de conexão, foi necessário também realizar a manipulação de todas estas tabelas pois, uma vez que um dado era manipulado dentro de disciplina, um conjunto de outras tabelas era afetado. Era necessário realizar a manipulação destes dados para que todo aquele conjunto de tabelas fosse alterado.

Como não poderia ser diferente, uma vez que as funcionalidades eram implementadas no *backend* tornava-se possível o desenvolvimento do *frontend* que iria consumir aqueles dados. Desta forma esta produção foi independente e feita em etapas.

Analogamente aos dois extremos de uma linha telefônica. Era necessário, assim como no *backend*, implementar no *frontend* funcionalidades que interfaceiam com as aplicações externas. Estas funções são responsáveis por requisitar algo a um programa externo e aguardar sua

resposta em um formato padronizado. Assim é possível interpretar estes dados e desenvolver uma interface para exibi-los.

Com os dois extremos da aplicação desenvolvidos, tornou-se possível iniciar uma etapa de testes que foram realizados para validar se a comunicação ocorreria da forma esperada, seus resultados muitas vezes demandam modificações de código, afim de corrigir alguma falha.

Após implementar adequadamente tanto as funcionalidades do *backend* quanto as do *frontend*, testá-las e afirmar seu funcionamento. Foi possível definir como concluídas todas as funcionalidades referentes ao micro serviço de secretaria, que é responsável pelo cadastro e manipulação de um conjunto de entidades da aplicação.

A encerrar seu desenvolvimento se tornava possível realizar a implantação do serviço e disponibilizá-lo de forma funcional se necessário. Isso se deve ao fato de que cada micro serviço independe dos demais para seu funcionamento, logo as tarefas realizadas pela secretaria podem ser executadas independente se os demais módulos estão ou não implementados.

Desta forma quando os outros módulos forem implementados e implantados, eles também funcionarão de forma independente do serviço de secretaria. É importante ressaltar que ao afirmar tal independente baseia-se que já existam informações no banco de dados a serem consumidas. Uma vez que aluno e professor necessitam ter seus dados no banco para funcionarem. Este caso não define uma dependência entre os serviços e sim a necessidade de uma base de dados.

3.2 Professor

Ao término do desenvolvimento do módulo de secretária foi iniciado o módulo de professores. Porém, antes de iniciar seu desenvolvimento foi necessário realizar uma reavaliação do serviço, revendo de forma rápida todas suas funcionalidades. A fim de planejar quais funcionalidades seriam desenvolvidas na *sprint* inicial do serviço.

As funcionalidades que foram listadas para o serviço de professores foram:

- Possibilitar a consulta das disciplinas ministradas;
- Desenvolver uma interface que permita o acesso às informações de determinada disciplina;
- Agendar atividades avaliativas, podendo disponibilizar a opção de entrega online;
- Disponibilizar material de estudo para *download*;
- Ferramenta para verificar a presença diária dos alunos.

Dentre todas as atividades propostas para este módulo, a primeira a ser desenvolvida foi a consulta das disciplinas ministradas pelo professor. Pois estes dados são necessários para que o professor possa navegar entre elas e visualizar suas informações. Para que seja possível realizar a consulta destas disciplinas é necessário que o cadastro da mesma tenha sido efetuado pela secretaria. Após este cadastro ser efetuado automaticamente a disciplina se mostra presente na lista do professor.

Após a conclusão desta etapa era necessário desenvolver o próximo e mais importante passo deste serviço, o ambiente referente a uma disciplina específica. Responsável por todas as interações referentes a professor e disciplina, esta *sprint* foi cuidadosamente planejada e desenvolvida para que atendesse a todos os pré requisitos propostos.

O termino dessa *sprint* possibilitou realizar atividades como cadastrar e consultar entregas referentes a atividades, enviar arquivos de conteúdo através do **Firebase**, realizar a validação de presença dos alunos, efetuar a postagem das notas referentes a tais atividades. Dentre outras diversas informações referentes aquela disciplina, que são de vital importância ao professor.

Para a exibição de todas estas informações é necessário que o *frontend* faça uma série de requisições ao serviço e este por sua vez realiza a consulta e filtragem de dados retornando-os ao Angular, que por sua vez trata estas informações e as exibe ao usuário.

3.3 Aluno

Com todas as funcionalidades referentes a professor implementadas é possível partir para o desenvolvimento do próximo serviço. O módulo referente aos alunos possibilita que os mesmos realizem tarefas que se mostram de extrema importância para este tipo de usuário. Seu desenvolvimento foi feito com base em todas informações reunidas através dos diagramas UML.

Assim como no serviço de professores, é necessário exibir ao aluno as disciplinas nas quais ele está matriculado. Esta é a *sprint* inicial referente a este módulo. Que visa fornecer ao aluno uma lista as materias em que ele está matriculado, alem de possibilitar uma visão prévia de informações básicas referente a elas. Desta forma o usuário pode escolher uma destas disciplinas e consultar suas informações mais detalhadamente.

Ao adentrar no componente referente a uma disciplina específica, é necessário que o mesmo ofereça todas as informações necessárias a aluno. Informações vitais como dados referentes às avaliações, materiais, faltas e demais dados. Além de possibilitar a entrega de trabalhos práticos.

Para isso o *frontend* realiza uma série de requisições externas tanto ao **micro serviço** quanto a **firebase**, essas requisições fazem com que ele manipule e ofereça informações e serviços ao angular. É importante ressaltar o fato de que estes serviços funcionam totalmente inde-

pendente dos fornecidos pelos demais módulos.

Desta forma é possível validar uma das principais propostas da aplicação, fornecer transparência de informações a ambas as partes que possibilita a interação entre as mesmas e automatizando funcionalidades que antes eram feitas manualmente.

Após o desenvolvimento de todos os módulos separados. Foi necessário desenvolver uma estrutura de rotas no *frontend* que possibilita a transição e comunicação das telas, assim é possivel realizar a troca de informações entre as telas, a fim de fornecer dados que permitam às requisições de informações ao *backend*.

Além destes três serviços principais, foi necessário o desenvolvimento de um quarto serviço. Por mais trivial que seja, foi necessário desenvolver um serviço que que fosse responsável por direcionar o usuário a sua respectiva área. Este serviço é a base para redirecionar o *frontend* aos demais serviços, uma vez que sua principal função é realizar a validação do usuário e direcioná-lo a sua área de atuação.

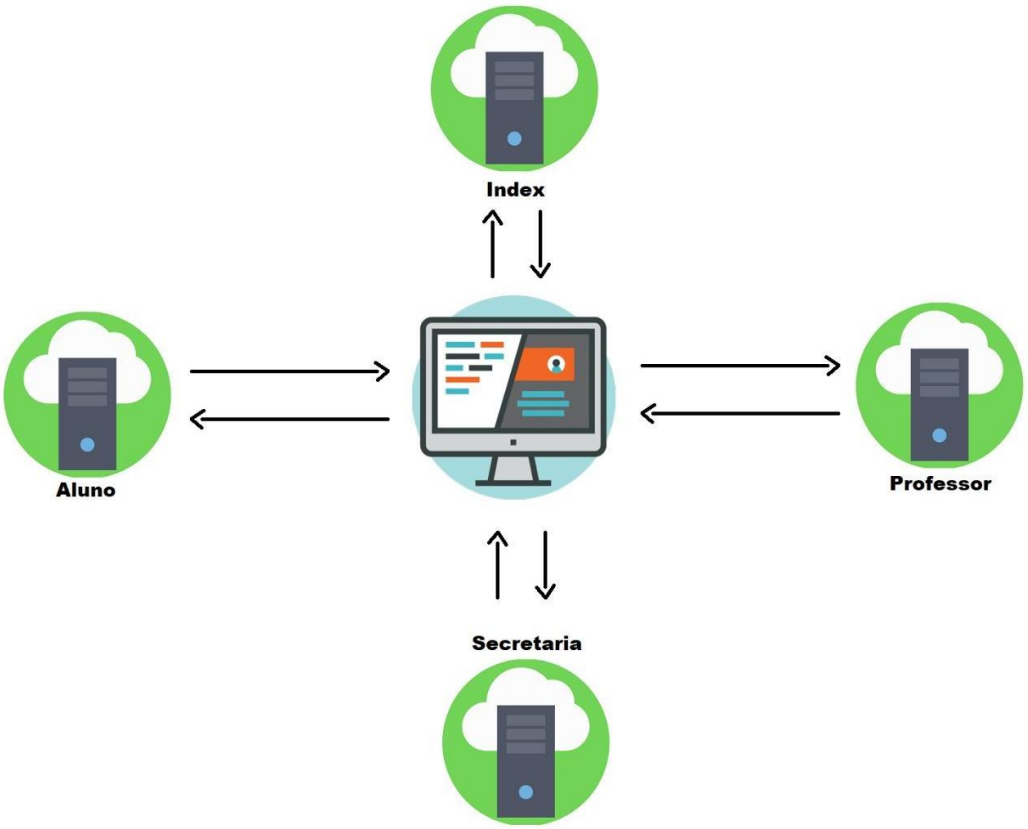
Após todas as *sprints* de análise, desenvolvimento e testes serem concluídas foi possível avaliar os resultados da aplicação de acordo com a proposta inicial da mesma. A fim de validar o funcionamento destas atividades.

4 RESULTADOS

Os resultados demonstrados a seguir são consequência de todo o processo de desenvolvimento descrito anteriormente. Através deles é possível validar os requisitos pré definidos no início deste documento.

É necessário iniciar este capítulo demonstrando na Figura 15 a estrutura/arquitetura de *software* desenvolvida para esta aplicação.

Figura 15 – Arquitetura Educasys



Fonte: O autor.

A Figura 15 demonstra o funcionamento do ecossistema do projeto. Os quatro serviços desenvolvidos fornecem informações para o *frontend* implementado em **Angular**. O serviço denominado Index é responsável por fornecer os dados necessários para o login do usuário e realizar a validação das informações e transmitindo qual tipo de usuário está utilizando a aplicação.

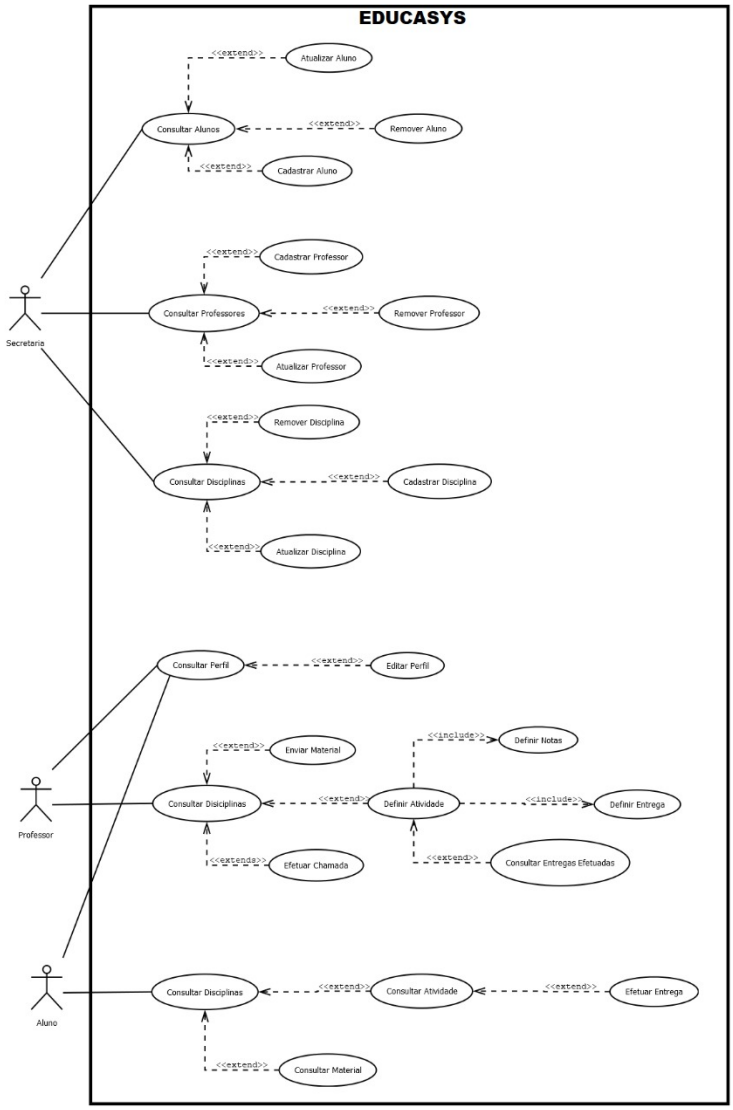
Desta forma cada um destes serviços são capazes de trabalhar de maneira independente além de acessar o banco de dados para manipular informações de acordo com sua funciona-

lidade, dessa forma é possivel obter o funcionamento paralelo e autônomo destas áreas. Caso um destes serviços venha a falhar, a área do *frontend* referente ao mesmo passa a não receber informações. Porém as demais áreas continuam funcionando.

O desenvolvimento do projeto demonstrado na Figura 15 foi realizado com base em uma **modelagem UML** desenvolvida e detalhada em um conjunto de diagramas. Estes diagramas foram de extrema importância para a abstração e implementação da aplicação.

O diagrama de casos de uso demonstrado na Figura 16 possibilitou a abstração das funcionalidades existentes em cada área de atuação do *software*. Assim é possivel assegurar o compreendimento da interação entre ator e atividades e se torna possivel abstrair, passo a passo, o modo que ele irá utilizar a aplicação (vide Anexo A).

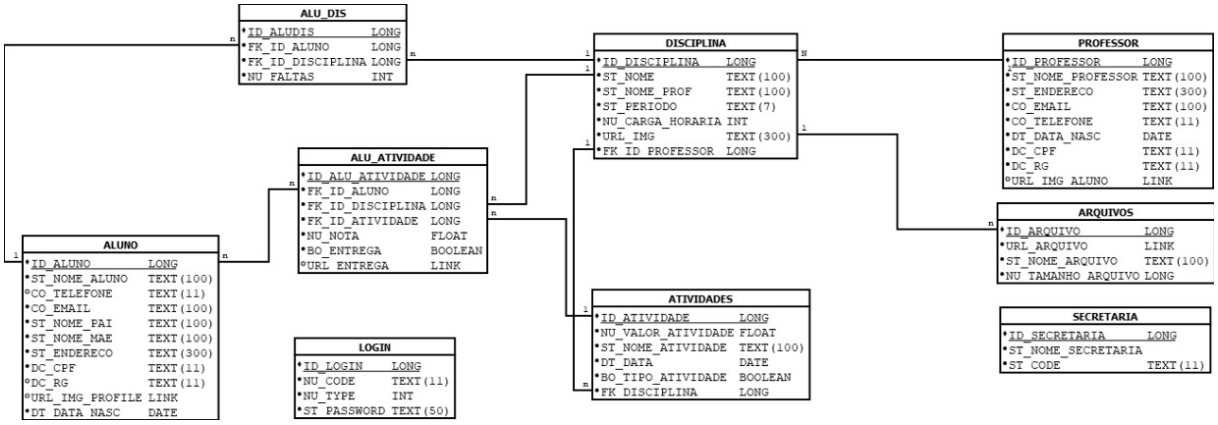
Figura 16 – Diagrama de Casos de Uso



Fonte: O autor.

Com base nas informações obtidas pelo diagrama de casos de uso foi possível planejar e desenvolver o **diagrama de classes**. Este diagrama foi de vital importância para a implementação da aplicação, pois ele possibilitou uma visualização de todas as entidades presentes no *software*, além de detalhar a interação existente entre elas. Desta forma a implementação do conjunto de entidades foi feita com base na Figura 17 (vide Anexo B).

Figura 17 – Diagrama de Classes



Fonte: O autor.

É importante salientar também quais entidades são utilizadas por cada micro serviço, de forma que existem entidades que fornecem informações para mais de um um serviço, enquanto outras entidades são específicas para determinado módulo. Através da Figura 18 é possível notar quais entidades fornecem informações a cada serviço.

Figura 18 – Entidades utilizadas pelos Micro Serviços

Fonte: O autor.

A base de dados da aplicação foi desenvolvida baseado nas informações obtidas pelos diagramas UML. Esta base foi implementada utilizando a JPA. Através da criação das entidades e o referenciamento das ligações existente entre elas, foi possível implementar toda a estrutura

do banco de dados, o que possibilita a consulta e a armazenagem de dados com base nestas entidades.

Estas entidades foram uma das bases para o funcionamento da aplicação e possibilitam a manipulação das informações de acordo com a necessidade de serviço. Isto se dá devido ao fato de que cada módulo lida com as informações, presente no banco de dados, de maneira diferente. Uma vez que cada área da aplicação necessita de informações diferentes referentes a cada entidade. Assim as consultas realizadas por cada serviço se diferenciavam dos demais.

A partir destas informações foi iniciado a implementação dos micro serviços, que possibilitaram desenvolver uma estrutura de aplicação dividida de acordo com a Figura 15. Tais serviços foram configurados de acordo com suas funcionalidades. Onde foi adicionado as bibliotecas necessárias para prover suas funcionalidades, além da configuração referente a conexão com o banco de dados e a porta de execução daquele serviço. Como pode ser vizualizado na Figura 19

Figura 19 – Application Properties

```
server.port = 8099
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/schoolar_system
spring.datasource.username=root
spring.datasource.password=root
```

Fonte: O autor.

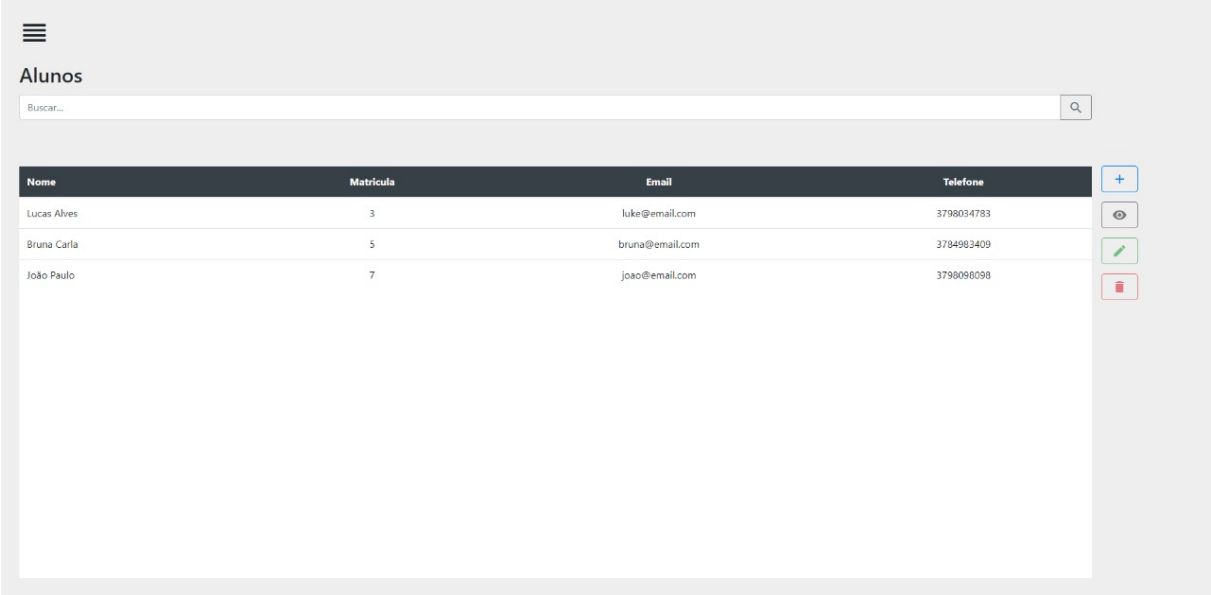
4.1 Secretaria

O primeiro serviço a ser implementado foi referente a secretaria. Esta área é responsável pela inscrição de alunos, professores e disciplinas. Para isso era necessário desenvolver um modelo padrão de interface que possibilite a realização do CRUD (*Create*, *Read*, *Update*, *Delete*) referente a cada uma destas entidades.

Esta padronização era necessária para tornar a utilização mais didática e melhorando interação do usuário com a aplicação com o intuito de tornar suas funcionalidades intuitivas e funcionais. Com base nisso, foi desenvolvido um modelo de tela para a listagem de tais entidades como demonstrado na Figura 20.

Como é possível observar a tela lista todos os alunos cadastrados. Esta listagem é feita através de uma consulta na tabela referente a entidade e é retornado todos os alunos cadastrados no banco de dados. Além disso, é possível também a busca pelo nome. Esta consulta também funciona utilizando apenas trechos do nome, de forma que será exibido todos os alunos que possuem aquele trecho descrito na busca.

Figura 20 – Listagem de Alunos



Fonte: O autor.

Este modelo de tela foi **replicado** tanto para professores quanto para disciplinas, é necessario apenas realizar as devidas modificações para se adaptar as entidades, fornecendo um CRUD padrão.

Além disso, a interface disponibiliza as opções básicas de um CRUD que são cadastro, visualização, edição e remoção de alunos, desta forma é fornecido todas as funcionalidades necessárias para gerenciar a entidade.

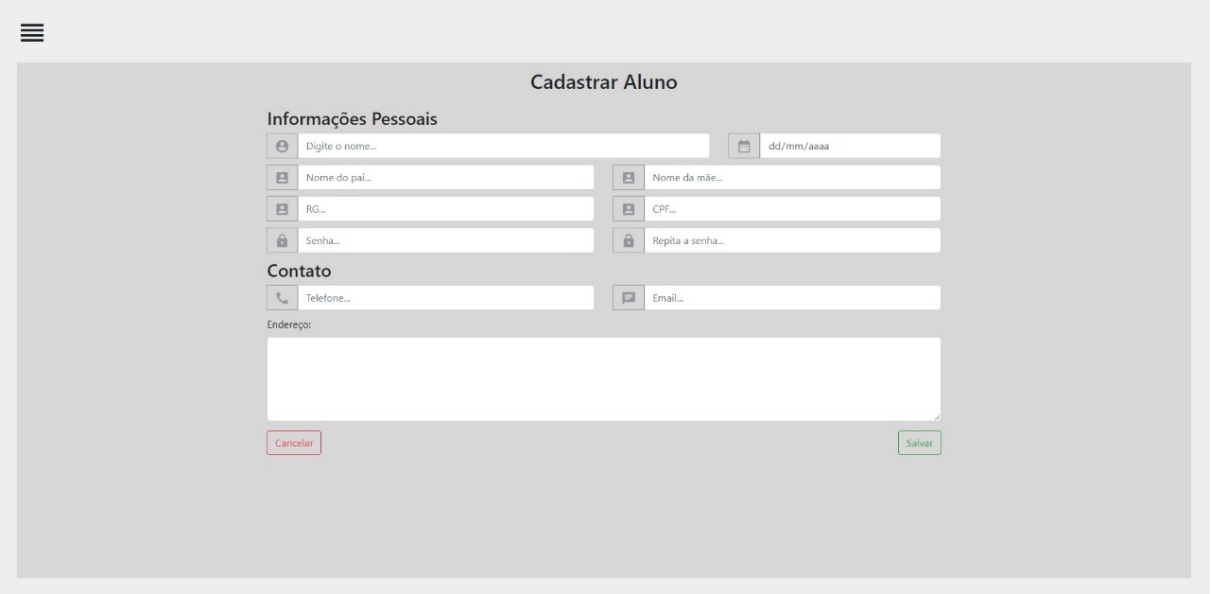
Estas funções são acessíveis através de botões presentes na parte direita da tela. Estes botões representam cada uma das funções básica de um CRUD. E quando necessário redirecionam o usuário a uma nova tela, que é responsável por exibir campos referentes às informações detalhadas da entidade. Esta nova tela pode ser visualizada na Figura 21.

As telas referentes a cadastro, visualização e edição são muito similares tanto para o aluno quanto para professor que contem campos com os dados básicos da entidade. Estes campos podem ser preenchidos ou visualizados de acordo com as informações referente a entidade que está sendo manipulada.

Já a tela referente a disciplina (Figura 22) possui algumas diferenças básicas. Uma vez que a disciplina possui ligação com demais tabelas, como por exemplo ao professor e aluno. Foi necessário o desenvolvimento de uma série de consultas que visam fornecer os dados necessários referentes aos mesmos para que seja possível realizar o cadastro, visualização e atualização de uma disciplina.

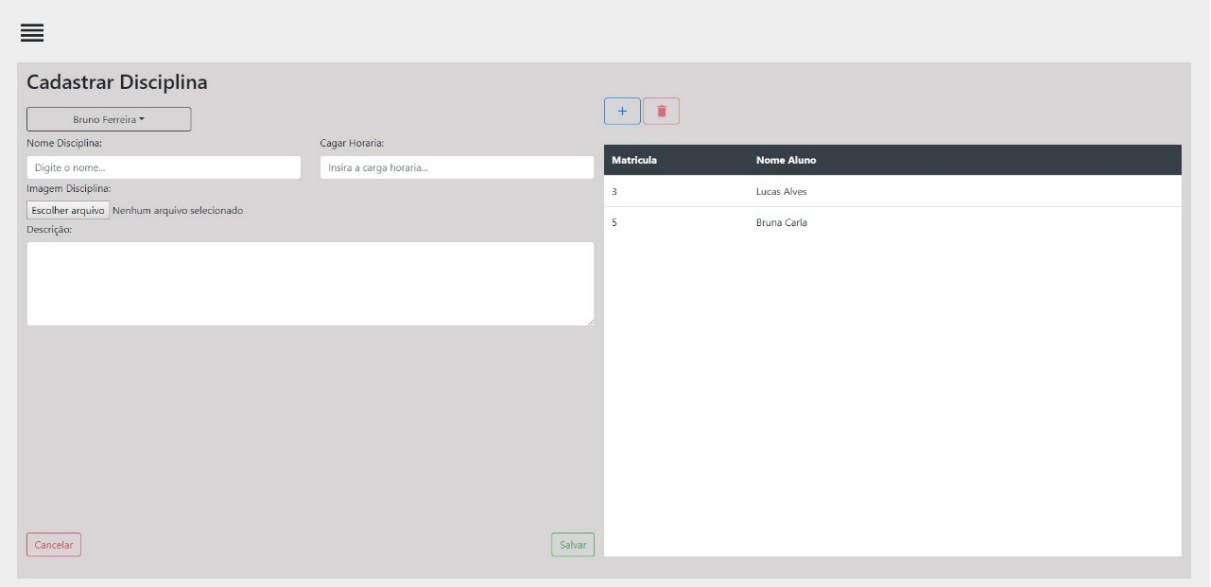
Neste caso ao realizar o cadastro ou atualização de uma disciplina. O processo realizado

Figura 21 – Cadastro de Alunos



Fonte: O autor.

Figura 22 – Cadastro de Disciplina



Fonte: O autor.

pelo *backend* é ligeiramente diferente dos demais CRUDs. Pois diferente de Aluno e Professor, ao armazenar dados referentes a disciplina é necessário realizar a interligação de suas informações utilizando tabelas auxiliares.

O principal exemplo dissa conexão é a relação entre aluno e disciplina. Esta relação é armazenada na tabela **AluDis**. Que possui campos que referenciam o ID da disciplina e o ID do aluno, classificando que aquele aluno frequenta determinada disciplina. Além disso, é contido também informações básicas daquele aluno com relação a determinada disciplina. Como por exemplo a sua frequência de faltas.

Basicamente o serviço de secretaria disponibiliza as funcionalidades citadas acima. É importante ressaltar que este serviço já pode ser implantado após o seu término, independentemente do estado dos demais serviços. As funcionalidades referentes a serviço de secretaria já estavam desenvolvidas e funcionais dentro daquele módulo que provê seus serviços de forma independente.

O único pré requisito existente para que um serviço possa fornecer suas funcionalidades é que já exista um ambiente no *frontend* que possibilite a exibição destas informações. Assim para que seja possível a entrega da aplicação em módulos é necessário que se desenvolva o *backend* e o *frontend* referente a cada serviço.

Assim uma vez que ambas partes estão funcionais, é possivel implementar tal módulo. Tal fato oferece grandes vantagens no *software* em questão, pois é vital que a secretaria cadastre os dados para que os demais serviços possam consumi-los do banco de dados. Desta forma com a implementação prévia deste módulo é possível realizar o cadastro destas informações.

4.2 Professor

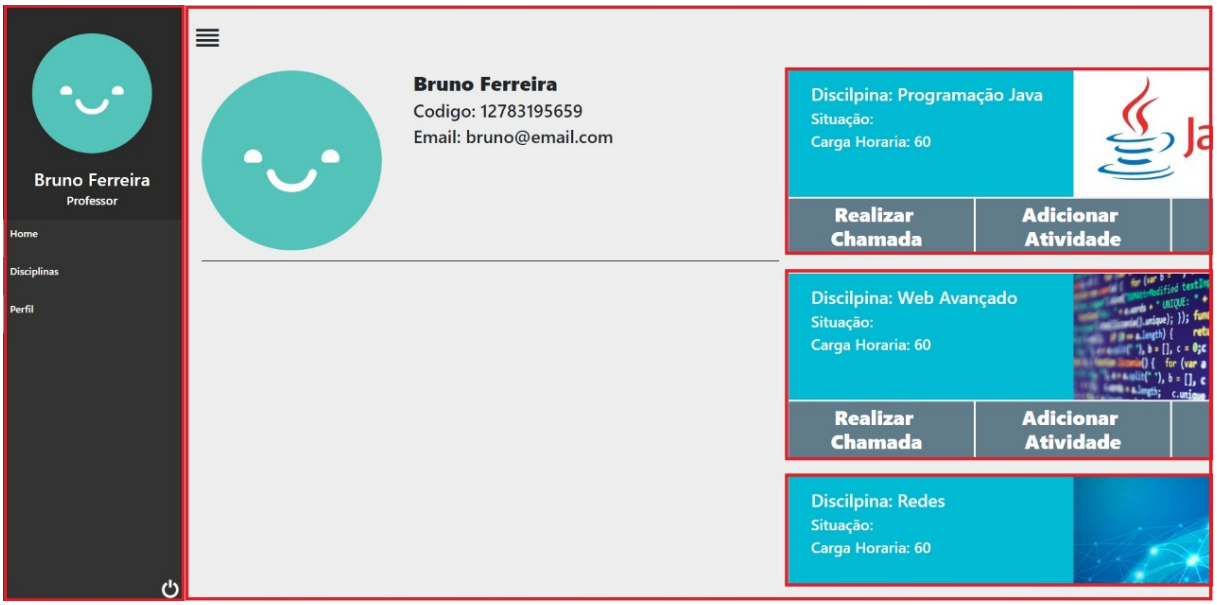
O módulo referente ao professor é responsável por fornecer um conjunto de funcionalidades ao usuário, que visam auxiliar na gerência de informações. Tais funcionalidades foram desenvolvidas com base nos pré requisitos informados pelos diagramas UML. Com base nestas informações foi desenvolvido uma interface que visa a implementação destas funcionalidades, de acordo com a proposta do diagrama de casos de uso (Figura).

Para isso, foi desenvolvido um ambiente dividido em um conjunto de componentes, que se comunicam para exibir as informações referentes aos dados que o professor deve consumir. Este ambiente exibe um conjunto de informações referentes ao professor e as disciplinas ministradas pelo mesmo. É possível observar a tela inicial referente ao professor na Figura 23.

Nela é demarcado de vermelho os componentes que fornecem serviços a esta interface. Cada componente é inserido no módulo principal e se necessário, como nos componentes de disciplina, eles são construídos com base nas informações contidas no módulo principal.

Ela demonstra a utilização de diretivas e anotações que possibilitam criar componentes

Figura 23 – Tela Inicial Professor



Fonte: O autor.

com base em dados presentes no módulo principal. Na Figura 24 podemos observar a criação de componentes com as informações das disciplinas que estão armazenadas em um vetor.

Figura 24 – Codigo gerador de componentes

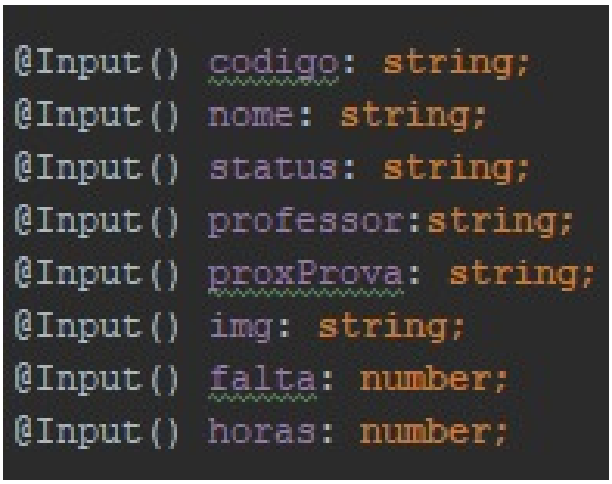
```
<div class="col-md-6 disciplina-style" id="scroll-style">
  <disciplina-perfil-p *ngFor="let item of materias"
                       [codigo]    ="item.id_disciplina"
                       [nome]      ="item.st_nome"
                       [status]    ="item.status"
                       [professor] ="item.st_nome_prof"
                       [proxProva] ="item.proxProva"
                       [img]       ="item.url_img"
                       [falta]     ="item.nu_faltas"
                       [horas]    ="item.nu_carga_horaria">
  </disciplina-perfil-p>
</div>
```

Fonte: O autor.

Dentro deste componente existe um conjunto de arquivos TS, HTML e CSS responsáveis pela interface e funcionalidades fornecidas pelo mesmo. Através das anotações contidas nele (Figura 25) é possível receber dados de outro componente e tratá-los para que as informações daquele componente sejam buscadas no *backend* por meio dos dados fornecidos externa-

mente.

Figura 25 – Anotações de entrada



Fonte: O autor.

Desta forma os diversos componentes existentes no módulo principal se comunicam para fornecer seus serviços de acordo com os dados concebidos pelo micro serviço. Essa comunicação possibilita o desenvolvimento do ambiente demonstrado na Figura 23, que fornece um conjunto de informações básicas necessárias ao professor.

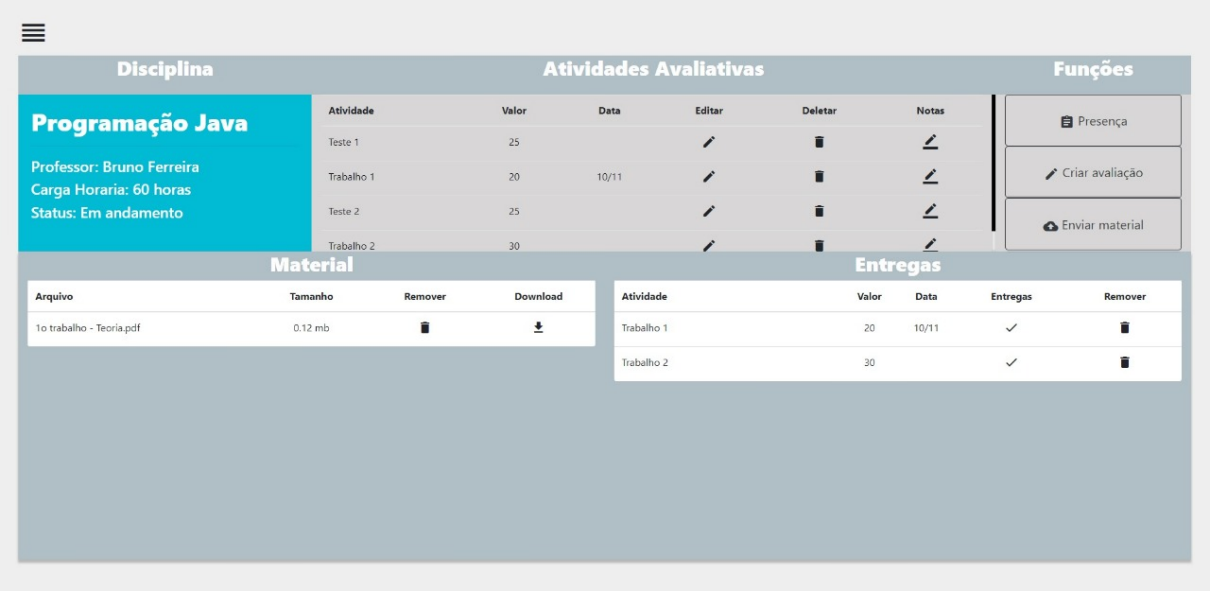
Este ambiente foi criado para que o professor tenha acesso de forma direta a um conjunto de informações vitais, de forma organizada e simples. Com o intuito de prover uma interface limpa e eficaz para a utilização do usuário.

Ao selecionar uma disciplina específica, o professor é redirecionado a um novo componente, responsável por exibir todas as informações de interesse a professor referentes aquela disciplina, além de exibir um conjunto de funcionalidades definidas na análise de requisitos. O componente é mostrado na Figura 26.

Analisando esta interface, é possível observar que é exibido informações referentes as atividades existentes na disciplina, materiais disponibilizados para o aluno, além de um conjunto de botões que possibilitam o cadastro de novas novas atividades, definição da presença diária, envio de materiais, consulta de atividades entregues e postagem de notas. Estas atividades podem ser realizadas através de um **Modal** como demonstrado na Figura 27 que disponibiliza todas as funcionalidades de forma organizada e planejada para tornar a experiência do usuário mais intuitiva e autodidata.

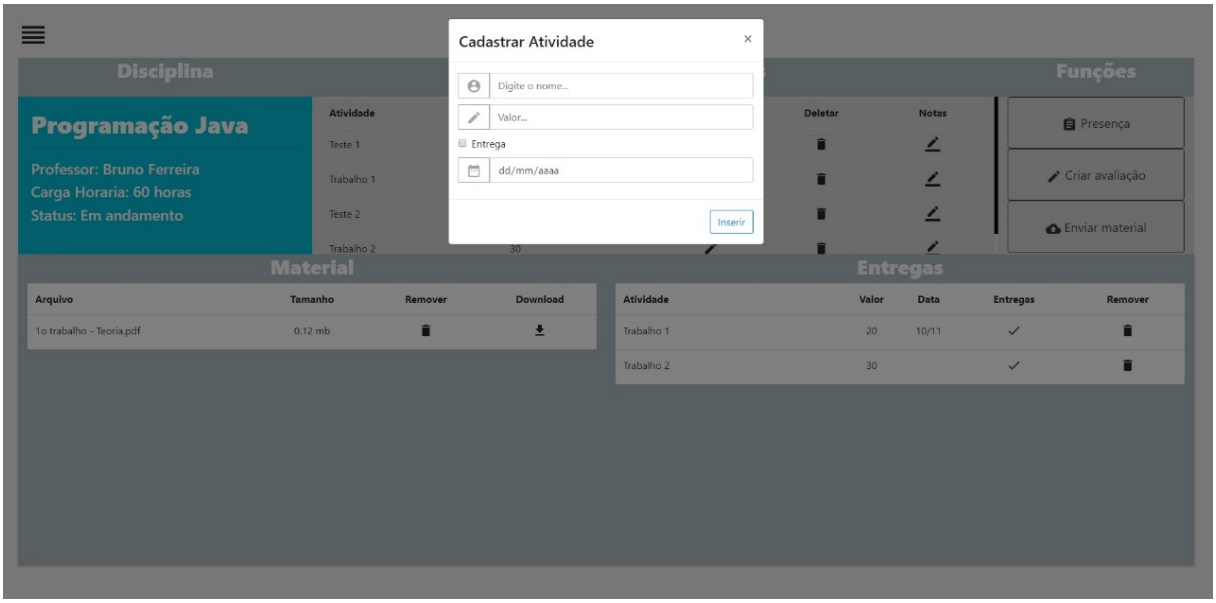
Funcionalidades como o envio de material possuem tecnologias específicas para seu funcionamento. É o caso do **Firebase**, que é utilizado para realização o *upload* e *download* de arquivos, mantendo os mesmos disponíveis em uma base de dados segura, funcional e de simples acesso. Porém, para a utilização desta tecnologia, foi necessário configurar o acesso

Figura 26 – Disciplina Professor



Fonte: O autor.

Figura 27 – Modal Disciplina Professor



Fonte: O autor.

a uma base de dados do *firebase* criada especificamente para a aplicação. Tal configuração foi realizada como demonstrado na Figura 28.

Figura 28 – Configuração *Firebase*

Fonte: O autor.

Após realizar esta configuração era possível, através das bibliotecas fornecidas pela ferramenta, realizar o *upload* de arquivos recebendo como retorno a URL de acesso ao mesmo. Desta forma esta URL é salva no banco de dados podendo ser acessada quando necessário.

É importante ressaltar que quando uma nova atividade é criada é fornecido ao professor a opção de disponibilizar esta atividade para entrega. Desta forma, o aluno pode transcrever a atividade em um documento e dentro do prazo limite enviá-la ao professor, que por sua vez terá acesso a todas as atividades recebidas, desta forma é possivel automatizar um processo antes feito manualmente.

Para que isso seja possível é necessário que ao criar uma nova atividade e automaticamente atrelar esta atividade aos alunos que cursam a disciplina em questão. Isto é feito através da tabela **AluAtividade**. Que possui um conjunto de campos nos quais os dados do aluno com relação a determinada atividade de determinada disciplina serão armazenados. Dados estes como sua nota e a URL do arquivo entregue, caso a atividade permita a entrega de arquivos.

Ou seja, ao criar uma nova atividade é necessário automaticamente criar a ligação desta atividade com os alunos. Todo este processo é automaticamente feito dentro do micro serviço. Pois a partir do momento que os dados referentes a atividade chegam ao *backend*, um conjunto de funções é encarregado de realizar toda a manipulação necessária.

Como não poderia ser diferente a postagem das notas realiza a atualização dos dados presentes na tabela AluAtividade, de forma que automaticamente estas notas serão exibidas ao aluno. Este ambiente também é responsável por fornecer uma funcionalidade de validação da presença diária. Para isso o professor informa, com base na lista de alunos que cursam a disciplina, quais daqueles não compareceram em determinada data. Ao enviar estes dados para o serviço, o mesmo processa e incrementa o número de faltas quando necessário.

Ao término do desenvolvimento destas atividades básicas é possível implantar este novo módulo e torná-lo funcional. Uma vez que todas funções referentes ao professor estão encapsu-

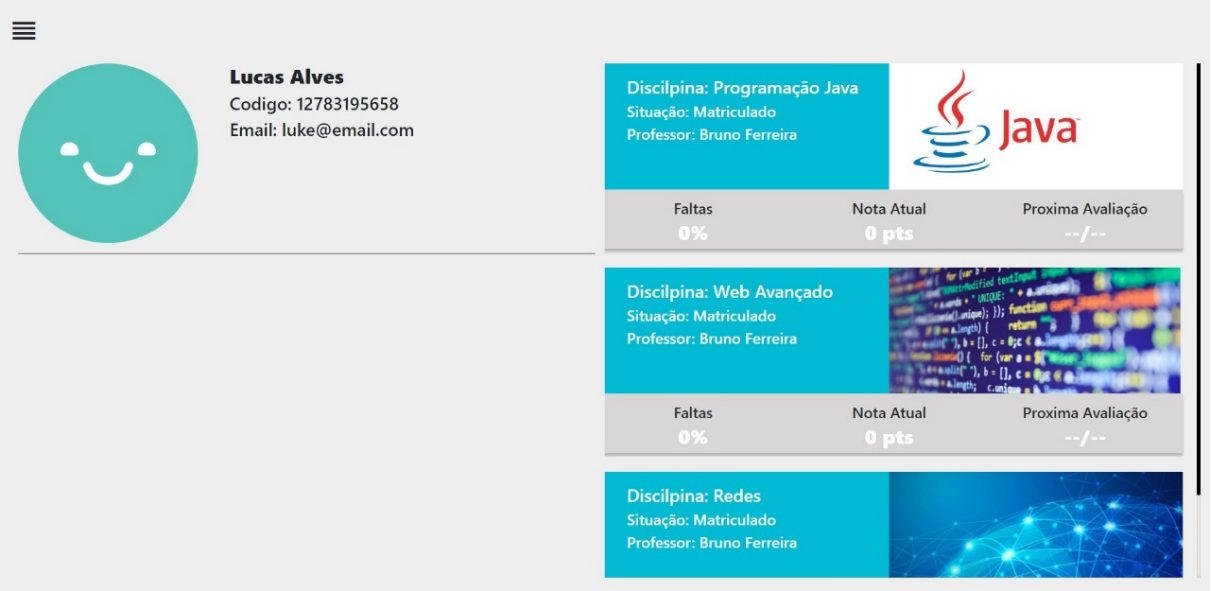
ladas dentro deste serviço, a área do *software* referente ao mesmo pode funcionar independente do resto da aplicação.

4.3 Aluno

O micro serviço referente ao aluno foi o ultimo dentre as três areas de atuação a serem implementadas. Isso se dá pelo fato de que muitas das informações consumidas por ele são cadastradas pela secretaria e e professor. Desta forma é usual que tais áreas estejam bem desenvolvidas, para assim desenvolver como este módulo irá lidar com os dados.

Assim com base nas informações coletadas através do levantamento de requisitos e da documentação UML, iniciou-se o desenvolvimento deste módulo. A fim de padronizar o modelo da aplicação, foi desenvolvido uma interface inicial similar a existente no serviço de professor. Porém, exibindo informações diferentes referentes às disciplinas nas quais o aluno está cadastrado. Uma vez que o tipo de usuário é outro, se torna trivial que as informações que o mesmo deseja consumir são diferentes. A Figura 29 possibilita a visualização desta interface:

Figura 29 – Tela Inicial Aluno



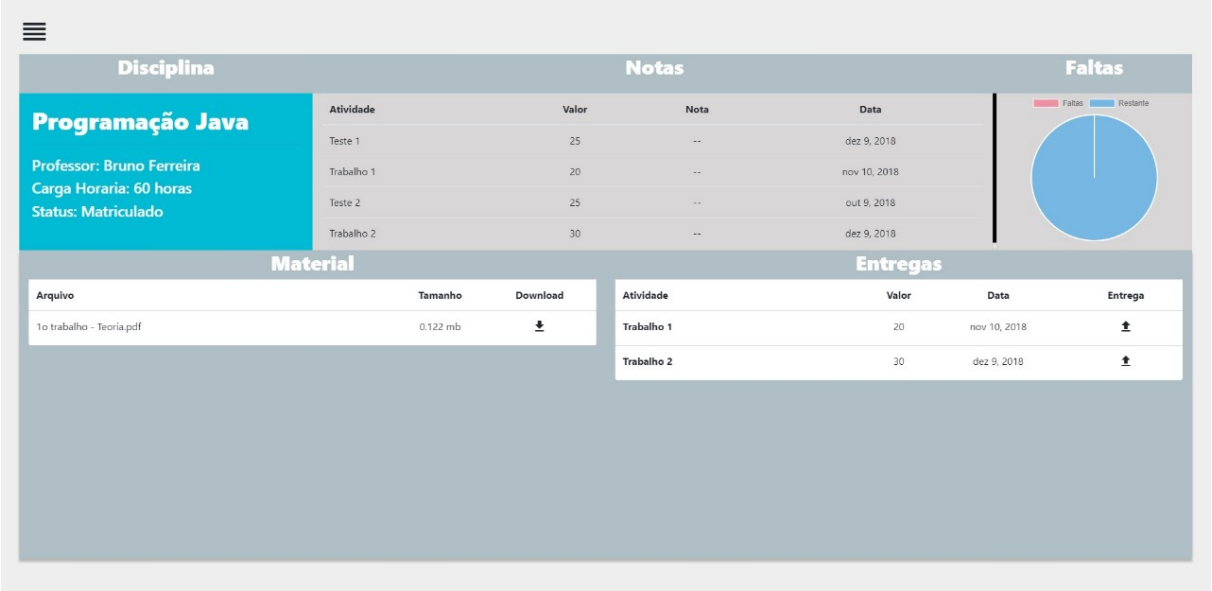
Fonte: O autor.

Como é possível possui uma componentização similar a da Figura 23, porém com componentes diferentes, que neste caso oferecem as funcionalidades e informações necessárias aos alunos. Desta forma o usuário possui acesso básico as informações essenciais do seu interesse, isso evita a necessidade de adentrar no detalhamento de tal disciplina.

Caso o usuário deseje consultar informações mais detalhadas referente a disciplina ele pode acessá-la selecionando-a na lista de disciplinas. Ao ser redirecionado para a tela referente a

uma disciplina específica. O mesmo irá ter contato com todas as informações necessárias sobre tal matéria. Esta interface pode ser visualizada na Figura 30:

Figura 30 – Disciplina Aluno



Fonte: O autor.

Ao observar a Figura 30 é possível notar um conjunto de informações relevantes para o usuário. Como por exemplo a disponibilidade das atividades e dados referentes às mesmas, além de porcentagem de faltas e outros dados. Tais informações foram disponibilizadas de forma simplificada afim de facilitar a interação do usuário.

Outro ponto que vale ressaltar é a possibilidade de entrega de atividades, uma vez que a atividade possibilita a entrega ela será exibida a aluno. Desta forma é possivel que o mesmo realize o *upload* de arquivos, via **firebase**, correspondentes àquela atividade. Ao realizar o *upload* o *backend* trata os dados e os disponibiliza para acesso do professor.

Além disso, é possível observar a disponibilidade de arquivos de apoio para disciplina, que disponibiliza o *download* do mesmo de acordo com a funcionalidade proposta pela aplicação.

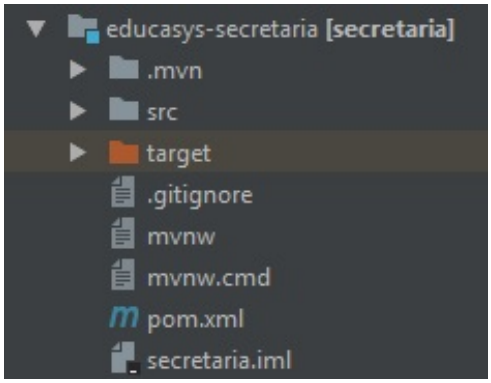
Através deste conjunto de funcionalidades, o aluno tem controle sobre todas atividades propostas nos pré-requisitos da aplicação que são encapsulados em um micro serviço que é responsável por gerenciar esta área de atuação do *software*.

4.4 Micro serviços

O **Spring Boot Framework** oferece suporte ao desenvolvimento de micro serviços. Desta forma ele disponibiliza um conjunto de ferramentas que possibilitam a implementação dos mesmos. Para isso as IDEs que suportam o *framework* disponibilizam a criação de novos módulos que são referentes aos micro serviços.

A IDE utilizada para desenvolvimento do prototipo foi a **IntelliJ IDEA Ultimate**, que possui suporte a todas tecnologias utilizadas nesta aplicação para possibilitar o maior desempenho no processo de desenvolvimento. Ela possibilita também a criação destes módulos de forma simplificada que gera a estrutura de dados mostrada na Figura 31:

Figura 31 – Estrutura de um micro serviço



Fonte: O autor.

Estes arquivos são responsáveis pela manutenção e execução daquele serviço. Desta forma é possivel realizar a configuração e adição de novas bibliotecas de forma independente para cada micro serviço, a fim de disponibilizar apenas as ferramentas necessárias especificamente para aquele módulo. A manipulação de dependências destes módulos é feita através do **Maven**. Uma ferramenta que possibilita a inserção de dependências de forma simplificada, alem de possuir suporte a um conjunto de bibliotecas que auxiliam na conexão com o banco de dados e demais tarefas.

Para o funcionamento adequado da aplicação é necessário configurar as definições básicas de execução no arquivo **application.properties** (Figura 19). Este arquivo é responsável pela definição das principais configurações do serviço. Como por exemplo a porta na qual o serviço será executado, se o mesmo será executado de forma local ou não além da configuração de conexões como por exemplo com a base de dados.

Com as devidas configurações efetuadas. É possível dar início a execução da aplicação. Cada módulo possui uma classe de inicialização, responsável por realizar as configurações existentes e dar início a execução do micro serviço. Esta classe pode ser visualizada na Figura

32.

Figura 32 – Classe de inicialização

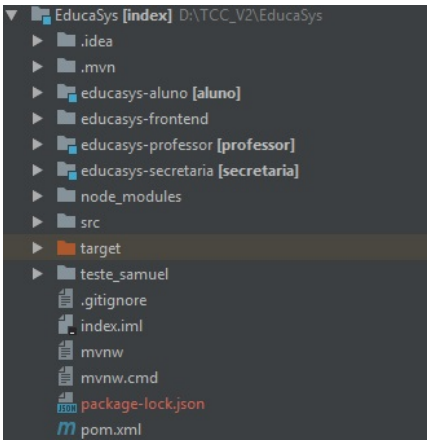
```java
package com.tcc.index;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EducasysApplication {

    public static void main(String[] args) { SpringApplication.run(EducasysApplication.class, args); }
}
```

Fonte: O autor.

Como é possível observar na Figura 32, a classe responsável por inicializar o micro serviço possui uma anotação que indica que aquela classe é referente a um micro serviço.

Desta forma ao criar, configurar e programar todos os módulos obteve-se uma estrutura dividida entre as áreas de atuação da aplicação, como pode ser visto na Figura 33. Dentro de cada pacote referente a um micro serviço possui os arquivos necessários (Figura 31) para a execução daquela aplicação, de forma independente a qualquer arquivo ou modulo externo.

Figura 33 – Estrutura de código da aplicação



Fonte: O autor.

4.4.1 Testes

Esta seção tem como intuito descrever os testes realizados para validar os objetivos propostos no início deste documento. Através desses testes é possível compreender melhor o funcionamento da aplicação e das tecnologias aplicadas na mesma alem de demonstrar a persistência da aplicação em situações adversas, como por exemplo a falha de um serviço.

4.4.1.1 Implantação por módulos

O primeiro teste realizado consistiu na implantação dos módulos separadamente. Ao término da implementação do módulo de secretaria, tanto no *frontend* quanto no *backend*, o módulo foi implantado para a realização de testes, desta forma antes mesmo do término dos demais serviços, o módulo de secretária já estava operando e fornecendo suas funcionalidades.

Através deste teste foi possível observar que a entrega de uma aplicação com base em micro serviços pode ser feita modulo a modulo, uma vez que cada módulo funciona sozinho é possível usufruir de suas funcionalidades sem a necessidade da conclusão dos demais.

4.4.1.2 Persistencia

O segundo teste realizado consistiu na validação da persistencia da aplicação. Para a realização do mesmo foi implementado um ambiente de teste onde o módulo de professor não estaria em execução. Desta forma seria possível validar o funcionamento e a independência dos demais módulos em relação ao serviço inoperante.

Após a execução dos módulos de aluno, secretaria e index, além é claro da interface de *frontend*, foi realizado o login de um usuário do tipo aluno. Ao ser validado o login, o usuário obteve acesso a todas informações referentes ao seu ambiente e o mesmo ocorreu ao usuário de secretaria. Já ao tentar realizar o login para um usuário do tipo professor, a interface foi incapaz de receber os dados deste usuário, uma vez que o seu módulo não estava em execução.

Através deste teste foi possível demonstrar a possibilidade de execução de cada serviço de forma independente, assim as funcionalidades dos serviços operantes podem ser acessadas mesmo em casos de inoperância dos demais módulos.

4.4.1.3 Acesso remoto aos serviços

Também foram realizados testes em diferentes máquinas, com o intuito de atestar a modularidade e total independência do micro serviço. Para isto foi preparado um ambiente que utiliza duas máquinas conectadas em uma mesma rede. Foi carregada uma copia do *software* em cada maquina, onde a maquina A iria executar um modulo e a maquina B executaria os demais alem do *frontend*.

Desta forma, todas as requisições do *frontend* ao módulo executado na máquina A foram configuradas para serem efetuadas de forma remota através do IP da máquina. A configuração da aplicação na máquina A pode ser visualizada na Figura 34. Esta configuração possibilita que os demais módulos possam acessar a aplicação executada na máquina A. Já a configuração das requisições do *frontend* podem ser visualizada na Figura 35. É necessário configurar estas requisições para que elas saibam identificar o endereço de IP que devem solicitar as informações.

Desta forma foi possível comunicar-se remotamente com o serviço executado na máquina A e mostrar que a modularidade possibilita que os serviço sejam executados em máqui-

Figura 34 – Configuração Máquina A

```
server.port = 8099
server.address = 192.168.1.109
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/schoolar_system
spring.datasource.username=root
spring.datasource.password=root
```

Fonte: O autor.

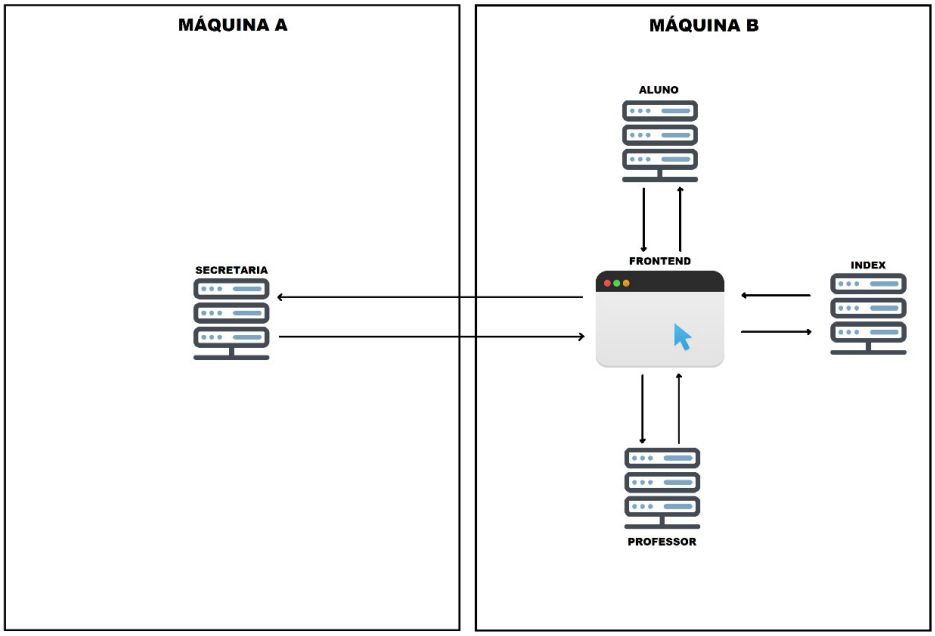
Figura 35 – Configuração Frontend

```
private urlD = 'http://192.168.1.109:8099/getDisciplinas';
private urlDById = 'http://192.168.1.109:8099/getDisciplinaById';
private urlDSet = 'http://192.168.1.109:8099/saveDisciplina';
private urlDDelete = 'http://192.168.1.109:8099/deleteDisciplina';
private urlDSearch = 'http://192.168.1.109:8099/searchDisciplinas';

private urlA = 'http://192.168.1.109:8099/getAlunos';
private urlAById = 'http://192.168.1.109:8099/getAlunoById';
private urlASet = 'http://192.168.1.109:8099/saveAluno';
private urlAUpdate = 'http://192.168.1.109:8099/updateAluno';
private urlADelete = 'http://192.168.1.109:8099/deleteAluno';
private urlASearch = 'http://192.168.1.109:8099/searchAlunos';

private urlP = 'http://192.168.1.109:8099/getProfessores';
private urlPById = 'http://192.168.1.109:8099/getProfessorById';
private urlPSet = 'http://192.168.1.109:8099/saveProfessor';
private urlPUpdate = 'http://192.168.1.109:8099/updateProfessor';
private urlPDelete = 'http://192.168.1.109:8099/deleteProfessor';
private urlPSearch = 'http://192.168.1.109:8099/searchProfessores';
```

Fonte: O autor.

nas diferentes, uma vez que sua comunicação é configurada é indiferente para aplicação onde o serviço está sendo executado. Esta comunicação é demonstrada na Figura 36. Com base nesta figura é possível é possível abstrair a comunicação efetuada pelo módulo de secretária com o *frontend*. Onde cada bloco refere-se aos serviços executados em cada máquina. Tendo os serviços da máquina B executados de forma local e o serviço da máquina A configurado para fornecer dados remotamente, isso evidencia a modularidade de cada serviço e confirmando a proposta da arquitetura.

Figura 36 – Esquematico comunicação remota



Fonte: O autor.

5 CONSIDERAÇÕES FINAIS

Durante o desenvolvimento do protótipo foi possível compreender a proposta desta arquitetura que provê um novo processo de desenvolvimento para aplicações WEB. Onde um grande esforço é empregado a fim de modularizar o projeto nas etapas de planejamento, desenvolvimento e testes.

Ao desenvolver o *software* baseado nessa arquitetura, notou-se a priorização na sua modularidade e independência, que possibilita o foco específico de desenvolvimento em cada módulo tornando maior o nível de abstração e detalhamento do mesmo. Uma vez que aquele módulo é planejado como uma aplicação específica focada em sua área de abrangência diferente de grandes aplicações monolíticas que abrangem diversas áreas.

Através dos testes realizados com a aplicação foi possível também validar a robustez da arquitetura. Dado o fato de que mesmo os módulos sendo executados em máquinas diferentes suas funcionalidades continuam sendo oferecidas ao *frontend*. Isso se dá devido ao fato da autonomia destes serviços, que conseguem exercer suas tarefas independente dos outros módulos do projeto. Isso caracteriza também uma maior prevenção contra falhas, a qual possibilita que a aplicação não pare de funcionar por completo quando um módulo venha a falhar, possibilitando a correção de bugs e atualização dos módulos sem afetar os demais.

Além disso, foi mostrado a integração desta arquitetura com *frameworks* para *frontend* como Angular 5 e Bootstrap, que são ferramentas poderosas e atuais que fornecem um conjunto de funcionalidades para a melhor interação com o cliente.

Ao analisar o projeto após sua conclusão, foi validado todos os requisitos básicos propostos. O que possibilitou o compreendimento do funcionamento desta arquitetura, frisando suas funcionalidades e possibilidades. Contudo, foi demonstrando que a modularidade provou-se eficaz para o isolamento de falhas e desacoplamento de responsabilidades. Esse desacoplamento possibilita o isolamento de um conjunto de funcionalidades e serviços em um determinado micro serviço.

Porém, este protótipo possibilita uma série de futuras melhorias que podem vir a ser implantadas em projetos futuros. Um exemplo disso é a utilização de uma base de dados distribuída, que possibilita a independência total dos micro serviços, uma vez que não haveria um banco de dados centralizado para acesso de informações. Além de melhorias na interface e possibilidade de desenvolvimento de novos módulos que possam gerenciar outras áreas, possibilitando assim o desenvolvimento da aplicação.

É importante também melhorar a arquitetura desenvolvida, tornando seus módulos menos acoplados e independentes em relação a sua base de dados. Desta forma a aplicação pode

trabalhar sem a dependência do banco de dados. Além disso, pode ser aplicado estudos para melhorar a interface com o usuário, tornando a experiência de utilização mais simples.

O *software* se encontra disponivel em um repositorio do GitHub [1] onde é possivel realizar o acesso e modificação desde que os mesmos estejam de acordo com a licença MIT.

[1] Disponível em: <https://github.com/lukecorf/EducaSys> Acesso em Junho de 2018

REFERÊNCIAS

ALBINO, J. P. et al. Design de interfaces para web baseados no sistema de grade do bootstrap 3 interface design for web based on grid system bootstrap 3. 2015. Citado na página 34.

ANTONOV, A. *Spring Boot Cookbook*. [S.l.]: Packt Publishing Ltd, 2015. Citado na página 33.

APPDYNAMICS. *Introducing Microservices iQ*. 2016. Disponível em: <https://blog.appdynamics.com/product/introducing-microservices-iq/>. Acesso em: Maio de 2018. Citado na página 29.

BALLEM, M. *Como funciona um webservice REST*. 2012. Disponível em: <http://matera.com/br/2012/10/22/como-funciona-um-webservice-rest/>. Acesso em: Maio de 2018. Citado na página 33.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. *Hypertext transfer protocol–HTTP/1.0*. [S.l.], 1996. Citado na página 28.

BORTOLI, N. d. S. de; RUFINO, R. R. Conceito para o desenvolvimento web utilizando spring boot, bootstrap e angular js. 2016. Citado na página 35.

CAELUM. *Arquitetura de microserviços ou monolítica?* 2015. Disponível em: <http://blog.caelum.com.br/arquitetura-de-microservicos-ou-monolitica/>. Acesso em: Maio de 2018. Citado 2 vezes nas páginas 25 e 26.

CROCKFORD, D. The application/json media type for javascript object notation (json). 2006. Citado na página 28.

D'AGOSTINO, D. et al. A microservice-based portal for x-ray transient and variable sources. *PeerJ PrePrints*, v. 5, p. e2519, 2017. Citado na página 30.

DMITRY, N.; MANFRED, S.-S. On micro-services architecture. *International Journal of Open Information Technologies*, . , v. 2, n. 9, 2014. Citado na página 24.

EDUREKA. *Angular Tutorial*: Getting started with angular 4. 2017. Disponível em: <https://www.edureka.co/blog/angular-tutorial/>. Acesso em: Maio de 2018. Citado na página 40.

JAQUES, R. *O que é um Framework? Para que serve?* 2016. Disponível em: <http://www.phpit.com.br/artigos/o-que-e-um-framework.phpit>. Acesso em: Maio de 2018. Citado na página 32.

JAVABEAT. *Bootstrap Setup*. 2014. Disponível em: <https://javabeat.net/bootstrap-setup/>. Acesso em: Maio de 2018. Citado na página 36.

JAVASAMPLEAPPROACH. *Spring Boot + React Redux + MySQL CRUD*. 2018. Disponível em: <http://javasampleapproach.com/spring-framework/spring-data-rest/spring-boot-react-redux-mysql-crud-example>. Acesso em: Maio de 2018. Citado na página 44.

JÚNIOR, O. A. Arquitetura de micro serviços: uma comparação com sistemas monolíticos. Universidade Federal da Paraíba, 2017. Citado na página 29.

KILOBYTE. *How we build a good responsive Design?* 2017. Disponível em: <https://www.kilobytetech.com/build-good-responsive-design/>. Acesso em: Maio de 2018. Citado na página 36.

KOSKINEN, M. et al. Microservices and containers: benefits and best practices. Turun ammattikorkeakoulu, 2016. Citado na página 31.

KUMAR, K. et al. Implementing smart home using firebase. *International Journal of Research in Engineering and Applied Sciences*, Euro Asia Research and Development Association, v. 6, n. 10, p. 193–198, 2016. Citado na página 43.

MICROSOFT. *ASP.NET - Single-Page Applications*: Build modern, responsive web apps with asp.net. 2013. Disponível em: <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>. Acesso em: Maio de 2018. Citado na página 37.

MIKOWSKI, M. S.; POWELL, J. C. Single page web applications. *B and W*, 2013. Citado na página 37.

MILANI, A. *MySQL-guia do programador*. [S.l.]: Novatec Editora, 2007. Citado na página 44.

PALIARI, M. *Persistência com Spring Data JPA*. 2012. Disponível em: <https://www.devmedia.com.br/persistencia-com-spring-data-jpa/24390>. Acesso em: Maio de 2018. Citado na página 34.

RAMOS, R. A. *Treinamento prático em UML*. [S.l.]: Universo dos Livros Editora, 2006. Citado na página 50.

SIMKHADA, K. Transitioning angular 2 user interface (ui) into react. Metropolia Ammattikorkeakoulu, 2017. Citado na página 39.

SODHANA. *Angular 2 Routes Tutorial With Example*. 2016. Disponível em: <http://blog.sodhanalibrary.com/2016/09/angular-2-routes-tutorial-with-example.html#.WvYORxzQ_CI>. Acesso em: Maio de 2018. Citado na página 41.

TEROPA. *Refactoring Angular Apps to Component Style*. 2015. Disponível em: <https://teropa.info/blog/2015/10/18/refactoring-angular-apps-to-components.html>. Acesso em: Maio de 2018. Citado na página 39.

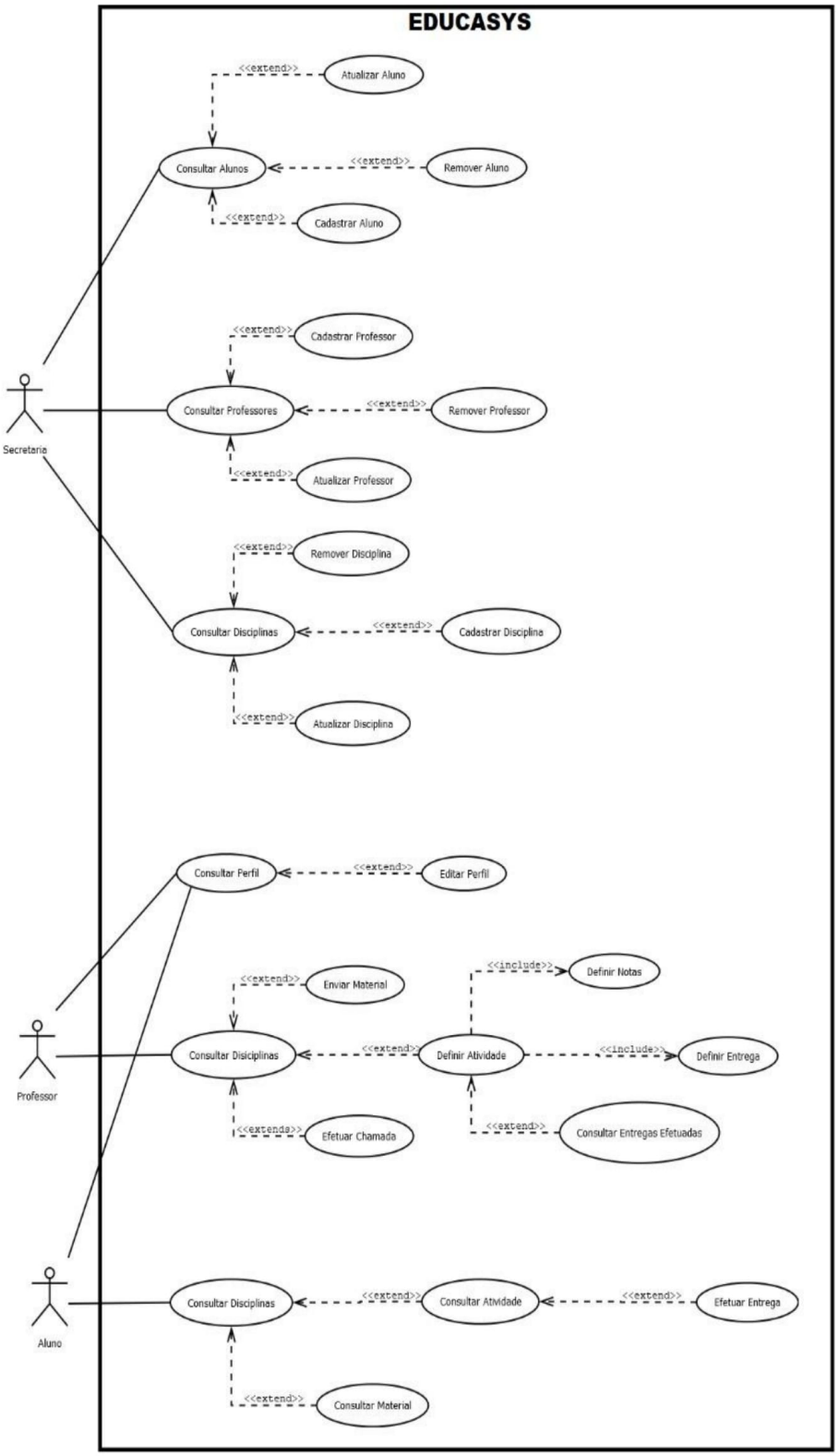
THÖNES, J. Microservices. *IEEE Software*, IEEE, v. 32, n. 1, p. 116–116, 2015. Citado na página 26.

UBIDOTS. *Top 4 online tools to simulate HTTP requests*. 2017. Disponível em: <https://ubidots.com/blog/top-4-online-tools-to-simulate-http-requests/>. Acesso em: Maio de 2018. Citado na página 28.

WLCONSULTORIA. *PHP- O que é e como funciona ?* 2017. Disponível em: <https://www.cursos.wlconsultoria.net/blog/php-o-que-e-e-como-funciona/>. Acesso em: Maio de 2018. Citado na página 23.

Anexos

ANEXO A – DIAGRAMA DE CLASSES



ANEXO B – DIAGRAMA DE CASOS DE USO