CENTRO UNIVERSITÁRIO DE ANÁPOLIS – UniEVANGÉLICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

DAVID FERRAZ PIRES
JONATHAN RAMOS NASCIMENTO

EDUPLAN – SISTEMA DE CRIAÇÃO E GERENCIAMENTO DE PLANOS DE ENSINO

ANÁPOLIS
2019

DAVID FERRAZ PIRES
JONATHAN RAMOS NASCIMENTO

EDUPLAN – SISTEMA DE CRIAÇÃO E GERENCIAMENTO DE PLANOS DE ENSINO

Trabalho de Conclusão de Curso II apresentado como requisito parcial para a conclusão da disciplina de Trabalho de Conclusão de Curso II do curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Orientador(a): Prof. Me. Millys Fabrielle Araujo Carvalhaes.

ANÁPOLIS
2019

DAVID FERRAZ PIRES
JONATHAN RAMOS NASCIMENTO

EDUPLAN - SISTEMA DE CRIAÇÃO E GERENCIAMENTO DE PLANOS DE ENSINO

Trabalho de Conclusão de Curso II apresentado como requisito parcial para a obtenção de grau do curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Aprovado(a) pela banca examinadora em 4 de dezembro de 2019, composta por:

Millys Fabrielle Araújo Carvalhaes
Presidente da Banca

Luciana Nishi
Prof(a). Convidado(a)

Viviane Carla Batista Pocivi
Prof(a). Convidado(a)

RESUMO

Este projeto tem como objetivo desenvolver uma aplicação *web*, para facilitar e agilizar a criação e gerenciamento dos planos de ensino dos cursos de bacharelados de Computação da UniEVANGÉLICA. A criação do plano de ensino é uma das atividades realizada por todos os professores de cursos superiores ao início dos semestres, tomando muito de seu tempo. O coordenador pedagógico dos cursos avalia todos os planos e retorna um feedback ao professor, aceitando ou pedindo que seja revisado o plano criado se houver algum erro. Como resultado espera-se que toda a atividade de criação e gerenciamento do plano de ensino seja realizado através da plataforma web Eduplan, na qual os docentes terão acesso aos planos de ensino a eles alocados, para que seja feita a edição dos campos que a ele for atribuído dentro do plano, e no final seja salvo para que o coordenador possa fazer a avaliação do plano dentro do próprio sistema, aprovando ou retornando um *feedback* com quais campos devem ser corrigidos, ou aprovando o plano e liberando para a impressão.

Palavras-chave: Plano de Ensino. Aplicação Web. Gerenciamento.

ABSTRACT

This project aims to develop a web application, to facilitate and speed up the creation and management of teaching plans of UniEVANGÉLICA's Bachelor Degree Computer courses. The creation of the teaching plan is one of the activities performed by all teachers of higher education courses at the beginning of semesters, taking much of their time. The course coordinator evaluates all plans and returns feedback to the teacher, accepting or asking to have the plan created revised if there are any errors. As a result, it is expected that all the teaching plan creation and management activity will be carried out through the Eduplan web platform, in which teachers will have access to the teaching plans allocated to them, so that the fields that can be edited will be edited. is assigned within the plan, and ultimately saved so that the coordinator can evaluate the plan within the system itself by approving or returning feedback with which fields to correct, or approving the plan and releasing it for printing.

Keywords: *Teaching Plan. Web application. Management.*

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API	*Application Programming Interface*
CLI	*Command-Line Interface*
FDD	*Feature Driven Development*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronic Engineers*
JSON	*JavaScript Object Notation*
MVC	*Model-View-Controller*
NoSQL	*Not Only SQL*
PDF	*Portable Document Format*
PPC	*Projeto Pedagógico de Curso*
SPA	*Single-Page Applications*
SVN	*Subversion*

Sumário

1 Introdução

O Plano de Ensino é um plano de ação; é o registro do planejamento das ações pedagógicas para o componente curricular durante o período letivo. É um instrumento didático-pedagógico e administrativo de elaboração e uso obrigatório (IFPR, 2014). Ele é um documento redigido todo início de semestre, no qual os professores fazem um planejamento da disciplina a ser ministrada, e interligam os objetivos, o conteúdo e suas metas.

No currículo tradicional, os Planos de Ensino se apresentaram como forma habitual de organização do trabalho docente. E neles estão contidos dados de identificação (turma, turno, disciplina, número de alunos, carga horária, etc.), ementa, objetivos (gerais e específicos), tópicos de conteúdo, metodologia, avaliação e bibliografia. Estes construídos individualmente pelo docente responsável pela disciplina, entregues à coordenação e, algumas vezes, apresentados aos alunos no início do ano ou semestre letivo.

Todas as informações são agrupadas em um *template* fornecido pela instituição, e o professor tem a responsabilidade de editar alguns campos, enquanto outros são fixos de acordo com o PPC (Projeto Pedagógico do Curso). Após a finalização, o docente envia o plano de ensino para a coordenação pedagógica do curso para que seja feita a avaliação individual de cada plano de ensino, a qual verifica se o plano está em conformidade com o PPC, se o calendário está de acordo com todas as atividades do curso e se corresponde a todo direcionamento institucional que é recebido, e caso não esteja o plano de ensino é devolvido para o professor solicitando as alterações, este processo é feito até que seja aprovado pela coordenação pedagógica do curso e posteriormente liberado para a impressão.

Conforme entrevistas com professores dos Cursos de Bacharelados em Computação da instituição UniEVANGÉLICA - Centro Universitário de Anápolis, foi identificado que uma das tarefas que consome muito tempo dos docentes e também da coordenação pedagógica no início de todos os períodos letivos é a elaboração dos planos de ensino das disciplinas, no desenvolvimento de um plano de ensino é necessário dedicação tanto dos docentes quanto da coordenação pedagógica, se houver problemas no plano de ensino pode ocorrer problemas desde a transferência de alunos para outros cursos e universidades, como também problemas com a validação do diploma do curso, porque tanto na transferência quanto na validação do diplomas são avaliados os planos de ensinos das disciplinas para identificar se o conteúdo da disciplinas atende os requisitos para o aproveitamento da matéria e para a conclusão do curso, e a avaliação mais importante da universidade que é feita pelo MEC (Ministério da Educação) faz a avaliação do plano de ensino e verifica se está de acordo com o PPC.

Dada a problemática descrita acima, como uma aplicação pode auxiliar os professores e coordenadores na criação e gerenciamento dos Planos de Ensino?

Este estudo tem como objetivo desenvolver uma aplicação *Web* com a finalidade de auxiliar os docentes e a coordenação na gestão de Planos de Ensino dos cursos de Bacharelados em computação na instituição de ensino UniEVANGÉLICA. Para a concretização desse objetivo é necessário fazer o levantamento de requisitos do projeto, definir as ferramentas a serem utilizadas para o desenvolvimento da aplicação *Web*, desenvolver o projeto de acordo com o que foi levantado e com as boas práticas de programação, realizar a validação do que foi desenvolvido de modo continuo e integrado.

A contribuição desse estudo se dá devido a uma possível solução por meio da criação de uma aplicação *Web*, para a automatização do plano de ensino. O desenvolvimento de uma aplicação para este tema, possibilitara a coordenação e os docentes dos cursos de Bacharelados em computação na instituição de ensino UniEVANGÉLICA gerenciar os Planos de Ensino.

Considerando o exposto, como justificativas pela a automatização do Plano de Ensino é o armazenamento dos documentos produzidos em uma plataforma digital, um melhor controle dos planos produzidos pelos docentes, redução de todo o processo no desenvolvimento do plano de ensino desde sua criação até o processo de correção e avaliação final, e por fim evitar erros como os descritos acima neste texto que podem acarretar problemas não só a universidade, mas também aos discentes que estão ou já passaram por ela.

2 Fundamentação Teórica

Neste capitulo trata-se da apresentação de toda a fundamentação teórica utilizada para a realização do projeto, desde técnicas, teóricas e ferramentas.

2.1 Plano de Ensino

De acordo com Anastasiou e Alves (2010, p. 64), “durante muito tempo, organizamos nossas ações anuais ou semestrais a partir dos planos de ensino, que tinham como centro do pensar docente o ato de ensinar; portanto, a ação docente era o foco do plano de ensino”.

O plano de ensino não é apenas um processo onde é definido o que vai ser ensinado em sala de aula, ele ressalta a importância da construção de um processo de parceria em sala de aula com o aluno, deslocando o foco do docente e do ensino para a aprendizagem.

Segundo a reitoria da IFPR (2014), “o plano de ensino também é estratégico, reflexivo, crítico e dinâmico, devendo, no decorrer de seu percurso de aplicação, ser revisado, questionado e aprimorado”.

E atualmente, as propostas didáticas estão deslocando o foco de ação docente e do ensino para a aprendizagem e destinando ao aluno o espaço do aprendiz, e assim ressaltando a importância da construção de um processo de parceria em sala de aula. Exigindo esforços conjuntos com o professor para o domínio do conhecimento (ANASTASIOU; ALVES, 2010).

2.2 Engenharia de Requisitos

Segundo Pressman e Maxim (2016, p. 14), “a engenharia de software abrange um processo, um conjunto de métodos (práticas) e um leque de ferramentas que possibilitam aos profissionais desenvolverem software de altíssima qualidade”.

Já o Sommerville nos fornece uma outra definição (2013, p. 3):

> A engenharia de software tem por objetivo apoiar o desenvolvimento profissional de software, mais do que a programação individual. Ela inclui técnicas que apoiam especificação, projeto e evolução de programas, que normalmente não são relevantes para o desenvolvimento de software pessoal.

A engenharia de software está presente em todo o desenvolvimento de um software, desde o levantamento de requisitos e até as etapas finais de um software. E ela nos capacita para o desenvolvimento de sistemas complexos dentro do prazo aplicando processo adaptável e ágil que conduza a um resultado de alta qualidade (PRESSMAN; MAXIM, 2016).

2.2.1 Levantamento de Requisitos

De acordo com Pressman e Maxim (2016, p. 131), “entender os requisitos de um problema está entre as tarefas mais difíceis enfrentadas por um engenheiro de software”. E estes

requisitos refletem as necessidades dos clientes para um sistema que serve a uma determinada finalidade, como controlar um dispositivo, colocar um pedido ou encontrar informações. (SOMMERVILLE, 2011).

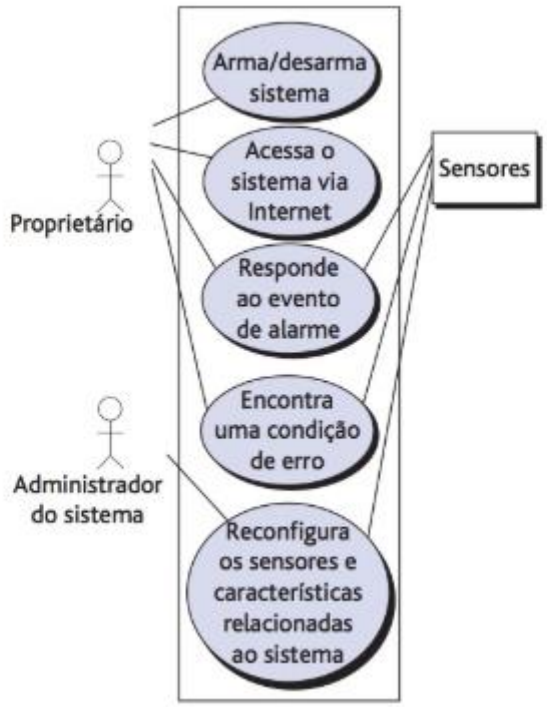
Segundo Pressman e Maxim (2016, p.131), a engenharia de requisitos é dividida em três etapas:

> A engenharia de requisitos começa com a concepção (uma tarefa que define a abrangência e a natureza do problema a ser resolvido). Ela prossegue para o levantamento (uma tarefa de investigação que ajuda os envolvidos a definir o que é necessário) e, então, para a elaboração (na qual os requisitos básicos são refinados e modificados).

Algumas técnicas de levantamento de requisitos que podem ser destacadas Aplicação da qualidade por QFD (Quality Function Deployment), Entrevista. O QFD usa observação e entrevistas com clientes, pesquisas e exame de dados históricos como evidências para a atividade de levantamento de requisitos, esses dados são então transformados em uma tabela de requisitos, que é revisada com o cliente e outros envolvidos, após feita a revisão são utilizados métodos de avaliação, diagramas e matrizes para extrair os requisitos (PRESSMAN; MAXIM, 2016). As entrevistas fazem parte da maioria dos processos de engenharia de requisitos e segundo Sommerville (2013, p. 72) “são boas para obter uma compreensão global sobre o que os stakeholders fazem, como eles podem interagir com o novo sistema e as dificuldades que eles enfrentam com os sistemas atuais”.

Uma outra técnica para o levantamento de requisitos é o Diagrama de caso de uso (Figura 1). De acordo com Pressman e Maxim (2016, p. 149) “um caso de uso conta uma jornada estilizada sobre como um usuário (desempenhando um de uma série de papéis possíveis) interage com o sistema sob um conjunto de circunstâncias específicas”. Sommerville (2013, p. 75) enfatiza “Casos de uso são técnicas eficazes para elicitar requisitos dos stakeholders que vão interagir diretamente com o sistema. Cada tipo de interação pode ser representado como um caso de uso”.

Figura 1 – Diagrama de Caso de Uso



Fonte: (Pressman e Maxim, 2016)

2.3 Metodologias Ágeis

Segundo Sbrocco (2012, p. 17) “as metodologias ágeis visam, entre outras coisas, proporcionar que o cliente tire proveito da aplicação o quanto antes, fazendo com que receba constantemente partes do software na medida em que são concluídas”

Existem diversas metodologias ágeis, cada qual com objetivos e características diferentes, no entanto todas devem seguir os princípios e valores do “Manifesto Ágil”, documento criado em 2001 em uma reunião de profissionais de software que já trabalhavam com os chamados métodos leves utilizados na época na qual foi assinado por todos os 17 (dezessete) presentes, o “Manifesto Ágil” tem um conjunto de princípios que definem critérios para os processos de desenvolvimento ágil de sistemas (SBROCCO, 2012).

2.3.1 SCRUM

Segundo Schwaber e Sutherland (2014, p. 13) define *Scrum* como sendo, “um framework dentro do qual pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível”.

O Scrum tem três pilares que apoiam a implementação de controle de processo empírico: transparência, inspeção e adaptação. No qual a transparência requer que os aspectos significativos do processo devem estar visíveis aos responsáveis pelos resultados, a inspeção onde os usuários Scrum devem, frequentemente, inspecionar os artefatos Scrum e o progresso

em direção a detectar variações e a adaptação no qual se um inspetor determina que um ou mais aspectos de um processo desviou para fora dos limites aceitáveis, e que o produto resultado será inaceitável, o processo ou o material sendo produzido deve ser ajustado (SCHWABER; SUTHERLAND, 2014).

O Time Scrum é composto pelo *Product Owner*, o Time de Desenvolvimento e o *Scrum Master*. O *Product Owner* é o responsável por maximizar o valor do produto e do trabalho do Time de Desenvolvimento, o Time de Desenvolvimento consiste de profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto "Pronto" ao final de cada Sprint, e o *Scrum Master* é responsável por garantir que o *Scrum* seja entendido e aplicado (SCHWABER; SUTHERLAND, 2014).

O Scrum é composto por vários pequenos ciclos de atividades denominado Sprint. "O coração do Scrum é a Sprint, um *time-boxed* de um mês ou menos, durante o qual um 'Pronto', incremento de produto potencialmente liberável é criado" (SCHWABER; SUTHERLAND, 2014, p. 8).

Figura 2 - Ciclo de Desenvolvimento

Fonte: (SBROCCO;MACEDO, 2012).

Conforme a Figura 2 ilustra, no início de cada projeto, clientes e desenvolvedores se reúnem com o objetivo de definir o *Backlog* do produto, depois que o *Product Backlog* é definido, a equipe deve se dedicar à definição da *Sprint Backlog*, que contém uma lista de atividades que serão realizadas na próxima sprint, após os desenvolvedores discutirem quais padrões serão adotados, as atividades de análise, codificação e testes devem se iniciar, ao final de cada *sprint* um incremento do produto deve ser apresentado ao cliente para que o time obtenha uma retroalimentação (SBROCCO; MACEDO, 2012).

2.3.2 Feature Driven Development (FDD)

Segundo Sbrocco e Macedo (2012, p. 99) “a metodologia ágil Feature Driven Development, ou FDD, como é comumente chamada, foi criada em Singapura nos anos 1990, mais especificamente em meados de 1997/1998”.

Sua equipe de projeto é constituída pelo gerente de projeto, arquiteto-chefe, equipe de modelagem, programador chefe e equipe de funcionalidades. No qual o gerente de projeto tem contato direto com o cliente (*Stakeholders*) do projeto e capta todos os requisitos, bem como as possíveis restrições, o arquiteto-chefe é o especialista no assunto do projeto a ser desenvolvido, para sanar dúvidas sobre regras de negócio, a equipe de modelagem elabora a lista de funcionalidades (*features*) do sistema, o programador chefe deve, entre outras atribuições refinar a lista de *features* e transformá-la em modelo de objetos e organizar o projeto escolhendo a linguagem e as formas de armazenamento, e a equipe de funcionalidades são so desenvolvedores, profissionais que tem o conhecimento para o desenvolvimento do projeto, além de saber como funciona o sistema (SBROCCO; MACEDO, 2012).

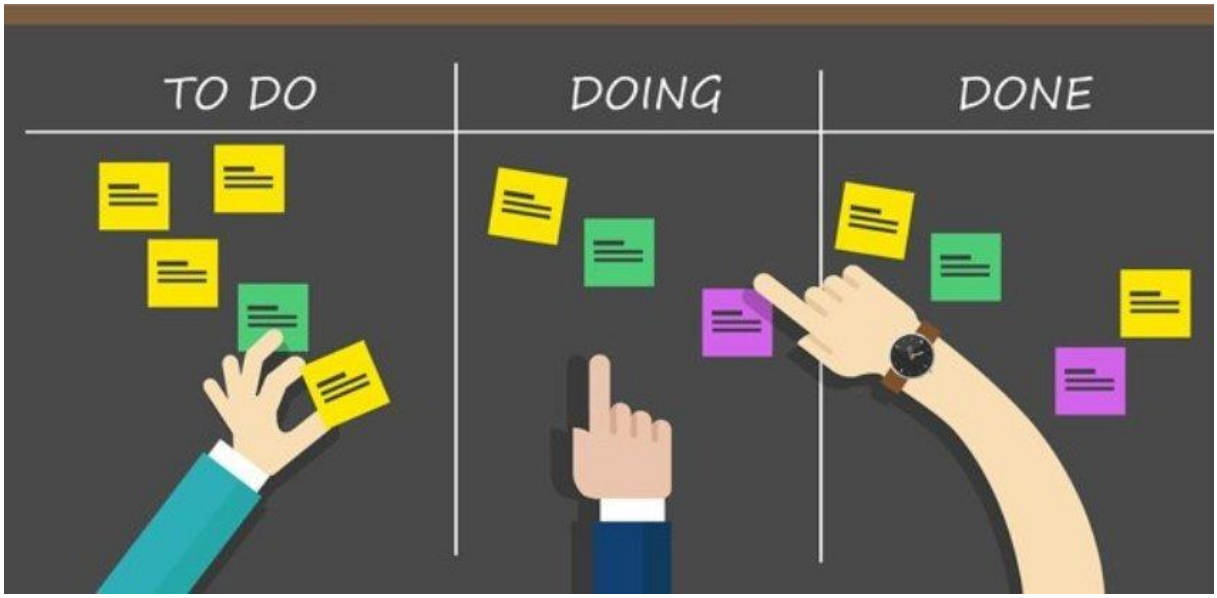
2.3.3 Kanban

Segundo Egestor (2017) o Kanban é um sistema que visa aumentar a eficiência da produção e otimizar seus sistemas de movimentação, produção, realização de tarefas e conclusão de demandas. Também conhecido como método de gestão visual.

De acordo com Anderson (2008) o Kanban foi primeiramente utilizado para manutenção de software, por conta da sua filosofia em aperfeiçoamento contínuo. Alcançado por meio de mudanças incrementais e evolucionárias, que respeitam o processo atual, os seus papéis, suas responsabilidades e seus cargos.

Esta metodologia baseia-se no conceito que é necessário conhecer o seu fluxo de trabalho (workflow) tendo cada item em produção separado. Além de limitar o número de itens em produção (Work in Progress (WIP) em cada estágio do projeto. Considerando viável o início de um trabalho somente quando o predecessor é entregue ou removido da produção (KNIBERG; SKARIN, 2010).

Figura 3 - Modelo de Kanban



Fonte: (BRASILEIRO, 2018)

2.4 Controle de Versão (Git)

Conforme especificado na documentação do Git (2018), Controle de versão é uma das atividades essenciais para o bom desenvolvimento e gerenciamento de software, no qual o sistema registra as mudanças realizadas em um arquivo ou um conjunto de arquivos ao longo do tempo, de forma que seja possível você visualizar e voltar para suas versões anteriores caso seja necessário.

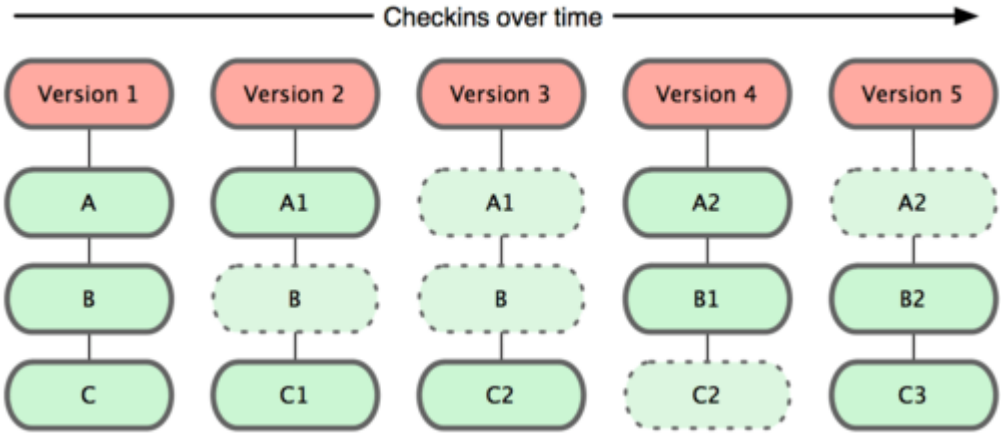
Dentro do desenvolvimento de software são criados muitos arquivos de vários formatos e tamanhos diferentes, documentos de requisitos, protótipos de sistema, arquivos fontes, script de teste e seus procedimentos. Resumidamente, todos os membros da equipe devem versionar seus documentos em um sistema de controle de versão (HUMBLE e FARLEY, 2014, p. 34).

Os sistemas de controle de versão mais conhecidos no universo do desenvolvimento de software são: Git, SVN e Mercurial. O mais utilizado é o Git, que é um sistema *open source* desenvolvido pelo Linus Torvalds, ele criou esse sistema de controle de versão pois no desenvolvimento do Kernel Linux era utilizado o *BitKeeper*, no qual não atendia todas as necessidades do Linus, assim ele criou seu próprio sistema de controle de versão no ano de 2005.

O Git considera que os dados são como um conjunto de *snapshots* de um sistema de arquivo, toda vez que é realizado um *commit* no estado do seu projeto no Git, é como se tirasse uma foto de todos os arquivos de seu projeto naquele momento e armazenasse uma referência para essa captura, e se nenhum arquivo for alterado, ele não e salvo novamente, a informação

não é armazenada novamente - apenas um link para o arquivo idêntico anterior que já foi armazenado. A figura abaixo demonstra melhor como o Git lida com os arquivos (Git, 2018).

Figura 4 - Diagrama de versionamento de dados com Git



Fonte: Documentação Git

2.5 Arquitetura de Software

Arquitetura de software refere-se ao modo que você organiza componentes de programa (módulos), a maneira que esses componentes interagem um com o outro e a estrutura de dados que são utilizadas pelos componentes. Segundo Pressman e Maxim (2016, p.232), arquitetura de software é “à organização geral do software e aos modos pelos quais ela disponibiliza integridade conceitual para um sistema”.

2.5.1 Padrões arquiteturais

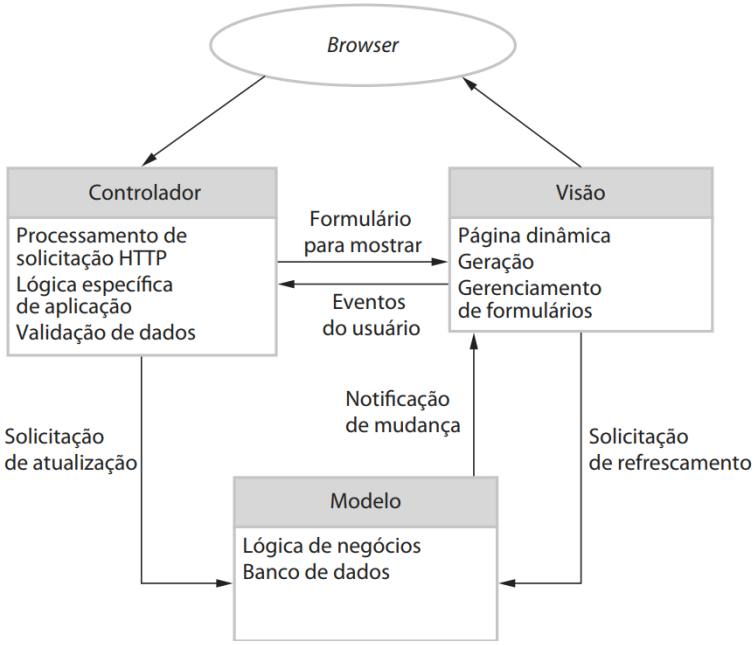
Padrão de arquitetura pode ser considerado uma descrição abstrata, estilizada de boas práticas que anteriormente já foram experimentadas e testadas em diversos tipos de sistemas e ambientes. Existem vários padrões arquiteturais, e cada um dele procura atender as necessidades de determinados projetos. Os padrões mais conhecidos são: Arquitetura em camadas, Cliente-servidor, MVC, MVP, etc. (SOMMERVILLE, 2013, p. 108).

2.5.1.1 MVC

O MVC (*Model View Controller*) e um dos padrões arquiteturais mais utilizado para o desenvolvimento de software hoje. Sommerville (2013) descreve o MVC como o modelo que separa a apresentação e a interação dos dados do sistema, estruturando-o em três componentes lógicos que comunicam entre si, *model* (Modelo), *view* (visão) e *controller* (controlador).

- *Model* gerencia o sistema de dados e as operações associadas a esses dados.
- *View* define e gerencia como os dados são apresentados ao usuário.
- *Controller* gerencia a interação do usuário (por exemplo, teclas, cliques do mouse etc.) e passa essas interações para a Visão e o Modelo.

Figura 5 - Arquitetura de aplicações Web usando o padrão MVC



Fonte: (SOMMERVILLE, 2013, p. 110)

O MVC é usado quando existem várias maneiras de se visualizar e interagir com dados. Pois ela permite que os dados sejam alterados de forma independente de suas representações, e vice-versa. E também ela apoia a apresentação dos mesmos dados de maneiras diferentes, com as alterações feitas em uma representação aparecendo em todas elas. O MVC não e recomendado quando o projeto e de baixa complexidade e de interações simples, podendo aumentar a complexidade desnecessariamente para resolver o problema (SOMMERVILLE, 2013, p. 109).

2.6 Tecnologias

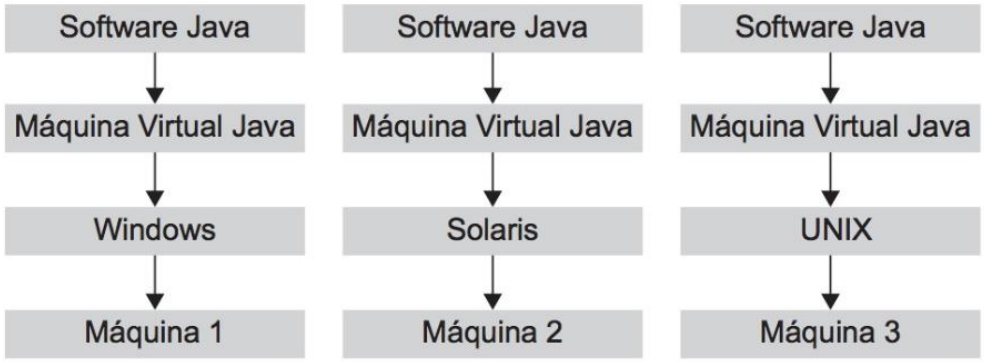
2.6.1 Java

Java e uma linguagem de programação desenvolvida na década de 90 pela empresa Sun, inicialmente a linguagem foi concebida para utilização em pequenos dispositivos eletrônicos, como: televisão, vídeo cassete, aparelhos de TV a cabo, etc., entretanto não foi bem-sucedido a introdução do Java nesse setor de dispositivos eletrônicos. A partir de 1995 a Sun decidiu anunciar o Java não apenas como uma linguagem de programação, mas como uma nova plataforma de desenvolvimento (FURGERI, 2015, p. 13).

A linguagem Java tem uma grande aceitação no mercado devido ao fato de que programas desenvolvido utilizando a linguagem pode ser executado virtualmente em muitas plataformas, aceitos em muitos tipos de equipamentos e em vários sistemas operacionais, o

mesmo programa pode ser executado em uma com Windows, MacOs ou Linux. Além de tudo com o Java o processamento pode deixar de ser realizado apenas do lado do servidor.

Figura 6 - Um mesmo programa Java executando em plataformas diferentes.



Fonte: (FURGERI, 2015, p. 17)

2.6.2 Spring Boot

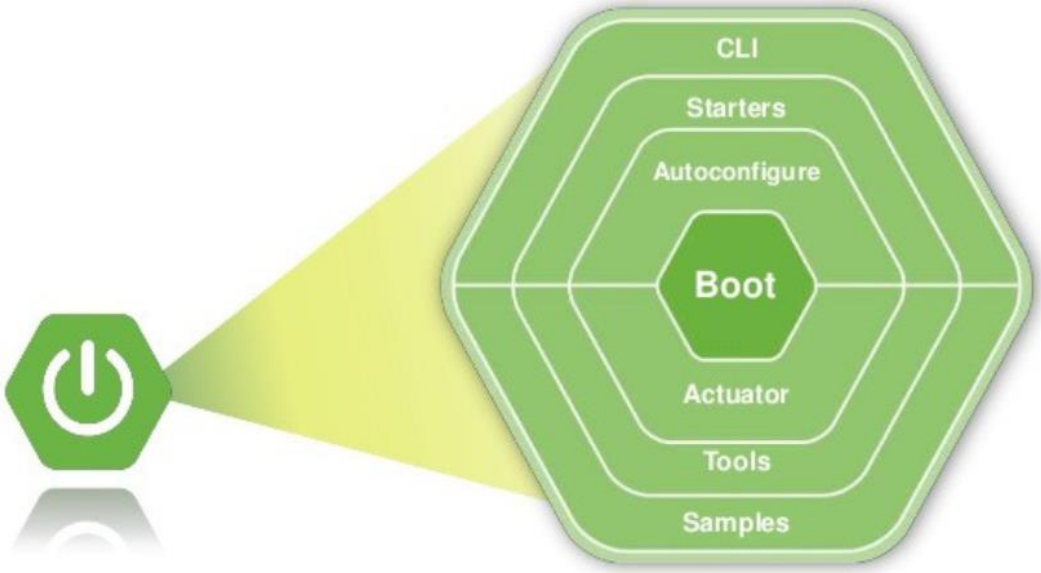
O *Spring Boot* é uma estrutura leve e que simplifica a criação e configuração de aplicações baseadas no *Spring Framework.* Segundo Boaglio (2017, p. 6), o *Spring Boot* teve sua primeira versão lançada no ano de 2014 após 18 meses de desenvolvimento e milhares de *commits*, a sua origem veio do *Spring framework.*

O *Spring Boot* veio para solucionar a complexidades que havia na inicialização de uma aplicação e também o gerenciamento das dependências de um projeto *Spring*, tratando de maneira coesa e eficiente a questão de configuração, no qual ele faz um uso extensivo de *Convention Over Configuration* (OKI, 2015).

O logotipo do *Spring Boot* foi criado do ícone de botão de inicializar a maioria das máquinas (*boot*), cuja ideia é iniciar a aplicação. A arquitetura do *Spring Boot* e formada por vários componentes: (BOAGLIO, 2017, p. 10-11)

- **CLI -** o *Spring Boot* CLI é uma ferramenta de linha de comando que facilita a criação de protótipos através de scripts em *Groovy*;
- ***Starters*** — é um conjunto de componentes de dependências que podem ser adicionados aos nossos sistemas;
- **Autoconfigure** — configura automaticamente os componentes carregados;
- ***Actuator*** — ajuda a monitorar e gerenciar as aplicações publicadas em produção;
- ***Tools*** — é uma IDE customizada para o desenvolvimento com *Spring Boot*;
- ***Samples*** — dezenas de implementações de exemplos disponíveis para uso.

Figura 7 - Arquitetura do Spring Boot



Fonte: (BOAGLIO, 2017, p. 11)

2.6.3 Angular

O Angular é um *framework* JavaScript criado e mantido pela equipe de desenvolvedores da Google, ele é utilizado para criar e executar aplicações SPA (*Single-Page Applications* ou Aplicação de uma Página). O desenvolvimento de uma aplicação utilizando o framework não é feita com JavaScript, mas sim em TypeScript. TypeScript é um *superset* (superconjunto) de JavaScript que é incorporado algumas bibliotecas e funcionalidades, em sua execução ele compilado para o JavaScript simples que pode ser executado nos browsers.

2.7 Banco de Dados

A definição de banco de dados segundo Alves (2014), “banco de dados é um conjunto de dados com significado implícito”. Existe uma grande diferença entre informação e dado, informação é qualquer fato ou conhecimento do mundo real e que pode ou não ser armazenado. Dado é a representação da informação, que pode estar registrado em papel, num quadro de aviso ou no disco rígido do computador. Para entender melhor a diferença entre os dois, veja a seguir a imagem.

Figura 8 - Diferença entre informação e dado

Informação	Dado
Está muito quente hoje	A temperatura hoje é de 38 graus Celsius

Fonte: (ALVES, 2014, p. 16)

Um exemplo de bando de dados de uma universidade poderia ter dados como: Entidades como alunos, professores, cursos e turmas. E também poderia haver relacionamentos entre estas entidades, como a matricula dos alunos em determinados cursos, quais professores ministram determinados cursos, e quais salas serão ministrados os cursos (RAMAKRISHNAN e GEHRKE, 2011, p. 3).

2.7.1 MongoDB

O MongoDB é um banco de dados NoSQL orientado a documentos ou *document database* (banco de dados de documentos), mas não são documentos comuns que se encontra em qualquer pasta dos computadores, são documentos em formato JSON. Ao contrário do banco de dados relacional onde as entidades são representadas em tabelas e interligadas por meio de chave primaria e estrangeira, no MongoDB o documento representa toda informação necessária, sem as restrições dos bancos de dados relacionais (BOAGLIO, 2015, p. 11).

Figura 9 – Exemplo de dados contidos em um documento do MongoDB

Fonte: Os autores.

No MongoDB os dados são organizados em função da aplicação, e não a aplicação organizada em função dos dados. Para melhora entender, no modelo relacional a aplicação obedece às regas do seu banco de dados, já no MongoDB é o contrário, é os dados que obedecem às ordens da aplicação. Os dados são organizados conforme a necessidade do sistema. A organização dos dados se dá por meio de *collections*, que representa as tradicionais tabelas dos bancos de dados relacionais. Dentro de uma *field* que é o mesmo que uma coluna no modelo relacional, você pode ter um *array*, uma lista de valores, algo impossível em uma tabela convencional (BOAGLIO, 2015, p. 1-3).

3 Desenvolvimento

Este capitulo explicitará todos os passos percorridos durante todo o processo de desenvolvimento deste projeto, tanto nas elicitações dos requisitos, quanto ao desenvolvimento e metodologias utilizadas para as tais tarefas.

3.1 Processo Metodológico

No princípio do desenvolvimento do projeto foi feita a análise dos requisitos juntamente com os *stakeholders*, assim especificando as características operacionais do software, um detalhamento da interface e outros elementos e também estabelecendo as restrições que o sistema irá conter. (PRESSMAN e MAXIM, 2016, p. 167).

Para a realização das atividades foram necessários identificar com os *stakeholders*, por meio de entrevistas informais e *brainstorming*, as funcionalidades e os cenários do sistema. Essas funcionalidades e cenários foram mapeadas utilizando modelo baseado em cenários no formato de História de Usuário.

3.1.1 Iterativo incremental

O modelo incremental auxiliara a equipe a entregar valor continuamente ao cliente, com a implementação e entrega de novas funcionalidades do sistema sempre que finalizados, assim podendo colher *feedbacks* do usuário final para a melhor validação do sistema.

3.1.2 Política de versionamento

O versionamento é uma das atividades excepcionais no desenvolvimento de qualquer que seja o produto de *software*, pois auxilia no controle dos códigos gerados, quanto o seu versionamento podendo ser preciso restaurar para alguma versão anterior com a finalidade de "corrigir" algum erro criado através de uma nova edição do código.

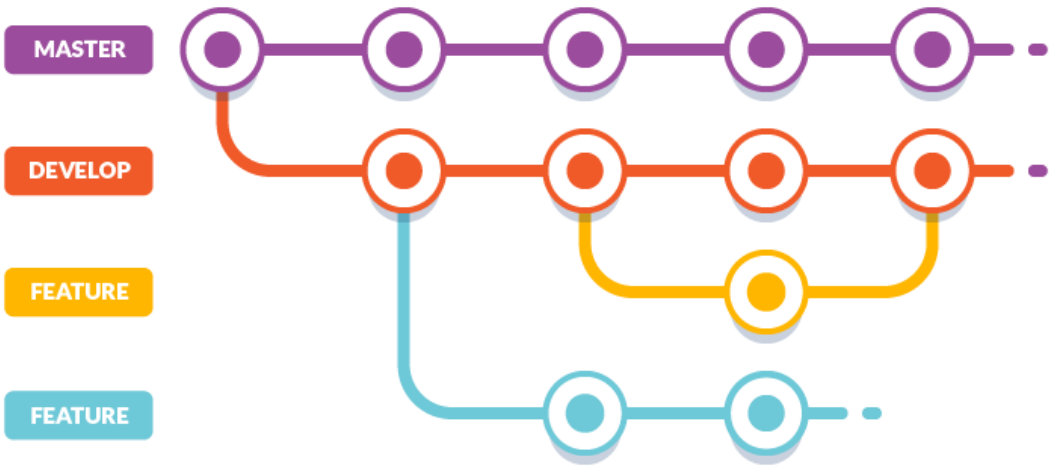
Quanto ao versionamento do sistema Eduplan será empregado o uso do modelo de versionamento denominado de *GitFlow*, que é uma metodologia que explica a maneira dinâmica e ativa de organização das *branchs* do Git. (WEED, 2014)

O modelo (figura 10) adotado para a organização do Eduplan é a seguinte: originalmente tem duas *branchs*, a *master* que é onde a versão que é entregue ao cliente em todo incremento, nela contém uma versão mais atual do sistema em funcionamento, a *development* é uma "cópia" da *master*, e tem como função servir de base para a criação das demais *branchs* quando necessário, e serão criadas novas *branchs* a partir da *development* quando for necessário realizar alguma mudança ou criação de uma nova funcionalidade do sistema.

Conforme for finalizado a atividade proposta com a *branch* criada, faz um merge da *development* com a determinada *branch*, assim incluindo todas as alterações feitas no sistema

para a *development*, logo após a realização dos testes necessários para verificar se o sistema está de acordo com o planejado e documentado, e feito um *merge* da *branch development* com a *master* para que seja incluído todas as alterações na *branch master*.

Figura 10 - Fluxo de organização do Git



Fonte: Desconhecido

3.1.3 Planejamento da *Sprint*

Foi definido pela equipe de desenvolvimento do sistema Eduplan que as histórias de usuário seriam divididas em três atividades, sendo elas: Desenvolvimento do *back-end*, onde seja implementado as regras de negócio do sistema conforme especificado na documentação, o desenvolvimento do *front-end* responsivo e de o fácil entendimento e de boa usabilidade para os clientes finais, e por fim, a atualização de documento conforme alguma mudança necessária no sistema a pedido do cliente ou para melhorar alguma descrição previamente feita no documento.

O ciclo de cada *sprint* foi definido durante o planejamento inicial, definindo um período de duas semanas para cada *sprint*, sendo esse tempo usado para o desenvolvimento do(s) requisito(s) elencado a ser desenvolvido na determinada *sprint*, podendo haver correções do mesmo se necessário em uma próxima *sprint*.

3.1.4 *Sprint* 1 – Levantamento de Requisitos

A primeira Sprint do projeto teve como objetivo principal o desenvolvimento dos artefatos do projeto (documento de visão, histórias de usuário, diagrama de caso de uso), procurando modelar por completo o sistema para que as atividades futuras sejam feitas sem que seja necessário remodelar o sistema.

A princípio foram realizadas as primeiras entrevistas com os *stakeholders* nas quais foram expostos seus interesses por meio da técnica de levantamento de requisitos *brainstorming*. Como resultado foi construído o Documento de Visão (APÊNDICE A), no qual está mapeado o problema que o projeto irá resolver e uma visão macro dos seus requisitos e a estrutura do projeto.

Com base no Documento de Visão, foi modelado o Diagrama de Caso de Uso (APÊNDICE B), que demonstra os atores que interagem com o sistema e suas responsabilidades, dessa forma foi possível validar os requisitos com os stakeholders quanto ao seu entendimento.

Com base no Diagrama de Caso de Uso (APÊNDICE B) foi escrito as Histórias de Usuário, na qual foram mapeadas as funcionalidades do sistema, desde o *login* do usuário até a criação a geração do PDF com o plano de ensino.

3.1.5 Sprint 2 - Definição da arquitetura do Sistema

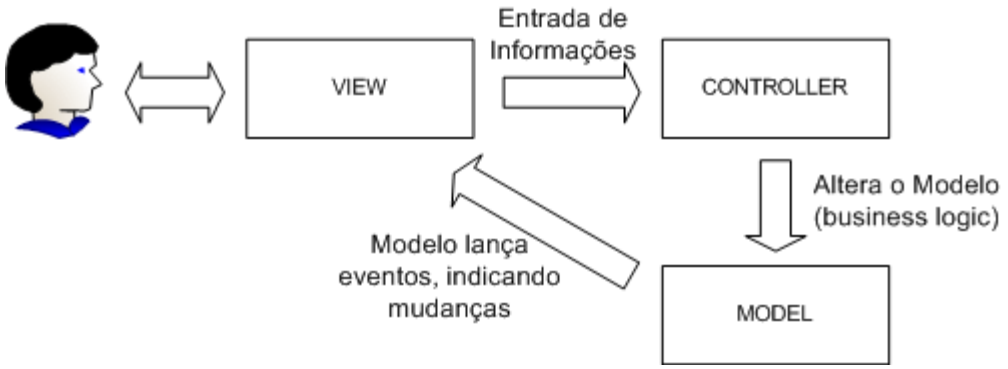
Para a definição da arquitetura Eduplan foram levados em consideração pontos importantes para o desenvolvimento de software. São eles:

- Uso de tecnologias com grande adoção no mercado e que tenha uma grande comunidade que possa auxiliar na resolução de possíveis problemas durante o desenvolvimento.
- Agilidade no desenvolvimento e alto nível de reaproveitamento;
- Iniciar o projeto com baixo custo e possibilidade de escalonamento de acordo com que o produto evolui.

A tecnologia adotada no desenvolvimento do sistema Eduplan foi o MVC, que é uma arquitetura muito utilizada, devido a sua facilidade de compreensão e entendimento dos componentes do *software*, quanto a sua funcionalidade que atende a maioria dos softwares básicos desenvolvidos atualmente.

A figura abaixo demonstra a arquitetura MVC (*Model View Controller*) também adotada no *Back-end* utilizando a linguagem de programação Java e o *framework Spring Boot*:

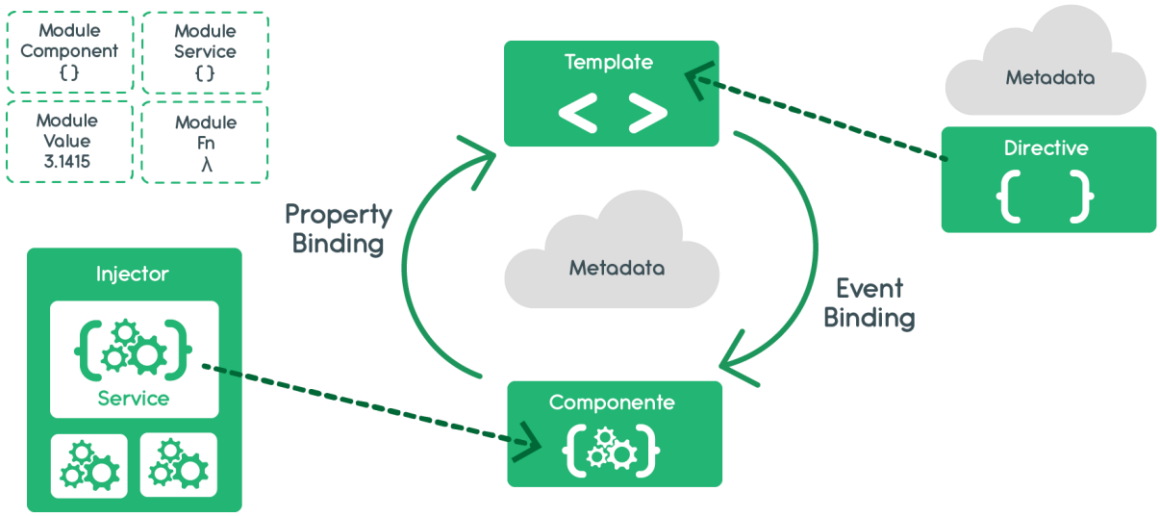
Figura 11 - Arquitetura do back-end



- *model* tem como função realizar toda a comunicação entre a aplicação e o banco de dados, seja para inserção de novos dados, busca, alteração ou deleção de dados contidos no banco.
- *view* é a interface que será apresentado ao usuário, que no projeto Eduplan é o *front-end* criado com o framework Angular.
- *controller* tem como função principal ser o intermediário da comunicação entre a *view* e o *model*, no qual terá as regras de negócios, as rotas para a realização da requisição necessária.

Veja na figura abaixo uma breve imagem representativa do funcionamento interno da arquitetura do *Front-end* utilizando o *framework JavaScript Angular*:

Figura 12 - Arquitetura do front-end



Fonte: Documentação Angular (GOOGLE, 2019)

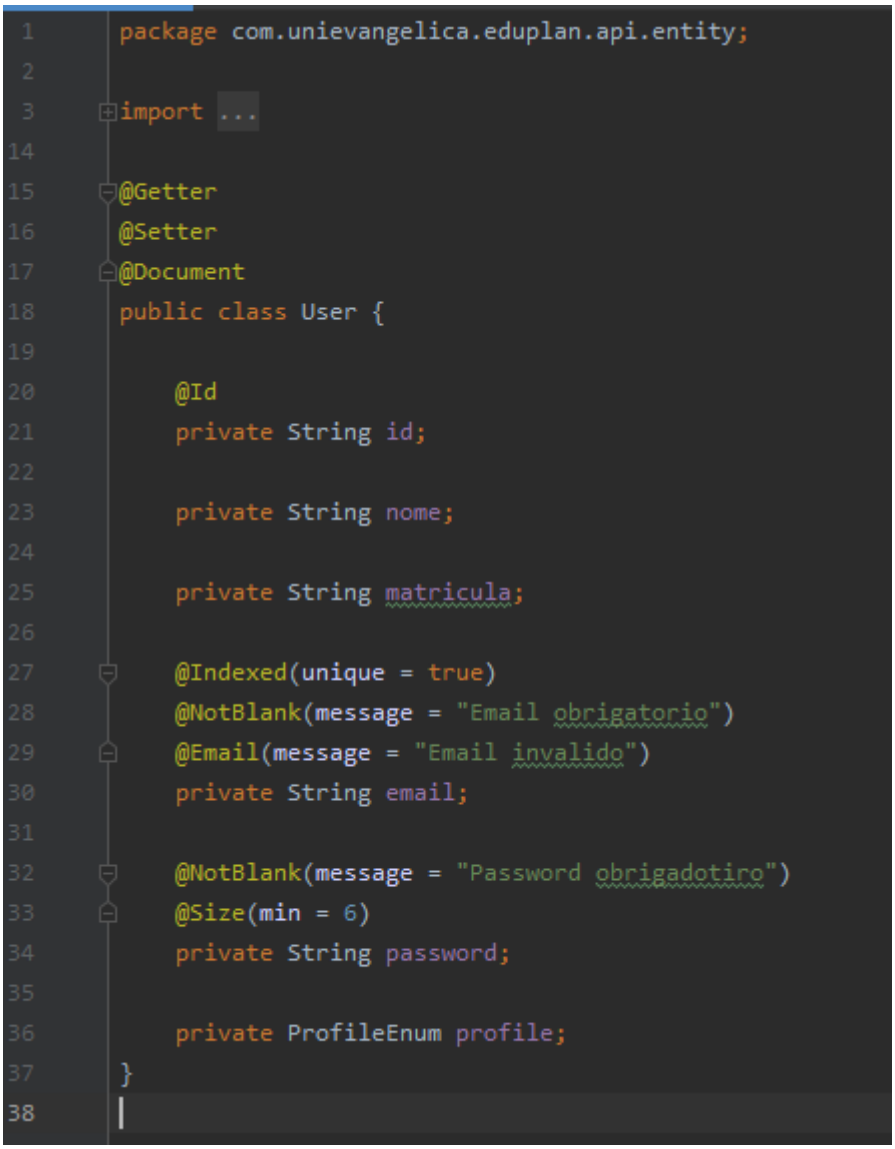
3.1.5.1 Padrão de desenvolvimento

Logo após a configuração das arquiteturas do projeto foi possível iniciar o desenvolvimento do sistema, onde foram dividias em três tarefas, o *back-end* e *front-end* e documentação. A seguir será explicado o processo de desenvolvimento tanto do *back* (API) quanto do *front*, será exemplificado como padrão o requisito Cadastrar e Manter Usuários.

3.1.5.2 Padrão de desenvolvimento Back-end

Ao iniciar o desenvolvimento do requisito, foi criado a entidade. A imagem abaixo mostra a entidade criada:

Figura 13 - Entidade User

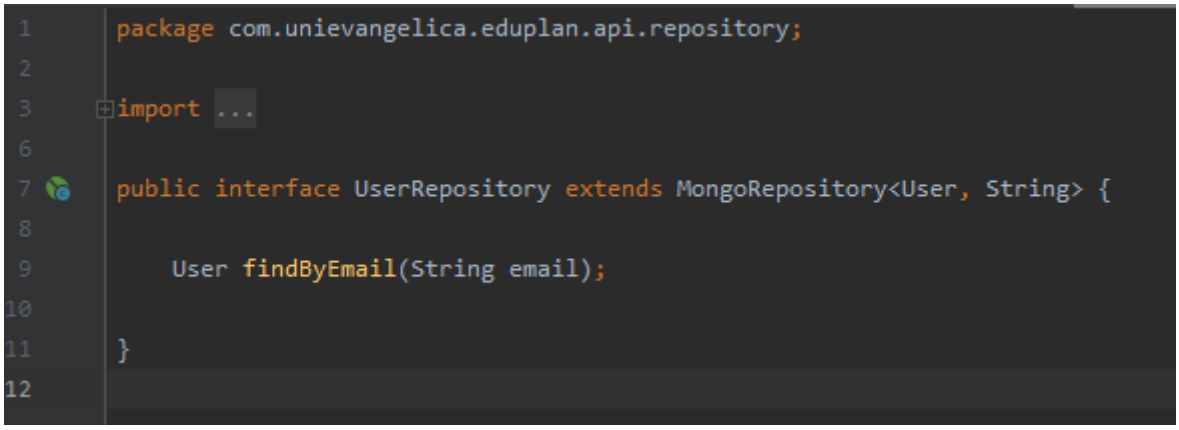


```
1   package com.unievangelica.eduplan.api.entity;
2
3   import ...
14
15  @Getter
16  @Setter
17  @Document
18  public class User {
19
20      @Id
21      private String id;
22
23      private String nome;
24
25      private String matricula;
26
27      @Indexed(unique = true)
28      @NotBlank(message = "Email obrigatorio")
29      @Email(message = "Email invalido")
30      private String email;
31
32      @NotBlank(message = "Password obrigadotiro")
33      @Size(min = 6)
34      private String password;
35
36      private ProfileEnum profile;
37  }
38
```

Fonte: Os Autores

Na entidade está definido os nomes dos atributos que serão posteriormente utilizados para a modelagem dos dados a serem salvos no banco de dados. Em seguida é realizado a criação do repositório do requisito com a extensão para o *MongoRepository*, para que seja utilizado as funções padrões da própria biblioteca do mongo, não sendo necessária criar tudo novamente.

Figura 14 - Contoller do User

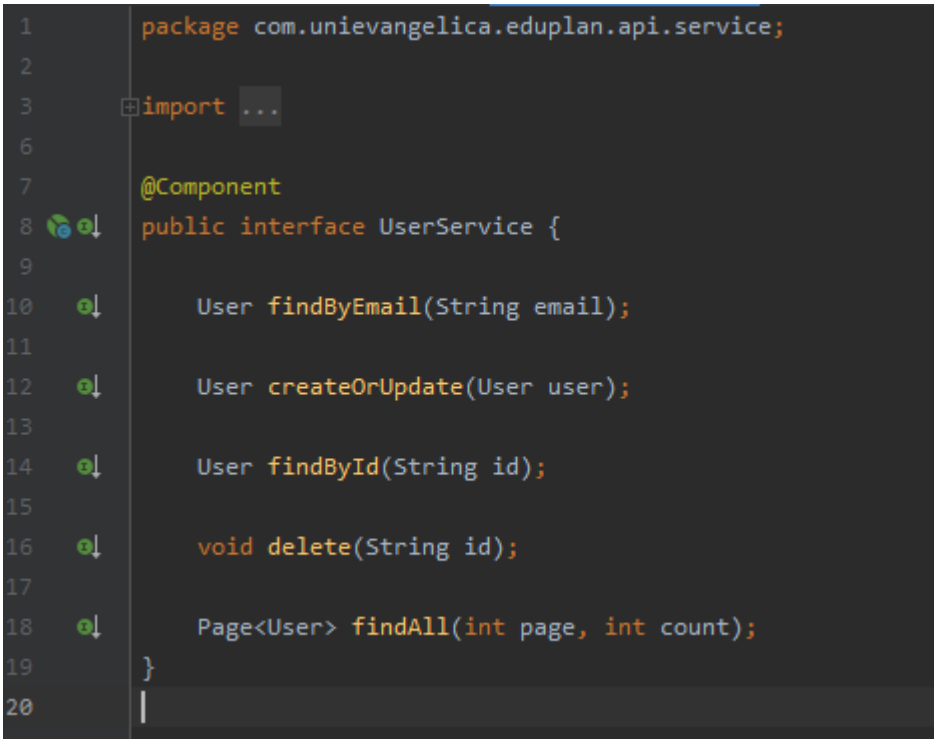


```
package com.unievangelica.eduplan.api.repository;

import ...

public interface UserRepository extends MongoRepository<User, String> {

    User findByEmail(String email);

}

```

Fonte: Os Autores

Após a criação do *repository* é criado o *service* do requisito, sendo criado um serviço como “modelo” e outro para a real implementação do serviço.

Figura 15 - Service "modelo" do User



```
package com.unievangelica.eduplan.api.service;

import ...

@Component
public interface UserService {

    User findByEmail(String email);

    User createOrUpdate(User user);

    User findById(String id);

    void delete(String id);

    Page<User> findAll(int page, int count);
}

```

Fonte: Os Autores

Em seguida a classe de implementação do serviço do *User*.

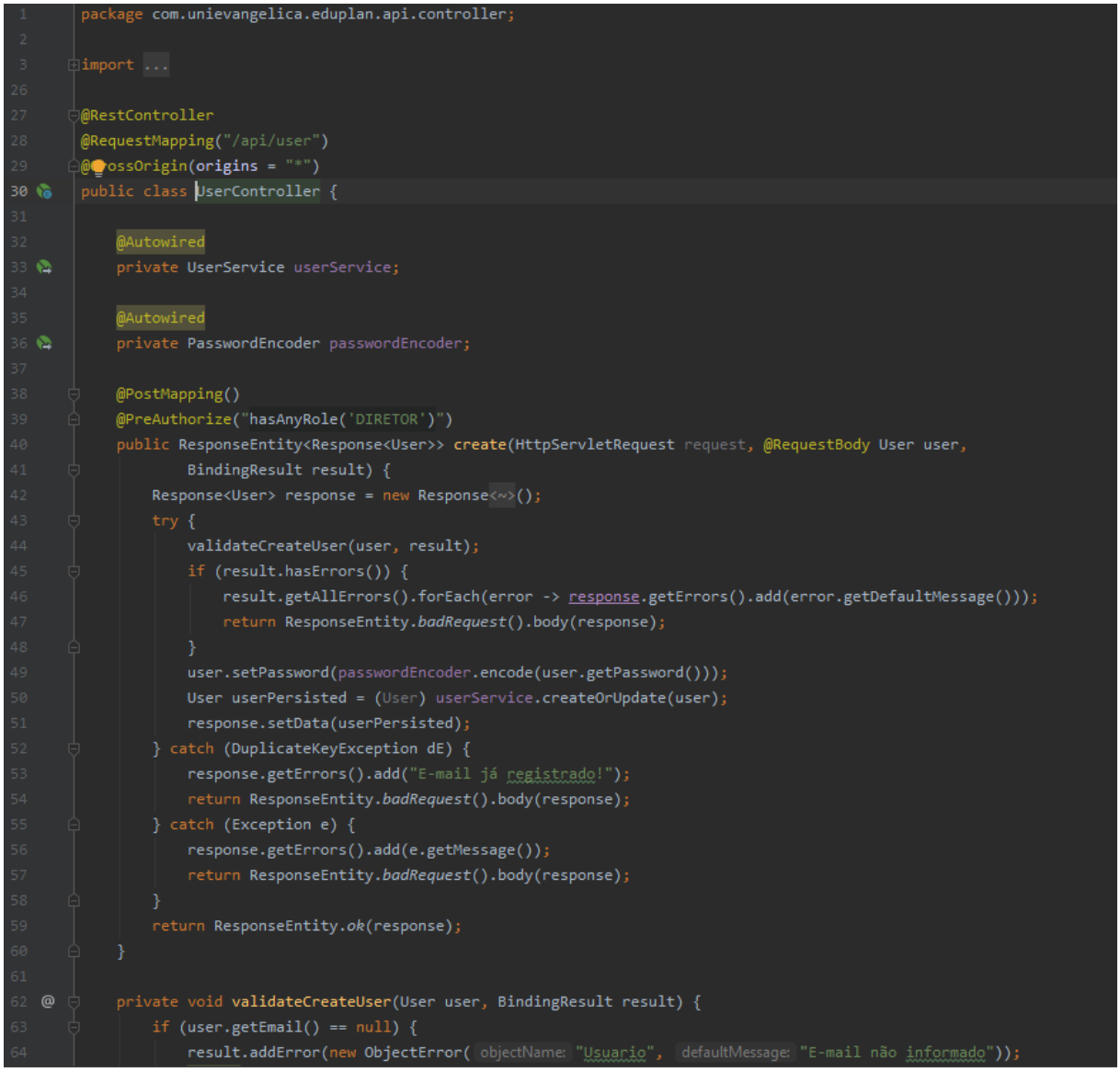
Figura 16 - Service Impl do User

```java
package com.unievangelica.eduplan.api.service.impl;

import ...

@Component
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;

    public User findByEmail(String email) {
        return this.userRepository.findByEmail(email);
    }

    public User createOrUpdate(User user) {
        return this.userRepository.save(user);
    }

    public User findById(String id) {
        return this.userRepository.findOne(id);
    }

    public void delete(String id) {
        this.userRepository.delete(id);
    }

    public Page<User> findAll(int page, int count) {
        Pageable pages = new PageRequest(page, count);
        return this.userRepository.findAll(pages);
    }
}
```

Fonte: Os Autores

Por fim, após a finalização da criação do *repository* dos *services* deu início à criação do *controller* do *User*, que por sua vez tem como função o recebimento das requisições feitas pelo *front-end* e interpretadas e retornar as respostas de acordo com cada requisição feita.

Figura 17 - Controller do User



```
package com.unievangelica.eduplan.api.controller;

import ...

@RestController
@RequestMapping("/api/user")
@CrossOrigin(origins = "*")
public class UserController {

    @Autowired
    private UserService userService;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @PostMapping()
    @PreAuthorize("hasAnyRole('DIRETOR')")
    public ResponseEntity<Response<User>> create(HttpServletRequest request, @RequestBody User user,
            BindingResult result) {
        Response<User> response = new Response<~>();
        try {
            validateCreateUser(user, result);
            if (result.hasErrors()) {
                result.getAllErrors().forEach(error -> response.getErrors().add(error.getDefaultMessage()));
                return ResponseEntity.badRequest().body(response);
            }
            user.setPassword(passwordEncoder.encode(user.getPassword()));
            User userPersisted = (User) userService.createOrUpdate(user);
            response.setData(userPersisted);
        } catch (DuplicateKeyException dE) {
            response.getErrors().add("E-mail já registrado!");
            return ResponseEntity.badRequest().body(response);
        } catch (Exception e) {
            response.getErrors().add(e.getMessage());
            return ResponseEntity.badRequest().body(response);
        }
        return ResponseEntity.ok(response);
    }

    private void validateCreateUser(User user, BindingResult result) {
        if (user.getEmail() == null) {
            result.addError(new ObjectError( objectName: "Usuario",  defaultMessage: "E-mail não informado"));
```

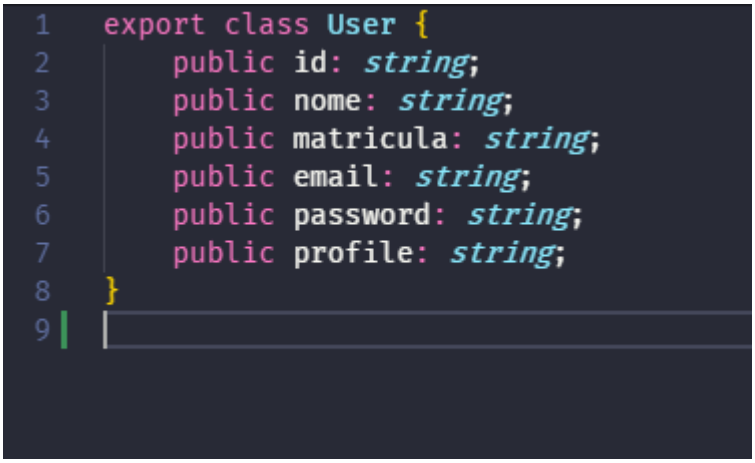
Fonte: Os Autores

Deste modo com a criação das classes citadas acima, finaliza o desenvolvimento do *back-end* para este requisito, podendo assim dar início ao desenvolvimento do *front-end* juntamente com a integração entre os dois. Em seguida será apresentado as etapas necessárias para o desenvolvimento do *front-end* que também será utilizada como referência para o desenvolvimento para os demais requisitos.

3.1.5.3 Padrão de desenvolvimento Front-end

No início do desenvolvimento do *front-end* é criado uma classe modelo, a qual contém os atributos da entidade *User*.

Figura 18 - Modelo User



```
export class User {
    public id: string;
    public nome: string;
    public matricula: string;
    public email: string;
    public password: string;
    public profile: string;
}
```

Fonte: Os Autores

Logo após e criado o serviço, na qual irá consumir os dados do *back-end*. A partir do serviço que é realizada as requisições necessárias para o requisito, a imagem abaixo demonstra o serviço em que engloba as funcionalidades do requisito.

Figura 19 - Service User

```
import { User } from './../../model/user';
import { Injectable } from '@angular/core';
import { HttpClient} from '@angular/common/http';
import { EDUPLAN_API } from '../eduplan.api';

@Injectable()
export class UserService {

  constructor(private http: HttpClient) {}

  login(user: User){
    return this.http.post(`${EDUPLAN_API}/api/auth`,user);
  }

  createOrUpdate(user: User){
    if(user.id != null && user.id != ''){
      return this.http.put(`${EDUPLAN_API}/api/user`,user);
    } else {
      user.id = null;
      return this.http.post(`${EDUPLAN_API}/api/user`, user);
    }
  }

  findAll(page:number,count:number){
    return this.http.get(`${EDUPLAN_API}/api/user/${page}/${count}`);
  }

  findById(id:string){
    return this.http.get(`${EDUPLAN_API}/api/user/${id}`);
  }

  delete(id:string){
    return this.http.delete(`${EDUPLAN_API}/api/user/${id}`);
  }
}
```

Fonte: Os Autores

Com a criação do serviço finalizado pode dar início a criação dos componentes do *User*, que será *user-new* e *user-list*, o componente *user-new* será implementado o formulário para cadastro com todos os campos necessários para que seja feito a requisição para o *back-end*. No user-*list* será feito uma requisição onde buscará todos os usuários cadastrados no sistema e os apresentará em forma de lista, pode edita-los e apagar. A imagem abaixo apresente como foi desenvolvido o componente *user-new*:

Figura 20 - Componente user-new

```
@Component({
  selector: 'app-user-new',
  templateUrl: './user-new.component.html',
  styleUrls: ['./user-new.component.css']
})
export class UserNewComponent implements OnInit {

  @ViewChild("form")
  form: NgForm;

  user = new User();
  shared: SharedService;
  message: {};
  classCss: {};
  id: string = this.route.snapshot.params['id'];
  edit: string;

  constructor(
    private userService: UserService,
    private route: ActivatedRoute,
    private router: Router
  ) {
    this.shared = SharedService.getInstance();
  }

  ngOnInit() {
    if (this.id != undefined) {
      this.findById(this.id);
      this.edit = 'Editar Usuário';
    }else {
      this.edit = 'Novo Usuário';
    }
  }

  register() {
    this.message = {};
    this.userService.createOrUpdate(this.user).subscribe((responseApi: ResponseApi) => {
      this.user = new User();
      let userRet: User = responseApi.data;
      this.form.resetForm();
      this.router.navigate(['/user-list']);
    }, err => {
      this.showMessage({
        type: 'error',
        text: err['error']['errors'][0]
      });
    });
  }

```

Fonte: Os Autores

A imagem abaixo apresentara uma parte da forma em que foi modelado no HTML o componente.

Figura 21 - HTML do Componente user-new

```
<div class="col-md-10" style="margin-left:0%">
  <div class="box box-info">
    <div class="box-header with-border">
      <h3 class="box-title">{{ edit }}</h3>
    </div>
    <form class="form-horizontal" #form="ngForm" (ngSubmit)="register()" novalidate>
      <div [ngClass]="classCss" role="alert" *ngIf="message">
        <strong>{{ message.text }}</strong>
      </div>
      <div class="box-body">

        <div [ngClass]="getFormGroupClass(nome.invalid,nome.dirty)">
          <label for="inputNome" class="col-sm-2 control-label">Nome</label>
          <div class="col-sm-10">
            <input type="text" autocomplete="off" [(ngModel)]="user.nome" name="nome" class="form-control" id="inputNome" #nome="ngModel"
              placeholder="Nome completo" nome>
            <span class="help-block" *ngIf="nome.invalid  && nome.dirty">Nome e obrigatorio </span>
          </div>
        </div>

        <div [ngClass]="getFormGroupClass(matricula.invalid,matricula.dirty)">
          <label for="inputMatricula" class="col-sm-2 control-label">Matricula</label>
          <div class="col-sm-10">
            <input type="text" autocomplete="off" [(ngModel)]="user.matricula" name="matricula" class="form-control" id="inputMatricula"
            #matricula="ngModel"
              placeholder="Matricula" matricula>
            <span class="help-block" *ngIf="matricula.invalid  && matricula.dirty">Matricula e obrigatorio </span>
          </div>
        </div>

        <div [ngClass]="getFormGroupClass(email.invalid,email.dirty)">
          <label for="inputEmail" class="col-sm-2 control-label">Email</label>
          <div class="col-sm-10">
            <input type="email" autocomplete="off" [(ngModel)]="user.email" name="email" class="form-control" id="inputEmail" #email="ngModel"
              placeholder="Email" email>
            <span class="help-block" *ngIf="email.invalid  && email.dirty">Email e obrigatorio </span>
          </div>
        </div>
        <div [ngClass]="getFormGroupClass(password.invalid,password.dirty)">
          <label for="inputPassword" class="col-sm-2 control-label" *ngIf="id == undefined; else elseBlock">Senha</label>
          <ng-template #elseBlock>
            <label for="inputPassword" class="col-sm-2 control-label">Nova Senha</label>
          </ng-template>
          <div class="col-sm-10">
            <input type="password" autocomplete="off" [(ngModel)]="user.password" name="password" class="form-control" id="inputPassword"
              placeholder="Password" #password="ngModel" required>
            <span class="help-block" *ngIf="password.invalid && password.dirty">Senha e obrigatoria</span>
          </div>
        </div>
```

Fonte: Os Autores

Para acessar cada um dos componentes existentes no sistema, é necessário a utilização de rotas para que o redirecionamento seja feito a cada alteração na rota utilizada no determinado momento, assim podendo navegar entre os componentes do sistema. O mapeamento das rotas foi realizado dentro do arquivo *app.route.ts*, veja na imagem abaixo:

Figura 22 - Rotas do Sistema

```
export const ROUTES: Routes = [
  { path: 'login' , component: LoginComponent },
  { path: '' , component:  HomeComponent, canActivate: [AuthGuard]},
  { path: 'user-new' , component: UserNewComponent, canActivate: [AuthGuard] },
  { path: 'user-new/:id' , component: UserNewComponent, canActivate: [AuthGuard] },
  { path: 'user-list' , component: UserListComponent, canActivate: [AuthGuard] },
  { path: 'plano-de-ensino-new' , component: PlanoDeEnsinoNewComponent, canActivate: [AuthGuard] },
  { path: 'plano-de-ensino-new/:id' , component: PlanoDeEnsinoNewComponent, canActivate: [AuthGuard] },
  { path: 'plano-de-ensino-list' , component: PlanoDeEnsinoListComponent, canActivate: [AuthGuard] },
  { path: 'plano-de-ensino-detail/:id' , component: PlanoDeEnsinoDetailComponent, canActivate: [AuthGuard] }
  ,
  { path: 'disciplina-new' , component: DisciplinaNewComponent, canActivate: [AuthGuard] },
  { path: 'disciplina-new/:id' , component: DisciplinaNewComponent, canActivate: [AuthGuard] },
  { path: 'disciplina-list' , component: DisciplinaListComponent, canActivate: [AuthGuard] },
  { path: 'disciplina-detail/:id' , component: DisciplinaDetailComponent, canActivate: [AuthGuard] },
]

export const routes: ModuleWithProviders = RouterModule.forRoot(ROUTES);
```

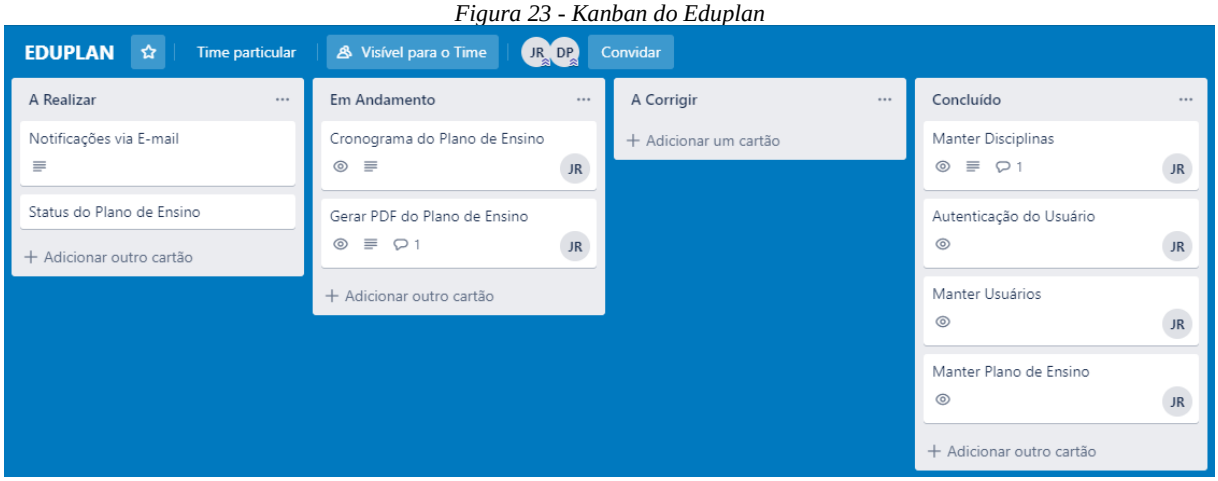
Fonte: Os Autores

3.1.5.4 Kanban (Trello)

O *Kanban* utilizado no projeto foi o *Trello*, que é bastante conhecido como uma das ferramentas de gerenciamento de projetos em lista extremamente versátil, e que pode ser ajustado de acordo com as necessidades da equipe e do projeto, nele foi elencado todos requisitos do sistema, no qual foram organizados através de listas de acordo com seu status no momento.

A imagem a seguir apresenta o *Kanban* do sistema Eduplan:

Figura 23 - Kanban do Eduplan



Fonte: Os Autores.

As listas elencadas no Kanban do projeto foram: A Realizar, onde está listado todos os requisitos que ainda não foram desenvolvidos no sistema. Em Andamento, é os requisitos que estão sendo desenvolvido do momento. A corrigir, são os requisitos nos quais foram reportados erros e estão disponíveis para a correção. Concluído, são os requisitos já finalizados e que estão em produção.

3.1.6 *Sprint* 3 – Desenvolvimento Cadastrar, Manter Usuários e Autenticação

A *Sprint* se iniciou com uma reunião para a definição dos requisitos a serem desenvolvidos, e ficou pré-definido que o desenvolvimento dos requisitos Cadastrar e Manter Usuários, onde deu se início ao desenvolvimento do *back-end* primeiramente, com a devida segurança com o sistema de *login* utilizando o JWT (*JSON Web Token*) para a segurança quanto ao *login*. Logo ao final do desenvolvimento do *back-end* dos requisitos, deu início a criação do *front-end*, com a utilização do *template* gratuito AdminLTE, onde foi criado o menu com as opções de criar um novo usuário e listar os usuários, na lista podendo editar e apagar um usuário se o usuário logado for “diretor”.

Logo após finalizar as funcionalidades de novo, editar e apagar usuário foi criado a página (componente) de login, onde o usuário para ter acesso ao sistema necessita de um *e-mail*

cadastrado no sistema e a senha, para que seja feita a autenticação do usuário concedendo o acesso seguro ao sistema e às funcionalidades atribuídas ao seu perfil.

3.1.7 Sprint 4 – Desenvolvimento Cadastrar e Manter Disciplinas

No início da *Sprint* foi realizada a reunião para definição do requisito a ser desenvolvido, também foram relatados algumas mudanças e erros a serem corrigidos nos requisitos desenvolvidos anteriormente, essas alterações foram realizadas antes do desenvolvimento do requisito elencado para esta *sprint*. Logo após foi iniciado o desenvolvimento dos requisitos que por sua vez foram Cadastrar e Manter Disciplinas, primeiramente foi definido quais os campos seriam padrões das disciplinas nos planos de ensino, assim foi criado o *back-end* com os devidos campos. O *front-end* foi criado o menu de Disciplinas, que será exibido somente para o *admin* (diretor), com o sub-menu Nova e Lista de disciplinas, na lista podendo editar, detalhar e apagar uma disciplina.

3.1.8 *Sprint* 5 – Desenvolvimento Cadastrar e Manter Planos de Ensino

A Sprint foi iniciada com a reunião para levantar o que seria atribuído a ela, e se havia algo a ser modificado ou corrigido das *sprints* anteriores, foram corrigido alguns campos do formulário de disciplina, e as permissões sobre ela, logo após iniciou-se o desenvolvimento do que foi atribuída à *sprint*, o requisito Cadastrar e Manter Planos de Ensino, o *back-end* foi desenvolvido tendo como base para criação da entidade os planos de ensino utilizado hoje pela instituição, assim criando os campos necessários para a criação do plano. O front-end foi criado um menu Plano de Ensino com os *sub-menus* Novo e Lista, o novo sendo para o preenchimento do formulário para a criação de um novo plano de ensino, no sub-menu lista de disciplinas, pode-se editar, detalhar e apagar um plano, ao abrir o menu detalhes de um plano é possível ver todos os campos preenchidos com os devidos dados, e no final um botão Gerar PDF, onde o usuário pode gerar um PDF com o *template* da instituição caso seja necessário.

3.1.9 Implementações futuras

Durante o tempo de desenvolvimento do projeto dentro da disciplina de Trabalho de Conclusão de Curso não foi possível finalizar todos os requisitos do sistema, assim ficou alguns para implementações a serem feitas ou finalizadas futuramente, abaixo uma lista dessas atividades:

- Finalizar a geração de PDF do Plano de Ensino de acordo com o *template* da instituição;
- Status dos Planos de Ensino;
- Notificação via E-mail;
- Possível integração com o Lyceum;

4 Considerações Finais

Com o plano de ensino sendo essencial em todo início de semestre dos Cursos de Bacharelados em Computação da instituição UniEVANGÉLICA - Centro Universitário de Anápolis, onde é necessário ser produzido um plano de cada uma das disciplinas presentes na grade do curso e precisando estar de acordo com a Proposta Pedagógica Curricular (PPC), o Eduplan veio para solucionar as cansativas tarefas de desenvolvimento e validação dos planos de ensino a fim proporcionar melhores resultados aos envolvidos, utilizando a tecnologia como meio de melhorar a eficiência no desenvolvimento dos planos.

O sistema conta com o cadastro de usuários, no qual podendo ser futuramente integrado com o *lyceum* para que possa ser utilizado os dados já cadastrados, no sistema pode ser cadastrado dois tipos de usuários distintos, como “diretor e docente”, no menu conta somente com os itens atribuídos a cada um deles por meio do tipo do perfil, sendo o diretor podendo criar e manter usuários, criar e manter disciplinas e podendo também criar e manter planos de ensino, já o perfil docente podendo somente criar e manter os planos de ensinos que o mesmo tenha criado. Alguma das funcionalidades propostas ainda não foram finalizadas, como o sistema de status do plano, a notificação via e-mail e o *template* da instituição na geração do PDF plano de ensino.

A experiência e o conhecimento adquiridos no decorrer de toda graduação, possibilitou que os autores colocassem em prática tudo que foi aprendido, através de pesquisas, ferramentas e métodos para solucionar o problema proposto. No qual, devido a problemas relacionados com o desenvolvimento do sistema e estimativas realizadas no início do projeto não foi possível implementar todas as funcionalidades para a conclusão do projeto, problemas estes relacionados à definição das tecnologias, inicialmente seria o *firebase*, após identificar que a equipe teria uma certa dificuldade na implementação foi definido o *java* como linguagem de desenvolvimento e o banco de dados *mongodb*.

Sendo assim tem-se trabalhos futuros para concluir o desenvolvimento do projeto, sendo elas feito pelos próprios autores ou terceiros, seguindo os conceitos abordados neste trabalho, a fim de alcançar a conclusão do mesmo e por fim a disponibilização da prova de conceito.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, W. P. **Banco de Dados**. 1ª. ed. São Paulo: Érica, 2014.

ANASTASIOU, L. D. G. C.; ALVES, L. P. **Processor de Ensinagem na Universidade**. 9. ed. Joenvile, SC: UNIVILLE, 2010.

ANDERSON, D. J. **The Kanban Primer:** A Cultural Evolotion in Software. [S.l.]: [s.n.], 2008.

BOAGLIO, F. **MongoDB:** Construa novas aplicações com novas tecnologias. 1ª. ed. São Paulo: Casa do Código, 2015.

BOAGLIO, F. **Spring Boot - Acelere o desenvolvimento de microsserviços**. 1ª. ed. São Paulo: Casa do Codigo, 2017.

BRASILEIRO, R. O que é Kanban. **Metodo Agil**, 2018. Disponivel em: <http://www.metodoagil.com/o-que-e-kanban/>. Acesso em: 15 out. 2019.

EGESTOR. O que é e como funciona o método, 2017. Disponivel em: <https://blog.egestor.com.br/o-que-e-e-como-funciona-o-metodo-kanban/>. Acesso em: 13 out. 2019.

FURGERI, S. **Java 8 - Ensino Didático - Desenvolvimento e Implementação de Aplicações**. 1ª. ed. São Paulo: Érica, 2015.

GIT. **Git - Sobre controle de Versão**, 2018. Disponivel em: <https://git-scm.com/book/pt-br/v1/Primeiros-passos-Sobre-Controle-de-Vers%C3%A3o>. Acesso em: 15 maio 2019.

GOOGLE. Angular. **Angular.io**, 2019. Disponivel em: <https://angular.io/guide/architecture>. Acesso em: 25 set. 2019.

HUMBLE, J.; FARLEY, D. **Entrega Contínua:** Como entregar Software de forma rápida e confiável. 1ª. ed. Porto Alegre: Bookman, 2014.

IFPR. IFPR. **Instituto Federal do Paraná**, 2014. Disponivel em: <http://reitoria.ifpr.edu.br/wp-content/uploads/2014/06/Orienta%C3%A7%C3%B5es-Plano-de-Ensino.pdf>. Acesso em: 05 maio 2019.

KNIBERG, H.; SKARIN, M. **Kanban and Scrum:** Making the Most of Both. [S.l.]: [s.n.], 2010.

O'GRADY, S. RedMonk. **RedMonk**, 2018. Disponivel em: <https://redmonk.com/sogrady/2018/03/07/language-rankings-1-18/>. Acesso em: 15 abr. 2019.

OKI, W. DevMedia. **Primeiros passos com o Spring Boot**, 2015. Disponivel em: <https://www.devmedia.com.br/primeiros-passos-com-o-spring-boot>. Acesso em: 10 maio 2019.

PRESSMAN, S.; MAXIM, B. R. **Engenharia de Software:** Uma Abordagem Profissional. 8ª. ed. Porto Alegre: AMGH, 2016.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistema de Gerenciamento de Banco de Dados**. 3ª. ed. São Paulo: McGraw-Hill, 2011.

SBROCCO, J. H. T. D. C.; MACEDO, C. **Metodologias Ágeis Engenharia de Software sob Medida**. 1ª. ed. São Paulo: Érica, 2012.

SCHWABER, ; SUTHERLAND,. Guia do Scrum. **Scrum Guides**, 2013. Disponivel em: <https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 12 maio 2019.

SOMMERVILLE, I. **Engenharia de Software**. 9ª. ed. São Paulo: Pearson, 2013.

WEED, C. **Introduction to Git-flow**. Boston: [s.n.], 2014.

APÊNDICES

APÊNDICE A – DOCUMENTO DE VISÃO

DOCUMENTO DE VISÃO

Eduplan – Sistema de Criação e Gerenciamento de Planos de Ensino

VERSÃO 0.4

Autores:
David Ferraz Pires
Jonathan Ramos Nascimento

Anápolis – GO
2019

Histórico da Revisão

Data	Versão	Descrição	Autor
28/03/2019	0.1	Criação do documento de visão.	Jonathan Ramos Nascimento
05/04/2019	0.2	Revisões e correções.	David Ferraz Pires
15/05/2019	0.3	Alterações necessárias no problema e nos Requisitos.	Jonathan Ramos Nascimento
04/11/2019	0.4	Alterações nos status dos requisitos	Jonathan Ramos Nascimento

Sumário

1. STAKEHOLDERS

JANEIRO – 2019/01		
Nome	**Cargo**	**Responsabilidade**
Jonathan Ramos Nascimento	Desenvolvedor e Scrum Master	Desenvolver o sistema de acordo com as necessidades e demandas do cliente.
Adrielle Beze Peixoto	Cliente	
Millys Fabrielle Araujo Carvalhaes	Cliente	
David Ferraz Pires	*Product Owner*	Responsável pelo Backlog do sistema.

2. ETAPAS DO PROCESSO

2.1 Levantamento de requisitos

- Documento de Visão
- Diagrama de Casos de Uso

2.2 Histórias de usuários

- Backlog do produto

2.3 Priorizar Requisitos

2.4 Backlog da Sprint

2.5 Codificação

- Concepção
- Especificação
- Design de Interfaces
- Prototipação
- Codificação

2.6 Testes

- Casos de Teste
- Teste Unitário
- Teste de Integração

2.7 Retrospectiva

- Dificuldades durante a Sprint
- Lições aprendidas

3. PROBLEMA

O Plano de Ensino é um documento redigido todo início de semestre, no qual os professores fazem um planejamento da disciplina a ser ministrada, e interligam os objetivos, o conteúdo e suas metas. O documento e composto de várias informações, tais como: caracterização da disciplina, professor, ementa, objetivos gerais e específicos, habilidades e competências, conteúdo programático, procedimentos didáticos, atividade integrativa, processo avaliativo da aprendizagem e as bibliografias.

Todas as informações são agrupadas em um template fornecido pela instituição, e o professor tem a responsabilidade de editar alguns campos, enquanto outros são fixos de acordo com o PPC (Projeto Pedagógico do Curso). Após a finalização o docente envia o plano de ensino para a coordenação pedagógica do curso para que seja feita a avaliação individual de cada plano de ensino, a qual verifica se o plano está em conformidade com o PPC, se o calendário está de acordo com todas as atividades do curso e atividades institucionais, se corresponde a todo direcionamento institucional que é recebido, e caso não esteja é retornado o plano de ensino para o professor solicitando as alterações, e este processo é feito até que seja aprovado pela análise da coordenação pedagógica do curso e posteriormente liberado para a impressão.

Conforme entrevistas com professores dos Cursos de Bacharelados em Computação da instituição UniEVANGÉLICA - Centro Universitário de Anápolis, foi identificado que uma das tarefas que consome muito tempo dos docentes e também da coordenação pedagógica no início de todos os períodos letivos é a elaboração dos planos de ensino das disciplinas.

Dada a problemática descrita acima, como uma aplicação web pode ajudar os professores e coordenadores na criação dos Planos de Ensino?

4. PRIORIDADES DOS REQUISITOS

A priorização dos requisitos será realizada utilizando a técnica MOSCOW, com os seguintes *status*:

- **Essencial**: Requisitos nos quais são importantes para o funcionamento correto do sistema.
- **Importante**: Requisitos nos quais deveriam ser implementados no sistema, mas o sistema funcionará sem eles, mas não atendera os requisitos do cliente.

- **Poderia**: Requisitos que podem ser implementados no sistema para auxiliar em alguma função, a não implementação deste não causará danos ao sistema.
- **Interessante**: Requisitos que seriam interessantes sem implementados no sistema.

5. VISÃO DO PRODUTO

O Eduplan tem como objetivo auxiliar a criação dos planos de ensino, permitir que a coordenação tenha maior controle dos planos, pois somente poderá ser preenchido pelos docentes os campos que lhes for direcionado, os demais estarão bloqueados para a edição.

6. REQUISITOS FUNCIONAIS

6.1 Autenticar Usuário

RF.	Descrição	Status	Prioridade
RF01	Autenticar Usuário	Em andamento	Essencial
Detalhamento: Este requisito permite que somente usuários autorizados realize o *login* para utilização do sistema.			

6.2 Criar Usuário

RF.	Descrição	Status	Prioridade
RF02	Criar Usuário	Em andamento	Essencial
Detalhamento: Este requisito permite que somente gestor(administrador) do sistema cadastre novos usuários.			

6.3 Editar Usuário

RF.	Descrição	Status	Prioridade
RF03	Editar Usuário	Em andamento	Essencial
Detalhamento: Este requisito permite que somente o gestor(administrador) realize edições nos usuários já cadastrados no sistema.			

6.4 Visualizar Usuário

RF.	Descrição	Status	Prioridade
RF04	Visualizar Usuário	Em andamento	Poderia
Detalhamento: Este requisito permite que somente o gestor(administrador) visualize os dados dos usuários cadastrados no sistema.			

6.5 Deletar Usuário

RF.	Descrição	Status	Prioridade
RF05	Deletar Usuário	Concluído	Essencial
Detalhamento: Este requisito permite que somente o gestor(administrador) apague usuários cadastrados no sistema.			

6.6 Criar Disciplina

RF.	Descrição	Status	Prioridade
RF06	Criar Disciplina	Concluído	Essencial
Detalhamento: Este requisito permite que somente o gestor(administrador) possa criar uma nova disciplina.			

6.7 Editar Disciplina

RF.	Descrição	Status	Prioridade
RF07	Editar Disciplina	Concluído	Essencial
Detalhamento: Este requisito permite que o gestor(administrador) e docentes editem as disciplinas.			

6.8 Visualizar Disciplinas

RF.	Descrição	Status	Prioridade
RF08	Visualizar Disciplina	Concluído	Essencial
Detalhamento: Este requisito permite que o gestor(administrador) visualize todas as disciplinas cadastradas no sistema.			

6.9 Deletar Disciplina

RF.	Descrição	Status	Prioridade
RF09	Deletar Disciplina	Concluído	Essencial
Detalhamento: Este requisito permite que somente o gestor(administrador) deletar as disciplinas cadastradas no sistema.			

6.10 Criar Plano de Ensino

RF.	Descrição	Status	Prioridade
RF10	Criar Plano de Ensino	Concluído	Essencial
Detalhamento: Este requisito permite que o gestor e docente possa criar um novo plano de ensino, que será automaticamente atribuído a ele mesmo.			

6.11 Editar Plano de Ensino

RF.	Descrição	Status	Prioridade
RF11	Editar Plano de Ensino	Concluído	Essencial
Detalhamento: Este requisito permite que o gestor(administrador) e docentes editem os planos de ensino e preencha os campos que a eles são permitidos.			

6.12 Visualizar Plano de Ensino

RF.	Descrição	Status	Prioridade
RF12	Visualizar Plano de Ensino	Concluído	Essencial
Detalhamento: Este requisito permite que o gestor(administrador) visualize todos os planos de ensino cadastrados no sistema, e o docente somente o(s) alocado(s) a ele.			

6.13 Deletar Plano de Ensino

RF.	Descrição	Status	Prioridade
RF13	Deletar Plano de Ensino	Concluído	Essencial
Detalhamento: Este requisito permite que o gestor(administrador) e docente deletar planos de ensinos cadastrados no sistema.			

6.14 Visualizar Usuários

RF.	Descrição	Status	Prioridade
RF14	Visualizar Usuários	Concluído	Poderia
Detalhamento: Este requisito permite que somente o gestor(administrador) avalie o plano de ensino se estiver de acordo com as normas da instituição, e aprove ou recuse-o direcionando o que está incorreto para que o docente possa fazer a correção do mesmo.			

6.15 Imprimir Plano de Ensino

RF.	Descrição	Status	Prioridade
RF15	Imprimir Plano de Ensino	Em andamento	Poderia
Detalhamento: Este requisito permite que o gestor(administrador) e o docente imprima o plano de ensino se ele estiver aprovado.			

7. REQUISITOS NÃO FUNCIONAIS

7.1 Usabilidade

Descrição	Status	Prioridade
Usabilidade	Em andamento	Essencial
Detalhamento: O usuário final deve utilizar o sistema com facilidade para entender o que cada tela e botão representam, e tenha sua interface e design intuitivo.		

7.2 Desempenho

Descrição	Status	Prioridade
Desempenho	Em andamento	Essencial
Detalhamento: O sistema deve ser executado em ambiente web e deve ter um tempo de resposta no máximo entre 1 – 5 segundos.		

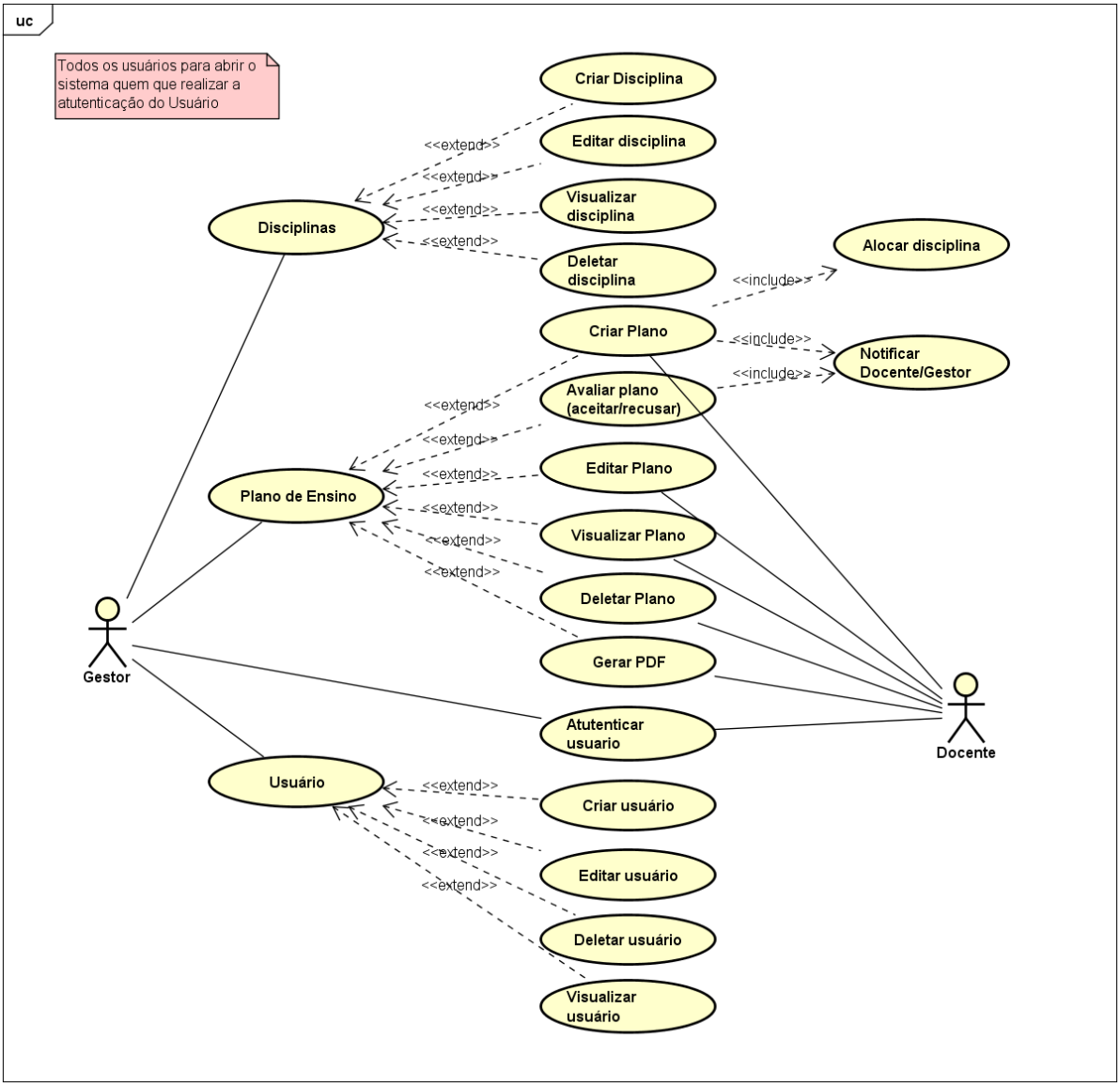
7.3 Confiabilidade

Descrição	Status	Prioridade
Confiabilidade	Em andamento	Essencial
Detalhamento: Todas as informações e resultados devem ser de acordo com as inseridas previamente, sendo os dados originais e não perca e não alterem dados sem que seja solicitado pelo usuário.		

7.4 Segurança

Descrição	Status	Prioridade
Segurança	Em andamento	Essencial
Detalhamento: Somente pessoas cadastradas no sistema e com devidas permissões terão acesso aos planos e aos usuários cadastrados.		

APÊNDICE B – DIAGRAMA DE CASO DE USO



APÊNDICE C – HISTÓRIA DE USUÁRIO

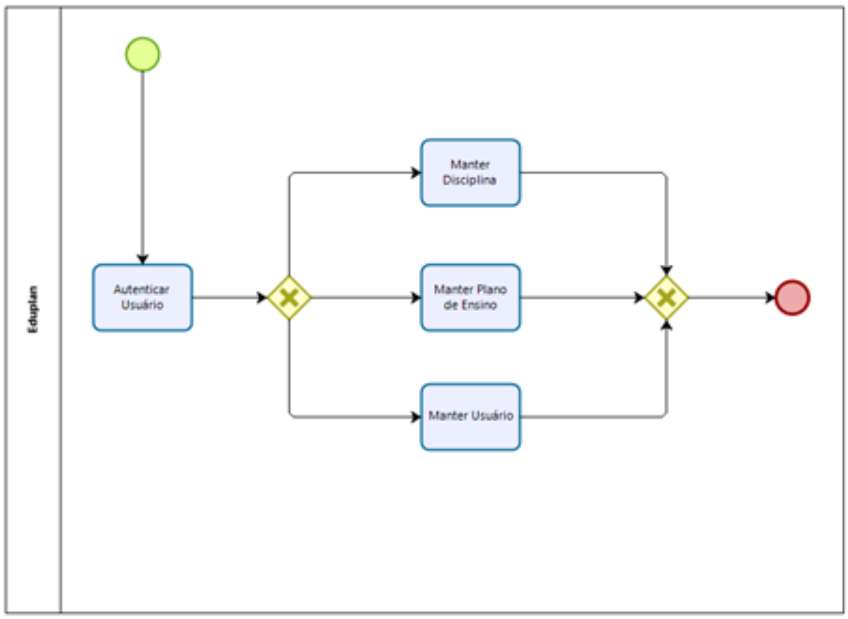
EduPlan

Bizagi Modeler

www.bizagi.com

Índice

1 Usuário


Eduplan
Autenticar Usuário
Manter Disciplina
Manter Plano de Ensino
Manter Usuário

bizagi

1.1 Eduplan

1.1.1 Elementos do processo

1.1.1.1 ☐ *Autenticar Usuário*

Histórico de Versão

Versão	Data	Responsável	Descrição
0.1	15/08/2019	David Ferraz Pires	Documentação e especificação em história de usuário
0.2	22/10/2019	David Ferraz Pires	Correção de História de Usuário

Personas

Administrador

Usuário

Pré-Condições

A persona deve estar cadastrada na base de dados com e-mail, perfil de acesso e matrícula ativos.

Como Acessar

Acesse a página de login do EduPlan.

História de Usuário

COMO persona **POSSO** controlar o acesso **PARA** acessar as funcionalidades do sistema, fazendo login.

Cenários

Acessar o Sistema

Sair do Sistema

Mockups

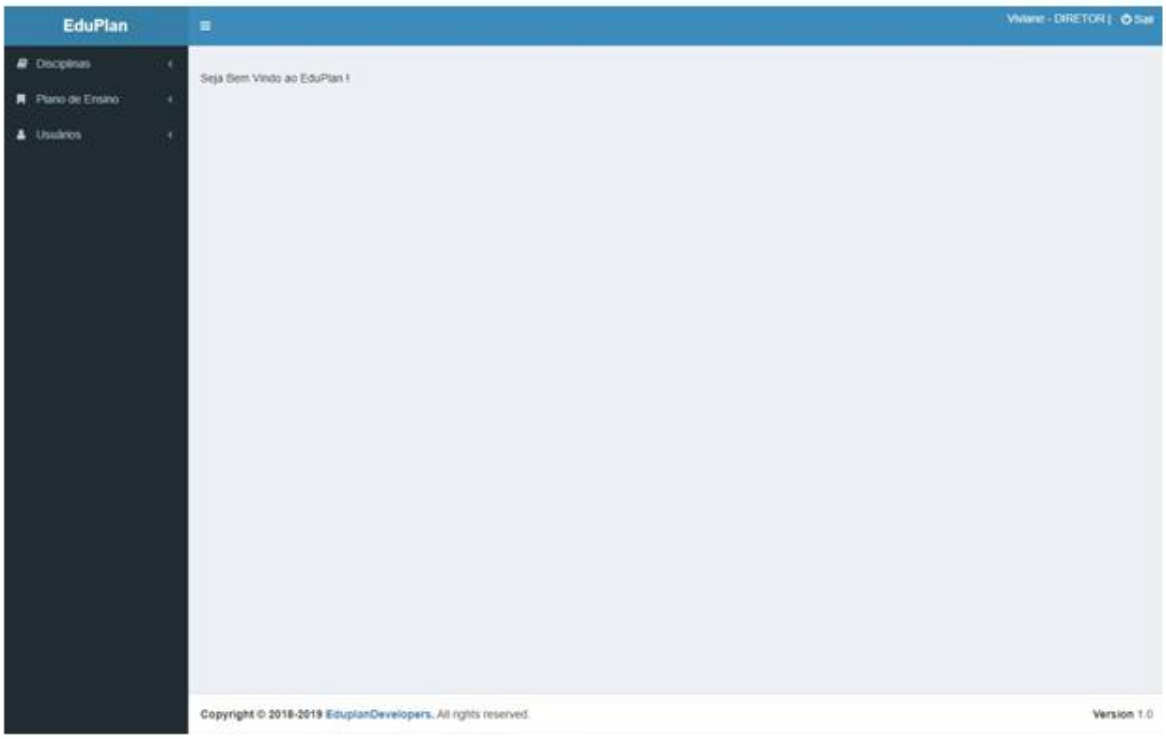
Login

DESCRIÇÃO DE CAMPOS				
Nome do campo	**Tipo do campo**	**Tipo de dado(tamanho)**	**Editável**	**Obrigatório**
Email	Input Text	Especiais()	Sim	Sim
Senha	Input Text	Alfanumérico () com mascara	Sim	Sim

bizagi

DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Entrar	Válida o Email no banco de dados e direciona para a página "Home"	Button	Não	N/A

Home



DESCRIÇÃO DE COMANDOS

Nome do comando	Ação	Tipo	Tooltip	Restrições
Sair	Redireciona para a página de Login	Button	Não	N/A

1.1.1.2 ☐*Manter Disciplina*

Histórico de Versão

Versão	Data	Responsável	Descrição
0.1	27/08/2019	David Ferraz Pires	Documentação e especificação em história de usuário
0.2	17/10/2019	David Ferraz Pires	Correção História de Usuários
0.3	27/11/2019	David Ferraz Pires	Correção História de Usuários

Personas

Administrador

Pré-Condições

A persona deverá estar cadastrado na base de dados, estar logado no sistema e ter permissões para essa funcionalidade.

Como Acessar

Acessar o menu principal ->Disciplinas.

bizagi

História de Usuário

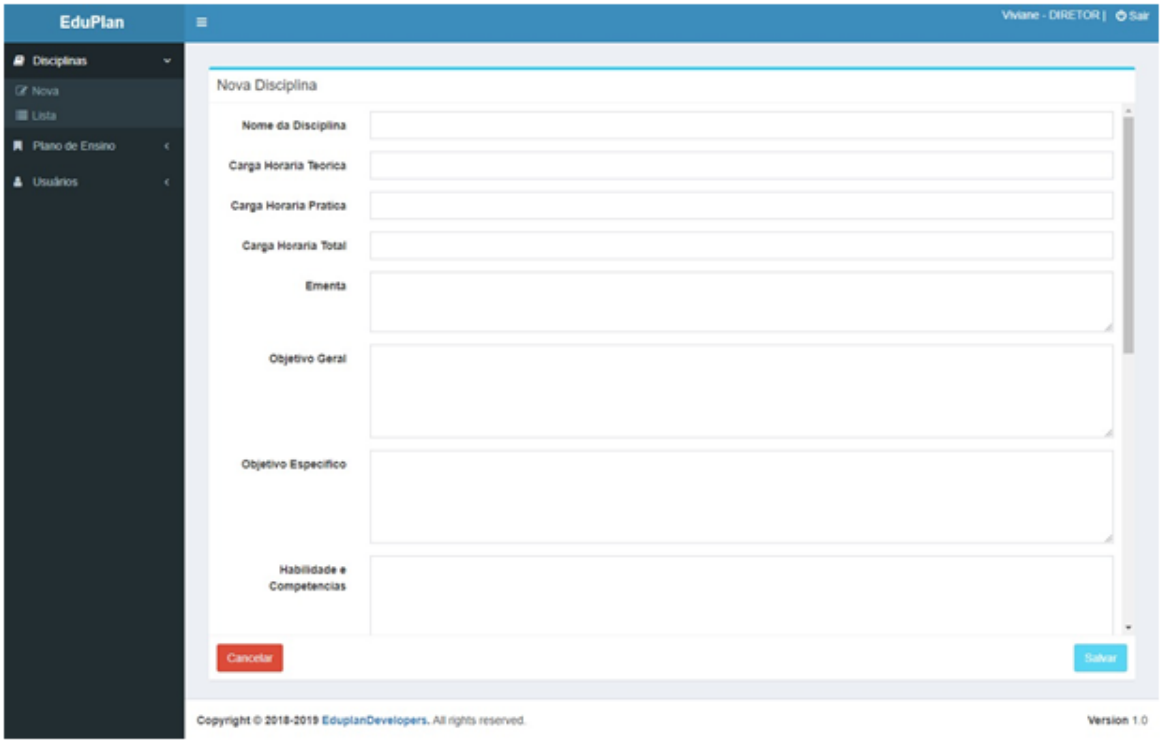
COMO persona **POSSO** cadastrar usuários **PARA** cadastrar planos de ensinos e as demais informações.

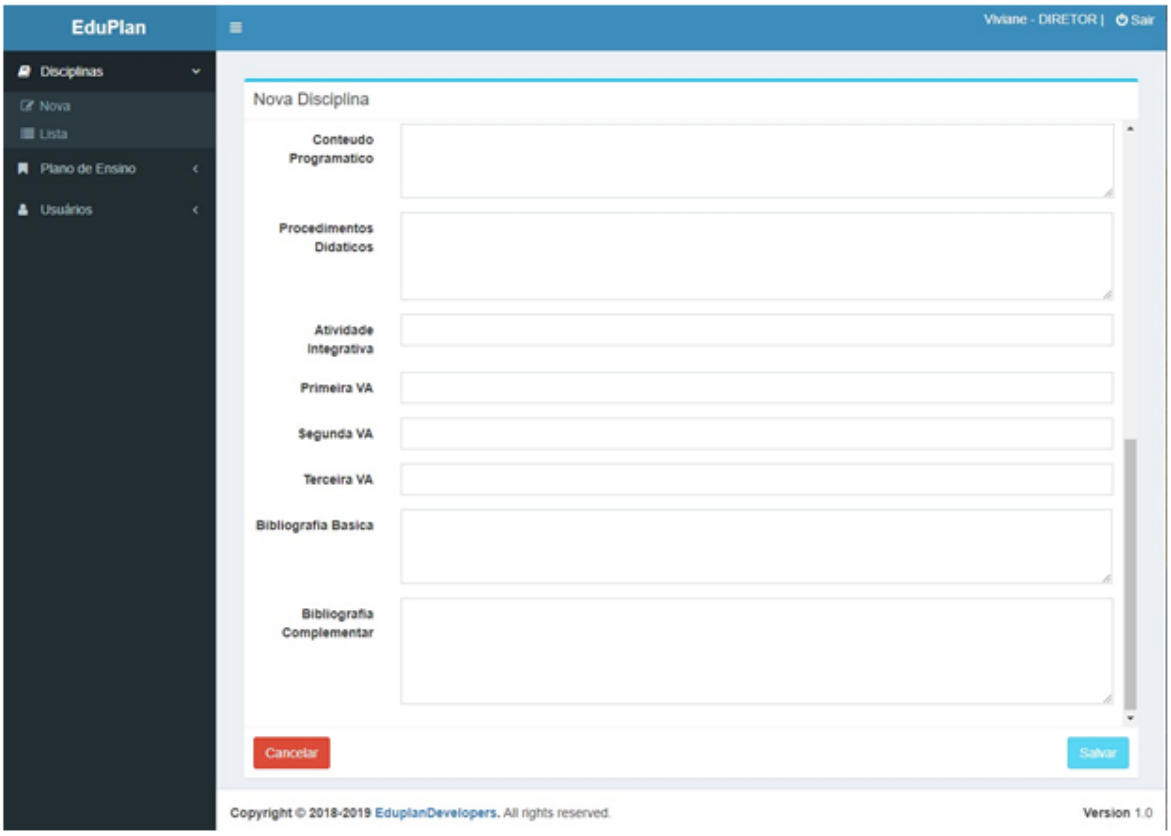
Cenários

Cadastrar

Mockups

Cadastrar Disciplina


EduPlan
Viviane - DIRETOR | Sair
Disciplinas
Nova
Lista
Plano de Ensino
Usuários
Nova Disciplina
Nome da Disciplina
Carga Horaria Teorica
Carga Horaria Pratica
Carga Horaria Total
Ementa
Objetivo Geral
Objetivo Especifico
Habilidade e Competencias
Cancelar
Salvar
Copyright © 2018-2019 EduplanDevelopers. All rights reserved.
Version 1.0

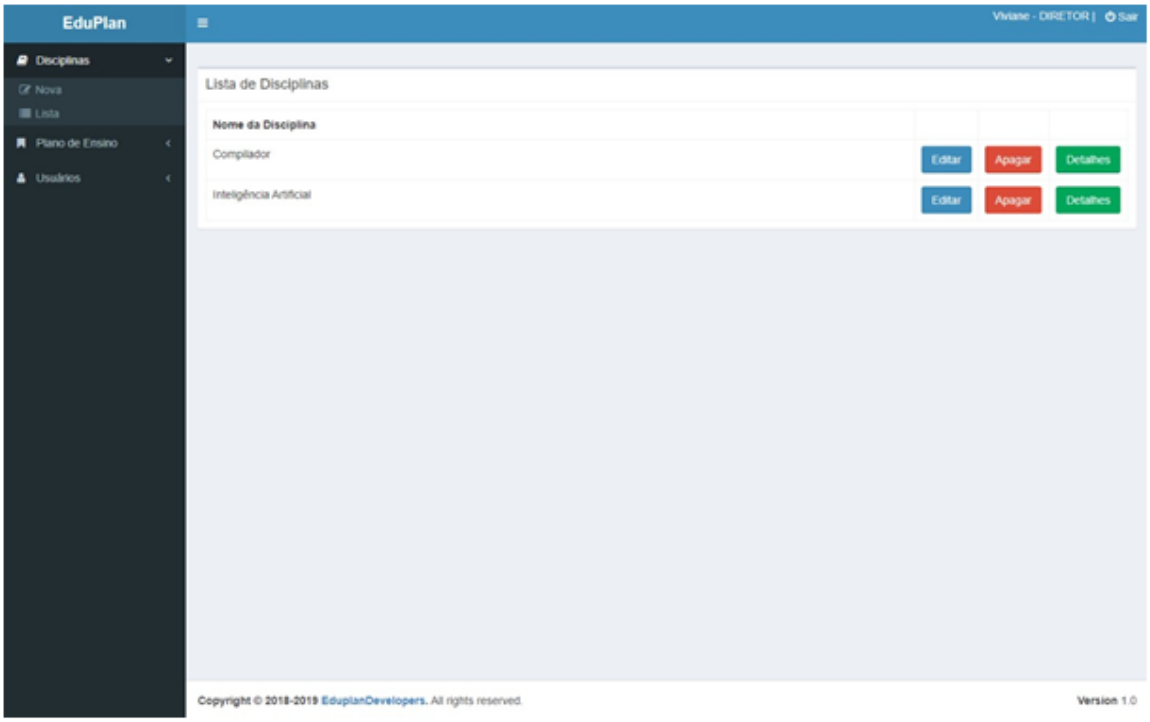


DESCRIÇÃO DE CAMPOS				
Nome do campo	**Tipo do campo**	**Tipo de dado(tamanho)**	**Editável**	**Obrigatório**
Nome da Disciplina	Input Text	Alfanumérico(30)	Sim	Sim
Carga Horaria Teorica	Input Text	Numérico(3)	Sim	Sim
Carga Horaria Pratica	Input Text	Numérico(3)	Sim	Sim
Carga Horaria Total	Input Text	Numérico(3)	Sim	Sim

Ementa	Input Text	Alfanumérico(500)	Sim	Sim
Objetivo Geral	Input Text	Alfanumérico(300)	Sim	Sim
Objetivo Especifico	Input Text	Alfanumérico(1000)	Sim	Sim
Habilidades e Competencias	Input Text	Alfanumerico(1000)	Sim	Sim
Conteudo Programatico	Input Text	Alfanumerico(2000)	Sim	Sim
Procedimentos Didaticos	Input Text	Alfanumerico(500)	Sim	Sim
Atividade Integrativa	Input Text	Alfanumerico(500)	Sim	Sim
Primeira VA	Input Text	Alfanumerico(300)	Sim	Sim
Segunda VA	Input Text	Alfanumerico(300)	Sim	Sim
Terceira VA	Input Text	Alfanumerico(300)	Sim	Sim
Bibliografia Basica	Input Text	Alfanumerico(200)	Sim	Sim
Bibliografia Complementar	Input Text	Alfanumerico(200)	Sim	Sim

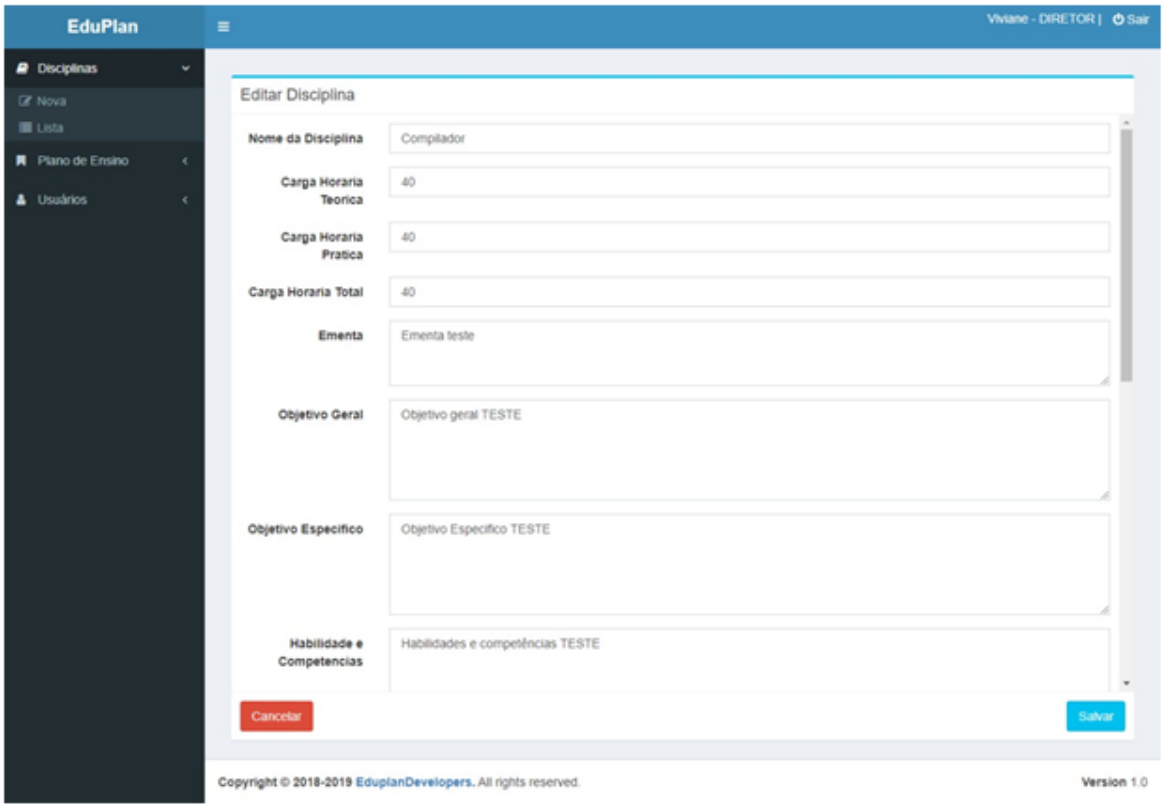
DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Cancelar	Cancela a ação anterior	Button	Não	N/A
Salvar	Salvar as informações na base de dados	Button	Não	N/A

Listar Disciplinas



DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Editar	Direciona para a página de alteração de disciplina.	Button	Não	N/A
Apagar	Exclui as informações do usuário da base de dados.	Button	Não	N/A
Detalhes	Direciona para pagina de visualização da disciplina	Button	Não	N/A

Editar Disciplinas

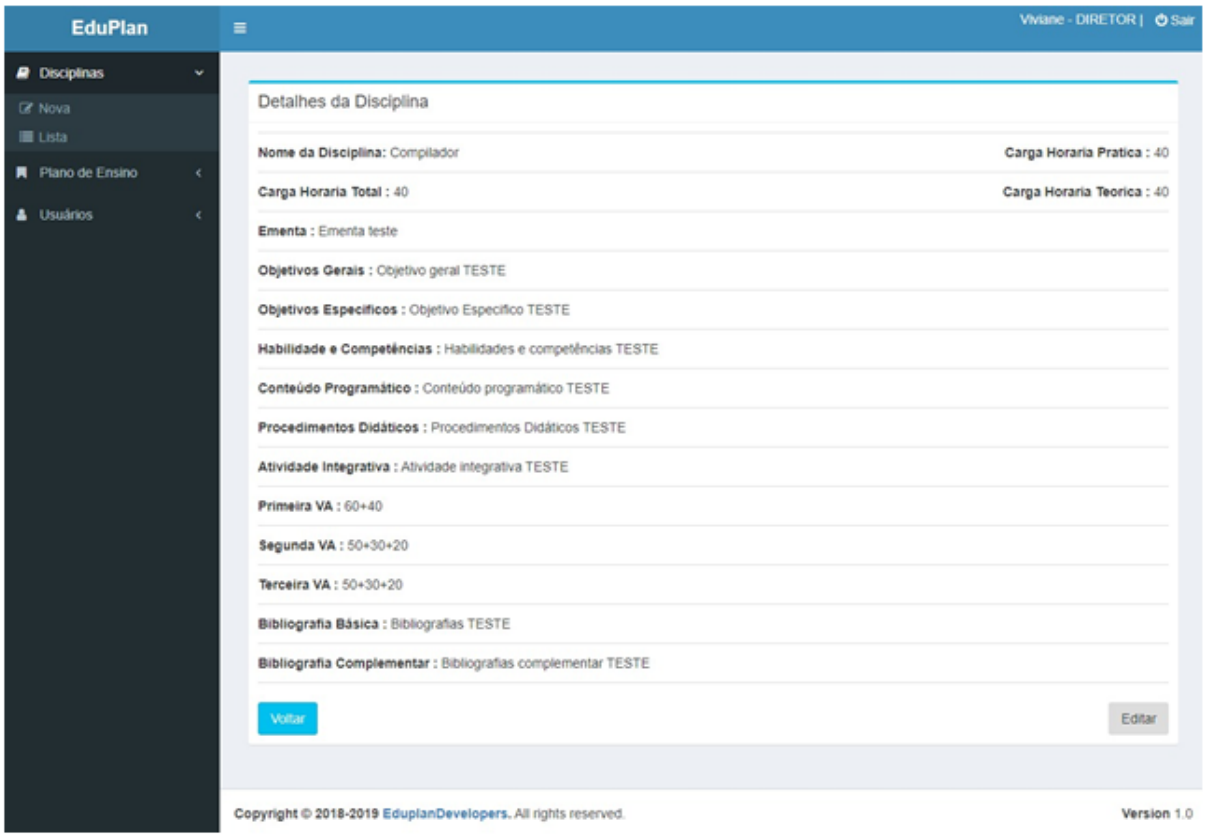


DESCRIÇÃO DE CAMPOS				
Nome do campo	**Tipo do campo**	**Tipo de dado(tamanho)**	**Editável**	**Obrigatório**
Nome da Disciplina	Input Text	Alfanumérico(30)	Sim	Sim
Carga Horaria Teorica	Input Text	Numérico(3)	Sim	Sim
Carga Horaria Pratica	Input Text	Numérico(3)	Sim	Sim

Carga Horaria Total	Input Text	Numérico(3)	Sim	Sim
Ementa	Input Text	Alfanumérico(500)	Sim	Sim
Objetivo Geral	Input Text	Alfanumérico(300)	Sim	Sim
Objetivo Especifico	Input Text	Alfanumérico(1000)	Sim	Sim
Habilidades e Competencias	Input Text	Alfanumerico(1000)	Sim	Sim
Conteudo Programatico	Input Text	Alfanumerico(2000)	Sim	Sim
Procedimentos Didaticos	Input Text	Alfanumerico(500)	Sim	Sim
Atividade Integrativa	Input Text	Alfanumerico(500)	Sim	Sim
Primeira VA	Input Text	Alfanumerico(300)	Sim	Sim
Segunda VA	Input Text	Alfanumerico(300)	Sim	Sim
Terceira VA	Input Text	Alfanumerico(300)	Sim	Sim
Bibliografia Basica	Input Text	Alfanumerico(200)	Sim	Sim
Bibliografia Complementar	Input Text	Alfanumerico(200)	Sim	Sim

DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Cancelar	Cancela a ação anterior	Button	Não	N/A
Salvar	Salvar as informações na base de dados	Button	Não	N/A

Detalhes Disciplinas



DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Voltar	Retorna a pagina anterior	Button	Não	N/A
Editar	Direciona para a página de alteração de disciplina.	Button	Não	N/A

bizagi

1.1.1.3 ☐*Manter Plano de Ensino*

Histórico de Versão

Versão	Data	Responsável	Descrição
0.1	28/08/2019	David Ferraz Pires	Documentação e especificação em história de usuário
0.2	19/10/2019	David Ferraz Pires	Correção Histórias de Usuários
0.3	22/11/2019	David Ferraz Pires	Correções de Histórias de Usuários

Personas

Administrador

Usuário

Pré-Condições

A persona deverá estar cadastrado na base de dados, estar logado no sistema e ter permissões para essa funcionalidade.

Para cadastrar um plano de ensino é necessário que tenha pelo menos uma disciplina cadastrada para realizar o vinculo.

Como Acessar

Acessar o menu principal ->Plano de Ensino.

História de Usuário

COMO persona **POSSO** cadastrar o plano de ensino **PARA** gerencia-los no sistema de acordo com o PPC.

Cenários

Cadastrar

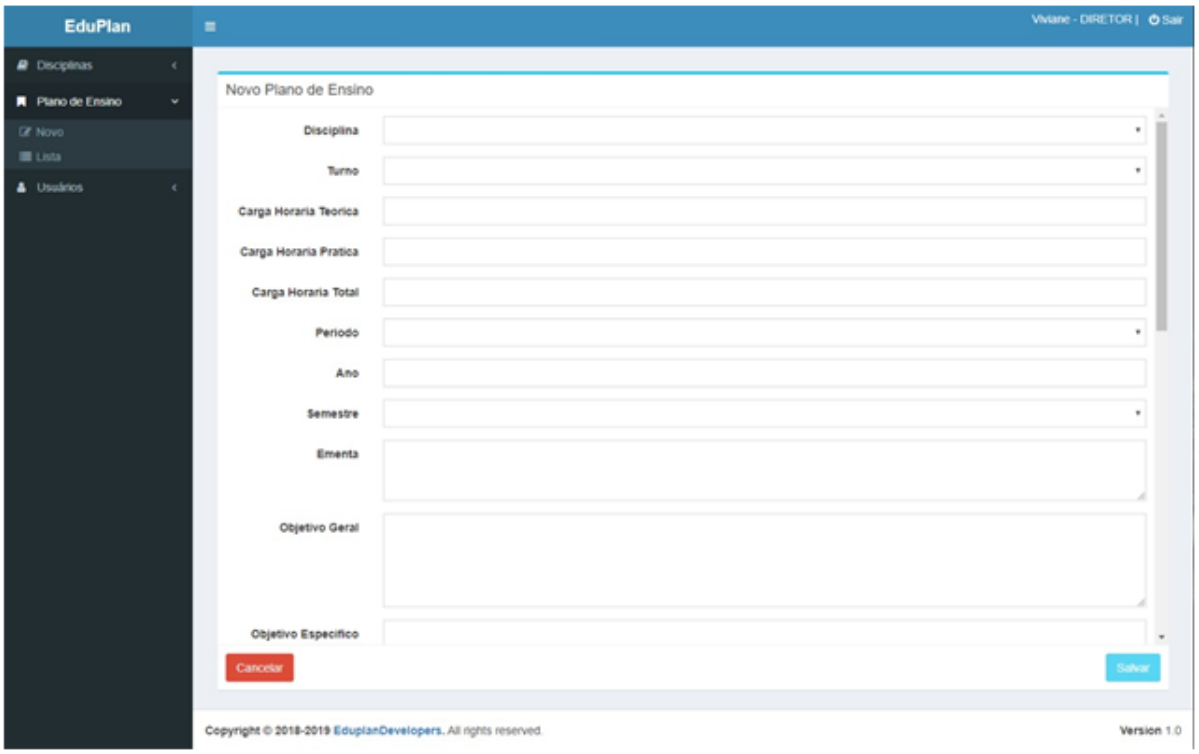
Listar

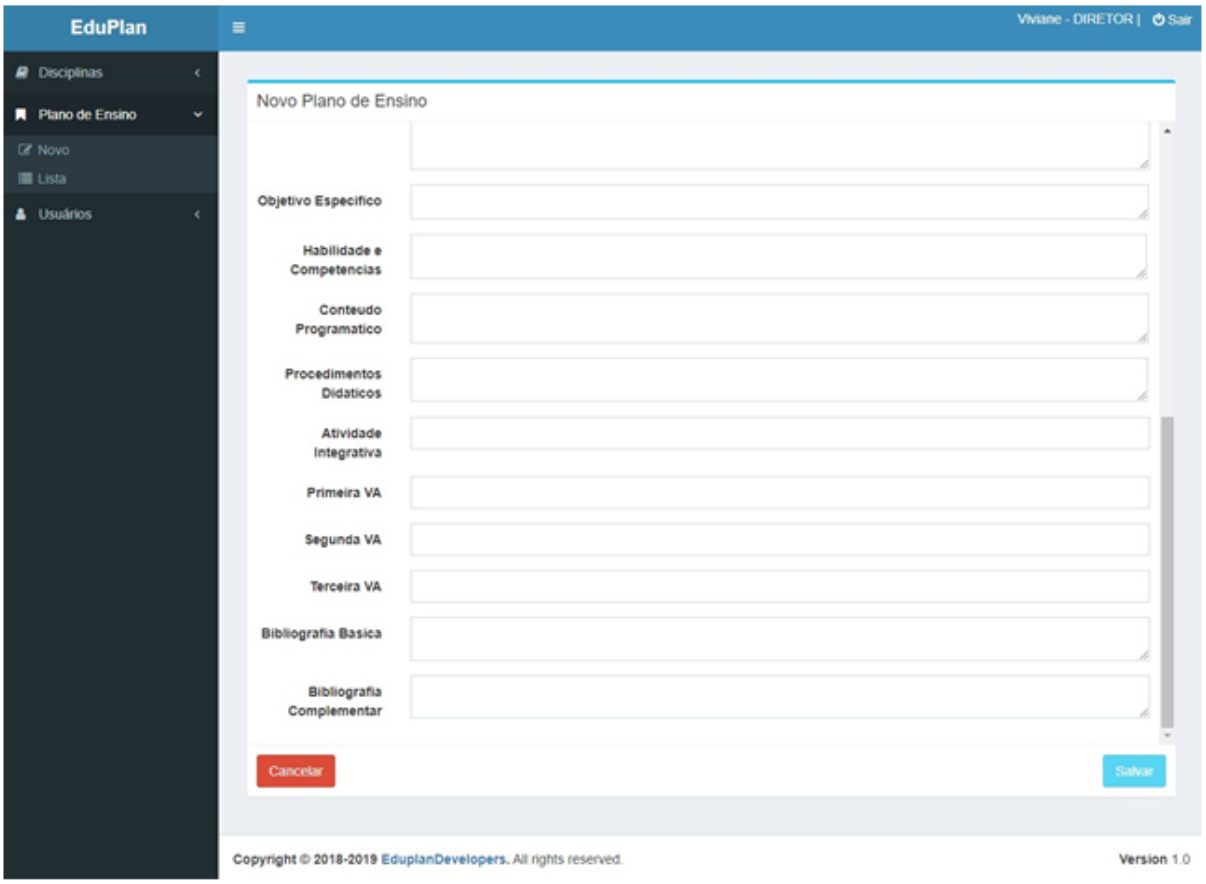
Editar

Detalhar

Mockups

Cadastrar Plano de Ensino


EduPlan
Viviane - DIRETOR | Sair
Disciplinas
Plano de Ensino
Novo
Lista
Usuários
Novo Plano de Ensino
Disciplina
Turno
Carga Horaria Teorica
Carga Horaria Pratica
Carga Horaria Total
Periodo
Ano
Semestre
Ementa
Objetivo Geral
Objetivo Especifico
Cancelar
Salvar
Copyright © 2018-2019 EduplanDevelopers. All rights reserved.
Version 1.0

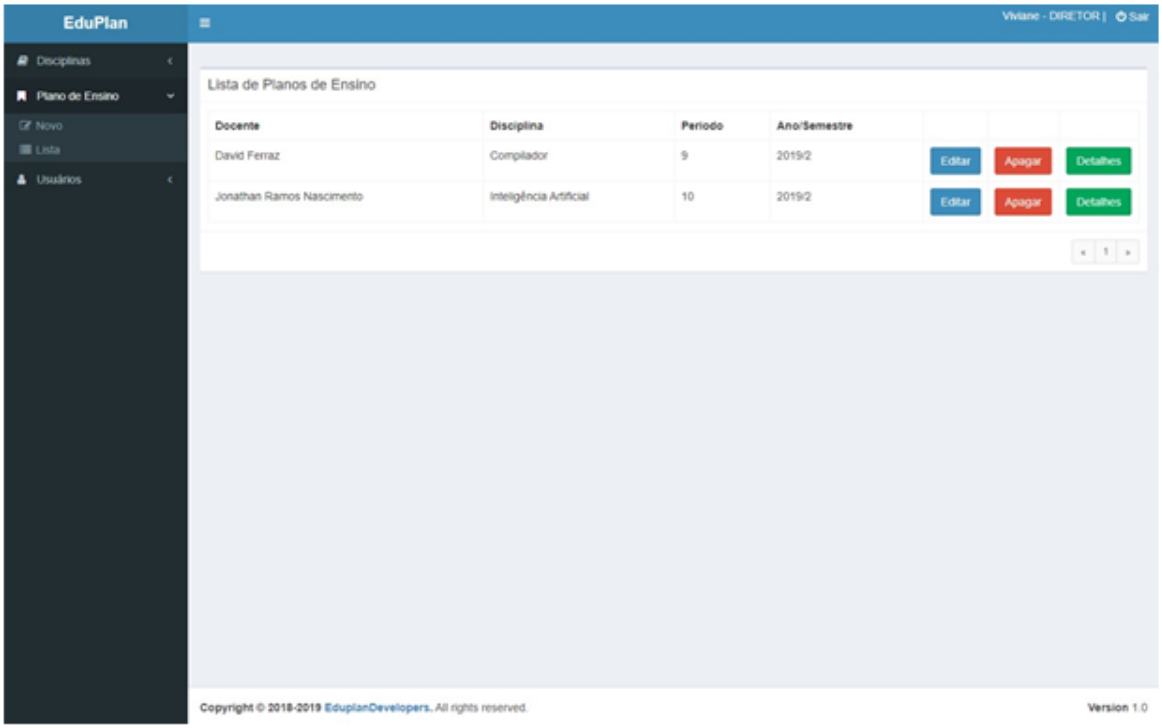


DESCRIÇÃO DE CAMPOS				
Nome do campo	**Tipo do campo**	**Tipo de dado(tamanho)**	**Editável**	**Obrigatório**
Ementa	Input Text	Alfanumérico(500)	Sim	Sim
Objetivo Geral	Input Text	Numérico(300)	Sim	Sim
Objetivos	Input Text	Numérico(1000)	Sim	Sim

Específicos				
Habilidades e Competências	Input Text	Numérico(1000)	Sim	Sim
Conteúdo Programático	Input Text	Alfanumérico(2000)	Sim	Sim
Procedimentos Didáticos	Input Text	Alfanumérico(500)	Sim	Sim
Atividade Integrativa	Input Text	Alfanumérico(500)	Sim	Sim
Primeira VA	Input Text	Alfanumerico(300)	Sim	Sim
Segunda VA	Input Text	Alfanumerico(300)	Sim	Sim
Terceira VA	Input Text	Alfanumerico(300)	Sim	Sim
Bibliografia Basica	Input Text	Alfanumerico(200)	Sim	Sim
Bibliografia Complementar	Input Text	Alfanumerico(200)	Sim	Sim

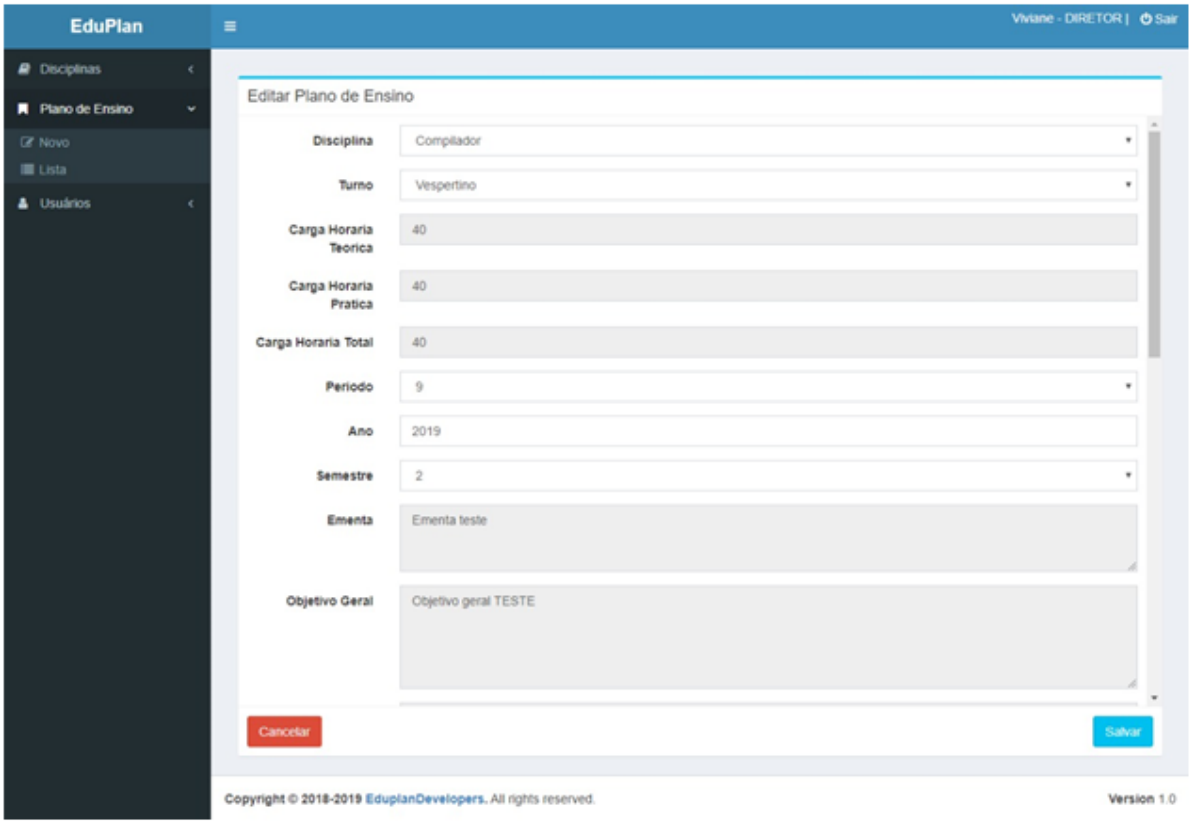
DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Cancelar	Cancela a ação anterior	Button	Não	N/A
Salvar	Salvar as informações na base de dados	Button	Não	N/A

Listar Planos de Ensino



DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Editar	Direciona para a página de alteração de disciplina.	Button	Não	N/A
Apagar	Exclui as informações do usuário da base de dados.	Button	Não	N/A
Detalhes	Direciona para pagina de visualização do Plano de Ensino	Button	Não	N/A

Editar Plano de Ensino

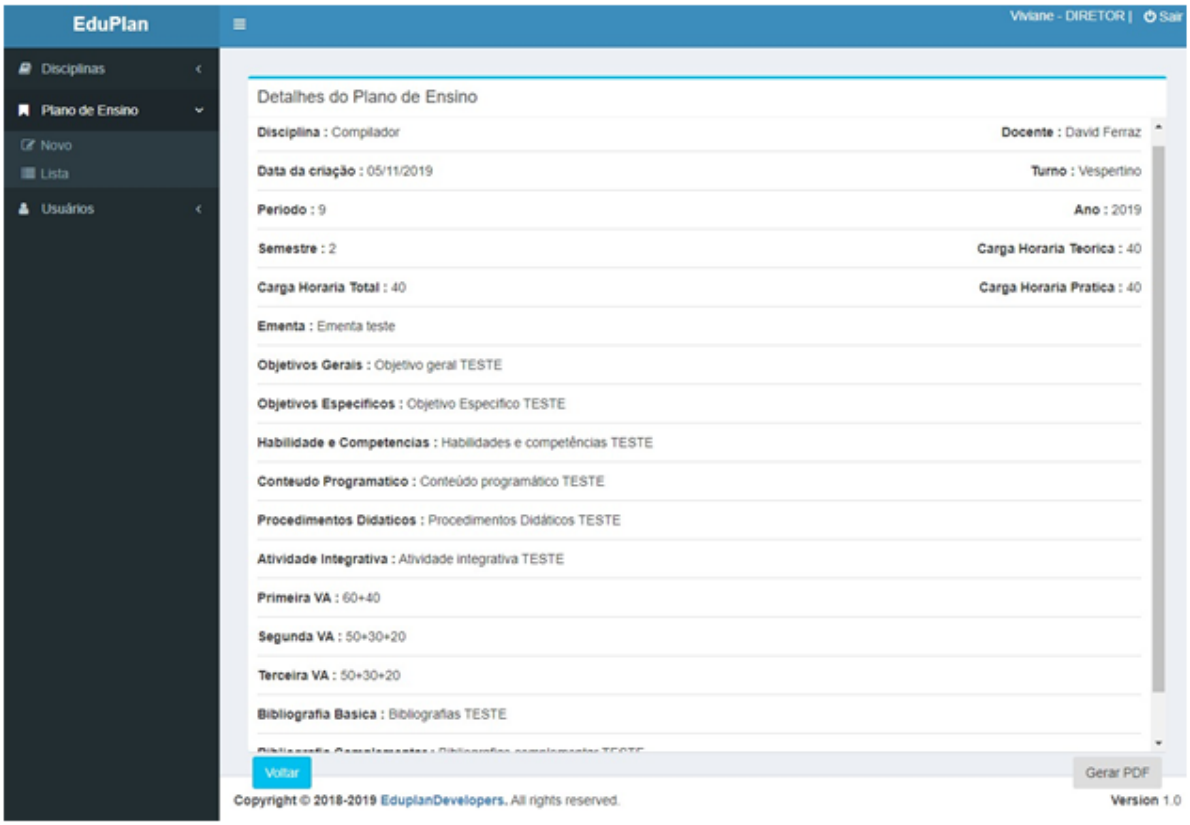


DESCRIÇÃO DE CAMPOS				
Nome do campo	**Tipo do campo**	**Tipo de dado(tamanho)**	**Editável**	**Obrigatório**
Ementa	Input Text	Alfanumérico(500)	Sim	Sim
Objetivo Geral	Input Text	Numérico(300)	Sim	Sim
Objetivos	Input Text	Numérico(1000)	Sim	Sim

Específicos				
Habilidades e Competências	Input Text	Numérico(1000)	Sim	Sim
Conteúdo Programático	Input Text	Alfanumérico(2000)	Sim	Sim
Procedimentos Didáticos	Input Text	Alfanumérico(500)	Sim	Sim
Atividade Integrativa	Input Text	Alfanumérico(500)	Sim	Sim
Primeira VA	Input Text	Alfanumerico(300)	Sim	Sim
Segunda VA	Input Text	Alfanumerico(300)	Sim	Sim
Terceira VA	Input Text	Alfanumerico(300)	Sim	Sim
Bibliografia Basica	Input Text	Alfanumerico(200)	Sim	Sim
Bibliografia Complementar	Input Text	Alfanumerico(200)	Sim	Sim

DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Cancelar	Cancela a ação anterior	Button	Não	N/A
Salvar	Salvar as informações na base de dados	Button	Não	N/A

bizagi



DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Voltar	Retorna para página anterior	Button	Não	N/A
Gerar PDF	Gera um arquivo PDF do plano de ensino	Button	Não	N/A

bizagi

1.1.1.4 ☐*Manter Usuário*

Histórico de Versão

Versão	Data	Responsável	Descrição
0.1	25/08/2019	David Ferraz Pires	Documentação e especificação em história de usuário
0.2	09/11/2019	David Ferraz Pires	Correções de Histórias de Usuários
0.3	21/11/2019	David Ferraz Pires	Correções de Histórias de Usuários

Personas

Administrador

Pré-Condições

A persona deverá estar cadastrado na base de dados, estar logado no sistema e ter permissões para essa funcionalidade.

Como Acessar

Acessar o menu principal ->Usuarios.

História de Usuário

COMO persona **POSSO** cadastrar usuários **PARA** gerencia-los no sistema.

Cenários

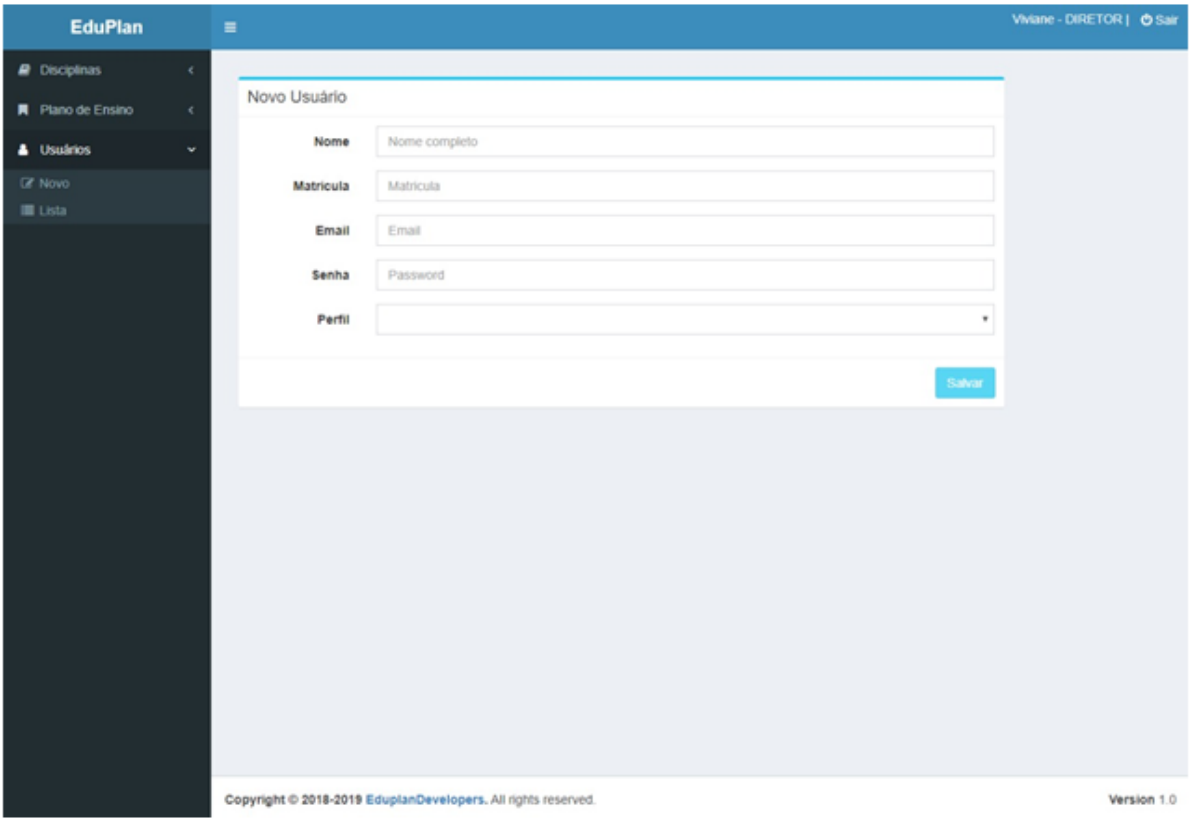
Cadastrar

Listar

Editar

Mockups

Cadastrar Usuários

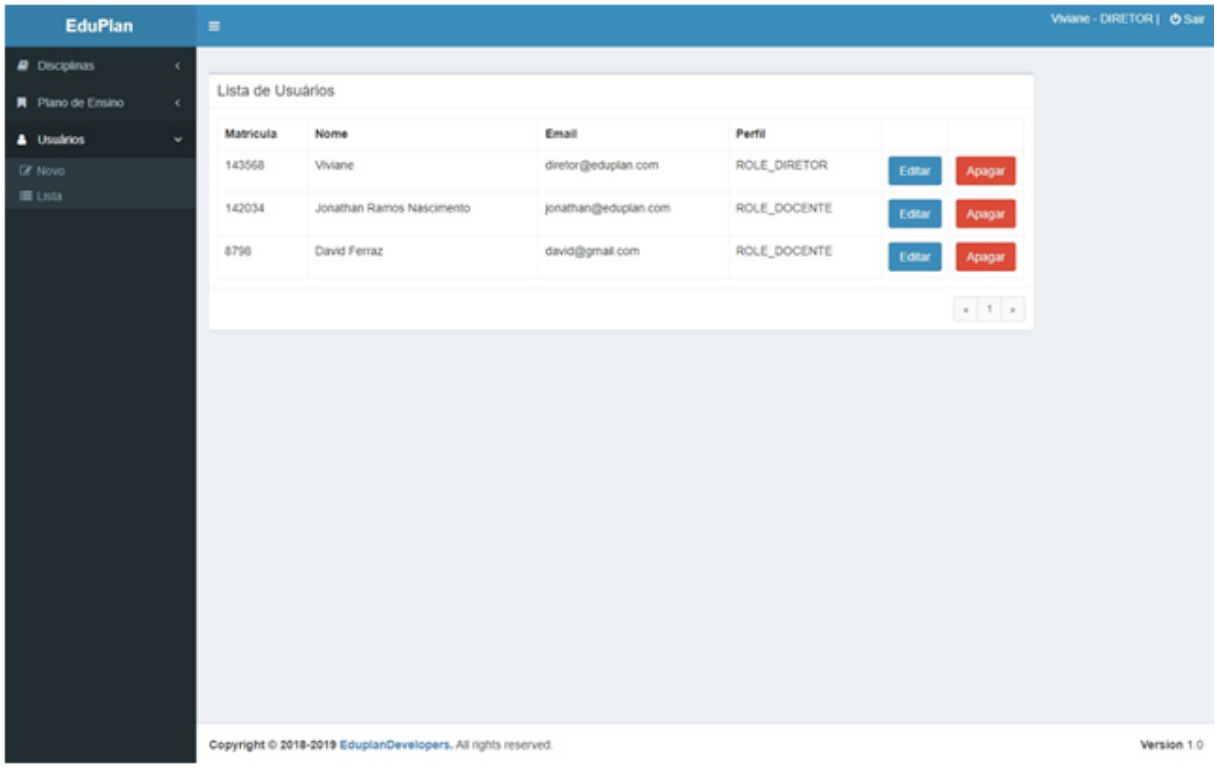


DESCRIÇÃO DE CAMPOS				
Nome do campo	**Tipo do campo**	**Tipo de dado(tamanho)**	**Editável**	**Obrigatório**
Nome	Input Text	Alfabético(100)	Sim	Sim
Matrícula	Input Text	Numérico(10)	Sim	Sim
Email	Input Text	Especiais(80)	Sim	Sim
Senha	Input Text	Alfanumérico(20) com máscara	Sim	Sim
Perfil	Select	Esse campo possui as	Sim	Sim

		seguintes opções: - Diretor -Docente		

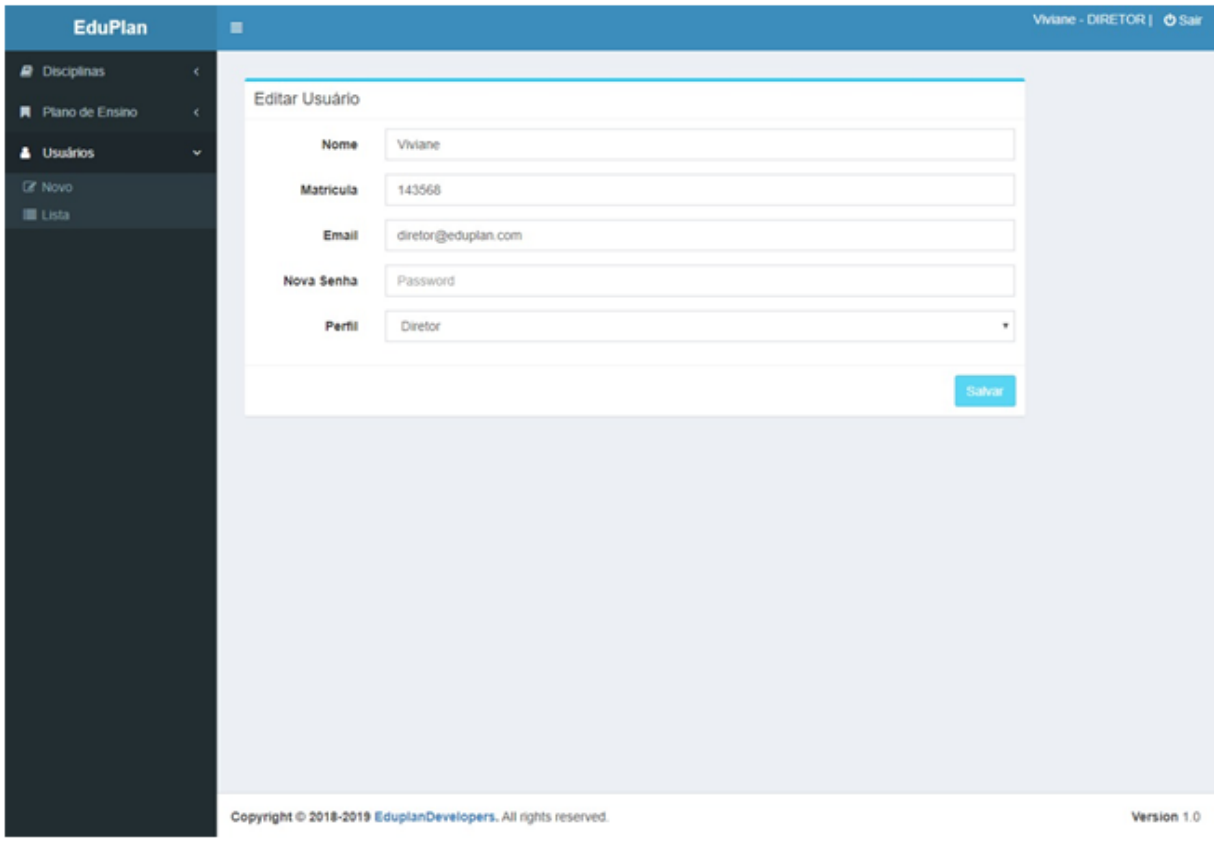
DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Salvar	Salvar as informações na base de dados	Button	Não	Será validado o campo "Matrícula".

Listar Usuários



DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Editar	Direciona para a página de alteração de usuário.	Button	Não	N/A
Apagar	Exclui as informações do usuário da base de dados.	Button	Não	N/A

Editar Usuários



DESCRIÇÃO DE CAMPOS				
Nome do campo	**Tipo do campo**	**Tipo de dado(tamanho)**	**Editável**	**Obrigatório**
Nome	Input Text	Alfabético(100)	Sim	Sim
Matrícula	Input Text	Numérico(10)	Sim	Sim
Email	Input Text	Especiais(80)	Sim	Sim
Senha	Input Text	Alfanumérico(20) com	Sim	Sim

		máscara		
Perfil	Select	Esse campo possui as seguintes opções: - Diretor -Docente	Sim	Sim

DESCRIÇÃO DE COMANDOS				
Nome do comando	**Ação**	**Tipo**	**Tooltip**	**Restrições**
Salvar	Salvar as informações na base de dados	Button	Não	Será validado o campo "Matrícula".