UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Mateus Inacio Rodrigues Damasceno

Manutenção do Sistema Online para Distribuição de Disciplina

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Mateus Inacio Rodrigues Damasceno

Manutenção do Sistema Online para Distribuição de Disciplina

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Bruno Augusto Nassif Travençolo

Coorientador: Jocival Dantas Dias Junior

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2021

Dedico esse trabalho a todos os professores, família e amigos que me apoiaram, principalmente ao meu orientador e minha esposa

Agradecimentos

Agradeço a todos os envolvidos que tornaram esse trabalho possível, meu orientador, meu co-orientador, minha esposa e minha família

Resumo

Assim como qualquer sistema necessita de manutenção, O Sistema Online de Distribuição de Disciplina (SODD), utilizado para a distribuição de disciplinas na Faculdade de Computação na Universidade Federal de Uberlândia, não é diferente e passa pelo mesmo processo. O presente Trabalho de Conclusão de Curso tem por objetivo realizar a correções e implementação de novas funcionalidades no SODD, relatar um histórico de melhorias que foi feito ao longo dos anos, e utilizar a metodologia Kanban para desenvolvimento do projeto. Várias das funcionalidades desenvolvidas estão em uso pelos docentes da FACOM.

Palavras-chave: Distribuição de disciplinas, Sistema online, Atualização de software, Manutenção de software.

Lista de ilustrações

Sumário

1 Introdução

A Faculdade de Computação da Universidade Federal de Uberlândia faz o uso de um sistema online para realizar semestralmente a distribuição de disciplinas para os professores. O sistema é chamado de Sistema Online de Distribuição de Disciplinas (SODD). O processo é conduzidor por uma comissão denominada CDD – Comissão de Distribuição de Disciplinas. Com a implantação do SODD, em 2015, por (LOCATELLI, 2015), veio a necessidade de fazer a manutenção e a implantação de novas funcionalidades no sistema. (LOCATELLI, 2015) desenvolveu um sistema web escalável com padrão de arquitetura MVC, capaz de distribuir as disciplinas dos professores de forma automática satisfazendo todos os critérios dos professores, mas outras tarefas importantes para a distribuição de disciplinas continuavam sendo feitas de forma manual, com o uso de *scripts* em Linguagem de Consulta Estruturada *(Structured Query Language - SQL)*. Isso tornava o sistema ineficiente e sujeito a falhas durante a execução desses scripts. Nos anos seguintes, outros alunos adicionaram novas funcionalidades ao sistema e corrigiram eventuais falhas relatadas pelos professores.

Na engenharia de software, segundo (PRESSMAN, 1995), existem quatro modalidades de manutenção de software: **corretiva**: correção de erros que não foram identificados na fase de teste; **adaptativa:** adaptação do software para suportar as mudanças no ambiente interno, como por exemplos novas regras de negócios; **evolutiva (perfectiva):** melhorias não previstas na documentação, tem como intuito melhorar a qualidade do software, adicionando novas funcionalidades, melhorando desempenho; **preventiva (reengenharia):** alterações do software em que se busca melhorar a confiabilidade ou oferecer melhor estrutura para futuras manutenções.

O processo de evolução de software é impreterível ser constante, com isso, identificou a necessidade de dar continuidade na manutenção do sistema, com o intuito de reduzir as falhas e implementar novas funcionalidades. Utilizando as tecnologias, implementações e ferramentas de gestão de software atuais, o processo evolutivo do SODD teve continuidade.

1.1 Objetivos

Nesta seção serão descritos os objetivos deste trabalho de forma geral e específica.

1.1.1 Objetivo Geral

O objetivo deste projeto é dar continuidade no processo de desenvolvimento do Sistema Online de Distribuição de Disciplinas (SODD), por meio de tarefas de manutenção e implementação novas funcionalidades.

1.1.2 Objetivos Específicos

A partir do objetivo geral, podemos enumerar os objetivos específicos:

- Corrigir eventuais falhas do sistema, relatadas pelos usuários ou identificadas pelos desenvolvedores.
- Adicionar novas funcionalidades ao sistema, de acordo com as necessidades da CDD e sugestões dos usuários.
- Introduzir uma nova forma de condução do projeto, com o uso da metodologia Kanban na plataforma *online* Trello[1].

1.2 Organização do Trabalho

Este trabalho está estruturado da seguinte forma. No Capitulo 2 são apresentados o histórico do SODD e as tecnologias utilizadas. No Capítulo 3 é apresentada a metodologia e as tarefas implementadas. No Capítulo 4 é apresentada a Conclusão.

[1] Trello.com

2 Fundamentação Teórica

Este capítulo está dividido em dois principais tópicos. O primeiro relata um histórico de todo o desenvolvimento do CDD. O segundo apresenta uma breve descrição das metodologias e tecnologias utilizadas para o desenvolvimento do sistema.

2.1 Histórico do CDD

Nesta seção será descrito o histórico das principais implementações do sistema SODD.

2.1.1 Primeira entrega

O desenvolvimento do SODD teve inicio no ano de 2015, conduzido por (LOCATELLI, 2015), em que verificou-se a necessidade de automatizar o processo de distribuição de disciplinas visto que era feito a partir de uma Comissão de Distribuição de Disciplinas (CDD). Segundo (LOCATELLI, 2015):

> *O trabalho da comissão é relativamente complexo, uma vez que são ofertadas por volta de 150 turmas de disciplinas, totalizando aproximadamente 600 horas-aula. Essas turmas são distribuídas entre aproximadamente 70 professores, sendo que a carga horária atribuída a cada professor é distinta, pois depende de fatores como o regime de trabalho do docente (e.g., Dedicação Exclusiva, 20h semanais, professor substituto) e cargos administrativos ou acadêmicos.*

Dado a complexidade do processo e da forma de como era feito, (LOCATELLI, 2015) fez o levantamento dos requisitos funcionais e não funcionais para implementar um sistema web que auxiliasse os professores e a FACOM na distribuição de disciplinas com funções de acesso as informações da disciplina, horários, turmas e preferência dos professores.

A aplicação teve as seguintes funcionalidades implementadas:

- Verificar horários das disciplinas
- Exibir fila por disciplina
- Exibir fila por professor (sem as turmas)

- Exibir fila por turma
- Exibir fila por professor (com as turmas)

2.1.2 Segunda Entrega

No ano de 2016, (SILVA, 2016) e (NAVES, 2016) assumiram a manutenção do projeto, criando um módulo administrativo, um módulo de relatórios, melhoraram a interface, usabilidade e tornaram o sistema responsivo.

Nesse ano foi criado um repositório *git* de controle de versões, para que ambos os envolvidos conseguissem trabalhar de forma simultânea e independentes, além de ser possível o controle do progresso das tarefas pelo orientador.

Enquanto (SILVA, 2016) implementava o módulo administrador e novas funcionalidades, (NAVES, 2016) assumia a função de aprimorar a interface, melhorar a usabilidade e torná-lo responsivo.

Abaixo as implementações feitas durante esse período.

- Implementações:
  - Configuração de Disciplinas
  - Configuração de Prioridades
  - Configurações de Turmas
  - Configuração da atribuição de um professor a uma disciplina
  - Configuração dos horários da turma
  - Configuração de Cursos
  - Configuração da Fila de professores da disciplina
  - Configuração de Professores
  - Configurações de Semestres
  - Configurar as Restrições
- Relatórios implementados:
  - Relatório de distribuição geral.
  - Relatório de filas geral.
- Melhorias visuais:
  - Ajuste do *grid* de listagem das disciplinas.

- Adequação do sistema de alerta quando há conflito de horários nas disciplinas selecionadas.
- Visualização das informações sobre impedimentos de um professor.
- Implementação do componente de *Typeahead*.

2.1.3 Terceira Entrega

O trabalho de (OLIVEIRA, 2017) continuou nos mesmos moldes dos trabalhos anteriores, fazendo o levantamento de requisitos e manutenções eletivas, corretivas e evolutivas. Durante o período que (OLIVEIRA, 2017) ficou responsável pela aplicação, as seguintes funcionalidades foram implementadas.

- Criar e editar disciplina.
- Duplicar turmas.
- Cadastrar oferta de turma.
- Exibir nome da disciplina na aba listagem de turma.
- Alterar os campos de edição ano e semestre para combo box.
- Criar e editar professor.
- Implementado algoritmo de fila de disciplinas.
- Login, verificar se usuário existe.
- Alterar campo de busca aberto de edição para *tag input*.
- Alterar nome do Menu para “Atribuição de turmas aos professores”.
- Inserir na aba “Exibir turmas por professor” campo de busca por nome.
- Melhorias na fila professor e correção de erros.
- Alterar, em prioridade, o campo de busca aberto para *tag input*.
- Corrigir erro de codificação dos campos em cursos.
- Permitir alterar o código do curso.
- Inserir campo campus do curso.
- Permitir alterar o código da disciplina.
- Permitir alterar o curso da disciplina.

- Inserir o campo do código antigo da disciplina.
- Melhorado a forma que cadastra um restrição.
- Exibir horários das disciplinas de um professor.
- Permitir que o docente indique os horário em que não pode participar de atividades.
- Criação de relatório de turmas.
- Criação de carga horária por curso.
- Criação de relatório filas não inscritas por docente.

2.1.4 Quarta Entrega

Em 2018, (SANTOS, 2018) ingressou no projeto dando continuidade, implementando novas funcionalidades, levantando novos requisitos e corrigindo eventuais *bugs* do sistema. Com o trabalho de (SANTOS, 2018), o sistema teve as seguintes funcionalidades implementadas.

- Funcionalidades
  - Visualização de filas de disciplinas e turmas.
  - Visualização de filas de um professor.
  - Visualização de horários.
  - Relatório de distribuição de turmas.
  - Indicação de processamento em andamento.
  - Duplicação de turmas.
  - Algoritmo de rotação de filas.
  - Status de professores.
  - Cadastramento de novo semestre
- Erros corrigidos
  - Remoção de turmas.
  - Consulta aos horários de disciplinas
  - Gerenciamento de prioridades

2.1.5 Quinta Entrega

Em 2018, (OLIVEIRA, 2018) ficou encarregado de fazer o módulo de distribuição das disciplinas. Esse módulo, diferente de todos os outros trabalhos prévios, que basicamente gerenciavam informações relacionadas aos dados armazenados no sistema, implementou um algoritmo que realiza a distribuição das disciplinas. O processo de distribuição é interativo, e permite a criação e gerenciamento de cenários e executar a distribuição para cada um deles, apresentando os resultados à comissão, que decidirá qual dos cenários será adotado. Diferente dos demais módulos, feitos em Java, o módulo de distribuição foi escrito em C# e integrado ao sistema.

O sistema desenvolvido por (OLIVEIRA, 2018) tem as seguintes características

- Resolver *deadlocks* no processo de distribuição.
- Possui interface gráfica para atribuição de disciplinas de forma manual.
- Verifica se a carga horária atribuída ao professor está maior ou menor que a prevista.
- Permite testar diferente cenários de distribuição, considerando diferentes cargas de disciplinas para professores.

2.2 Tecnologias utilizadas

2.2.1 Arquitetura MVC

A arquitetura MVC é um padrão de arquitetura de *software* cujo o princípio básico é construir aplicações separando seus principais componentes. Para (ERICH et al., 2000), a abordagem da arquitetura MVC é composta por três tipos de objetos. O Modelo é o objeto de aplicação, a Visão é a apresentação na tela e o Controlador é o que define a maneira como a interface do usuário reage às entradas do mesmo. Antes da arquitetura MVC, os projetos de interface para o usuário tendiam a agrupar esses objetos. A arquitetura MVC separa esses objetos para aumentar a flexibilidade e a reutilização.

A camada Modelo é a ponte entre as demais camadas, somente na camada modelo é que ocorre as operações de inserção, remoção, edição e atualização, ou em inglês a sigla CRUD (*Create*, *Read*, *Update* e *Delete*). A camada modelo consiste na parte lógica da aplicação cujas função é gerenciar as regras de negócios, lógicas e funções.

A camada visão é responsável por apresentar as informações de forma visual ao usuário. Em seu desenvolvimento devem ser aplicados apenas recursos ligados a aparência como mensagens, botões ou telas.

A camada controle é responsável por intermediar a comunicação entre as outras camadas, ela recebe os dados da camada visão, trata eles para a camada modelo poder

processar, depois de processado, a camada modelo envia os dados para a camada controle, que repassa para a camada visão.

2.2.2 JAVA

Java é uma linguagem de programação altamente difundida na comunidade. Em 2015 essa foi a linguagem escolhida para fazer a aplicação SODD. Segundo o índice *TIOBE*[1], que é um indicador da popularidade das linguagens de programação, Java foi a linguagem do ano em 2015 e se manteve entre os primeiros colocados nesse índice desde então.

2.2.2.1 JVM (*Java Virtual Machine*)

O poder do JAVA está em sua JVM (*java virtual machine*), que permite rodar as aplicações independente do sistema operacional instalado e, devido à gama de dispositivos que suportam java, como celulares, televisões, geladeiras, entre outros, torna o java extremamente poderoso.

Linguagens como C, dependem do sistema operacional para compilar, ou seja, para transformar os arquivos .c em executáveis binários para serem interpretados pelo SO, com isso fica restrito os programas para aqueles sistemas operacionais cuja a aplicação foi desenvolvida.

A linguagem java, utiliza o conceito de maquina virtual, adicionando uma camada extra entre a aplicação e o sistema operacional. A JVM, maquina virtual java, recebe o *bytecode* gerado por um compilador java, javac por exemplo, e traduz esses *bytecodes* para o sistema operacional em questão fazer as chamadas de sistema que a aplicação requisitou.

2.2.2.2 JRE (*java Runtime Enviroment)*

O JRE *Java runtime enviroment* é um conjunto de componentes para criar e executar aplicações Java. Ele está incluído no kit de desenvolvimento Java (JDK). O JRE combina código Java criado usando o JDK com as bibliotecas necessárias para executá-lo em uma JVM e, em seguida, cria uma instância da JVM que executa o programa resultante.

O JDK e o JRE interagem entre si para criar um ambiente de tempo de execução sustentável que permite executar aplicativos com base em Java sem interrupções. Seus componentes da arquitetura de tempo de execução são:

[1] https://www.tiobe.com/tiobe-index

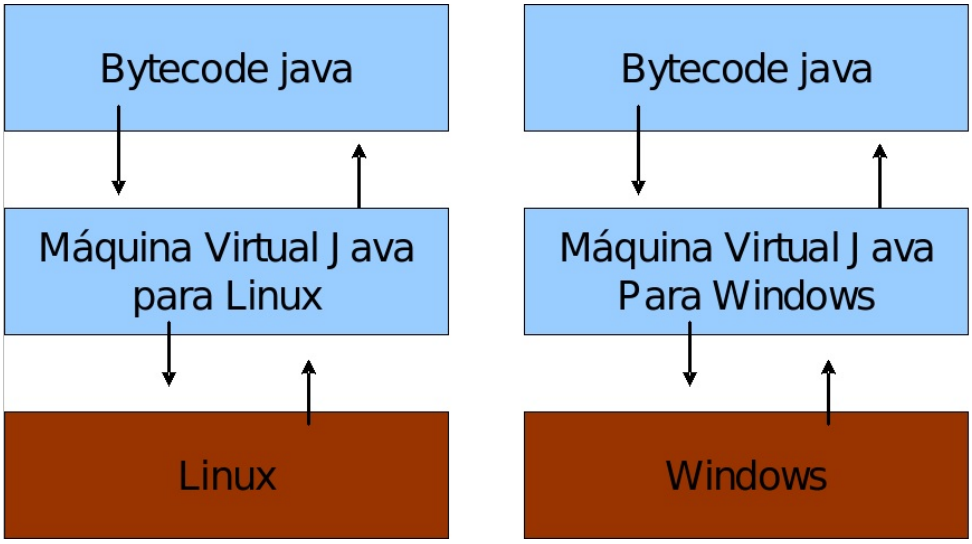

Figura 1 – Diagrama de compilação JVM.

2.2.2.2.1 ClassLoader

O Java *ClassLoader* carrega dinamicamente todas as classes necessárias para executar um programa Java. Como as classes Java são apenas carregadas na memória quando necessárias, o JRE usa ClassLoaders para automatizar este processo sob demanda.

2.2.2.2.2 Verificador de bytecode

O verificador de bytecode garante o formato e a precisão do código Java antes de passar para o interpretador. No caso de o código violar a integridade do sistema ou os direitos de acesso, a classe será considerada corrompida e não será carregada.

2.2.2.2.3 Interpretador

Após o *bytecode* carregar com sucesso, o interpretador Java cria uma instância da JVM que permite que o programa Java seja executado nativamente na máquina subjacente.

2.2.3 *Front-end*

Para o desenvolvimento do *Front-end* foi utilizado Javascript com o *framework* estrutural de código aberto AngularJS [2], mantido pelo Google, e um *framework* de CSS e javascript, o *Bootstrap*[3], criado por funcionários do Twitter e mantido atualmente pela comunidade. O Bootstrap é voltado para a estilização de páginas e novas funcionalidades como menus de navegação, controles de paginação, formulários, janelas modais, enquanto

[2] https://angularjs.org/
[3] https://getbootstrap.com/

o AngularJS facilita a manutenção de layouts complexos e possibilita o isolamento de responsabilidades de cada recurso da aplicação.

2.2.4 PostgreSQL

O PostgreSQL foi escolhido por ser um gerenciador de banco de dados relacional *opensource*, com uma forte reputação por sua arquitetura comprovada, confiabilidade, integridade de dados, conjunto de recursos robustos, extensibilidade e dedicação da comunidade de código aberto por trás do software.

2.2.5 Maven

Maven é uma ferramenta de gerenciamento e compreensão de projetos de software. Baseado em o conceito de um modelo de objeto de projeto (POM), Maven pode gerenciar um projeto construir, relatar e documentar a partir de uma peça central de informação. Com o Maven, é possível automatizar o *download* de *frameworks* requeridos, organizar as dependências e verificar/atualizar suas versões.

2.2.6 GIT

GIT é um sistemas de controle de versão distribuído *open-source*, com ele é possível controlar o fluxo de novas funcionalidades entre vários desenvolvedores no mesmo projeto com ferramentas para análise e resolução de conflitos quando o mesmo arquivo é editado por mais de uma pessoa em funcionalidades diferentes.

2.2.7 IDE - IntelliJ

IntelliJ foi a IDE escolhida para o ambiente de desenvolvimento por possuir uma variedades de benefícios que ajudam o programador, como o recurso de preenchimento de código inteligente, assistência específica para cada estrutura, impulsionadores de produtividade, ergonomia e assistente de codificação inteligente. O IntelliJ possui recursos com integração com o banco de dados e servidores web, além de fácil criação do *deploy*, arquivo para rodar a aplicação no servidor.

3 Desenvolvimento

Neste capítulo serão apresentadas a metologia de trabalho adotada para desenvolvimento do trabalho e o detalhamento de cada tarefa executada.

3.1 Metodologia Adotada

Para a realização desse trabalho foi aplicado um método chamado Kanban utilizado em gestão de projetos, que consiste em limitar o trabalho em progresso, apresentando de forma visual a progressão das atividades, tornando problemas evidentes e cultivando a cultura de melhoria contínua.

O Kanban, assim como outras metodologias ágeis, fornecem transparência sobre as atividades em andamento e concluídas e reportam métricas de velocidade, mas o Kanban, além da transparência ao processo e seu fluxo, fica evidente gargalos, filas, variabilidade e desperdícios, ou seja, tudo que impacta no desempenho da equipe.

Para auxiliar com o Kanban, foi utilizado uma plataforma de gerenciamento de processos, Trello.com, em que as tarefas foram adicionadas com o status de "À fazer", quando se executava uma tarefa, o status dessa mudava para "Em correção", depois de concluída mudava para o status "Realizar escrita". O status "refinar correção/Tratar Erro" foi usado para caso identificasse outro problema durante a tarefa em correção e, por último, o status concluído.

Utilizar a ferramenta Trello foi um grande avanço para organização do projeto pois permitiu descrever de forma bem mais detalhada as necessidades do sistema. Antes, no início do projeto, o controle de erros e requisitos era um grande arquivo texto com descrições breves sobre as funcionalidades, o que muitas vezes dificultava o entendimento. Com o Trello os requisitos ficaram bem mais organizados. Para cada um é possível agora inserir imagens, vídeos e rótulos. Isso facilitou de gerenciamento de tarefas e o compartilhamento de informações entre membros da equipe. A Figura 2 mostra um cópia da tela do Trello utilizada no projeto.

3.2 Tarefas implementadas

Nesta seção são apresentadas as tarefas implementadas com a descrição do problema e solução encontrada. Para facilitar a descrição das tarefas, foi adotado como título das subseções os nomes originais da tarefas no Trello. A numeração dos itens das tarefas – apresentados como *Task* (tarefa em inglês) seguido de um número – não obedece uma

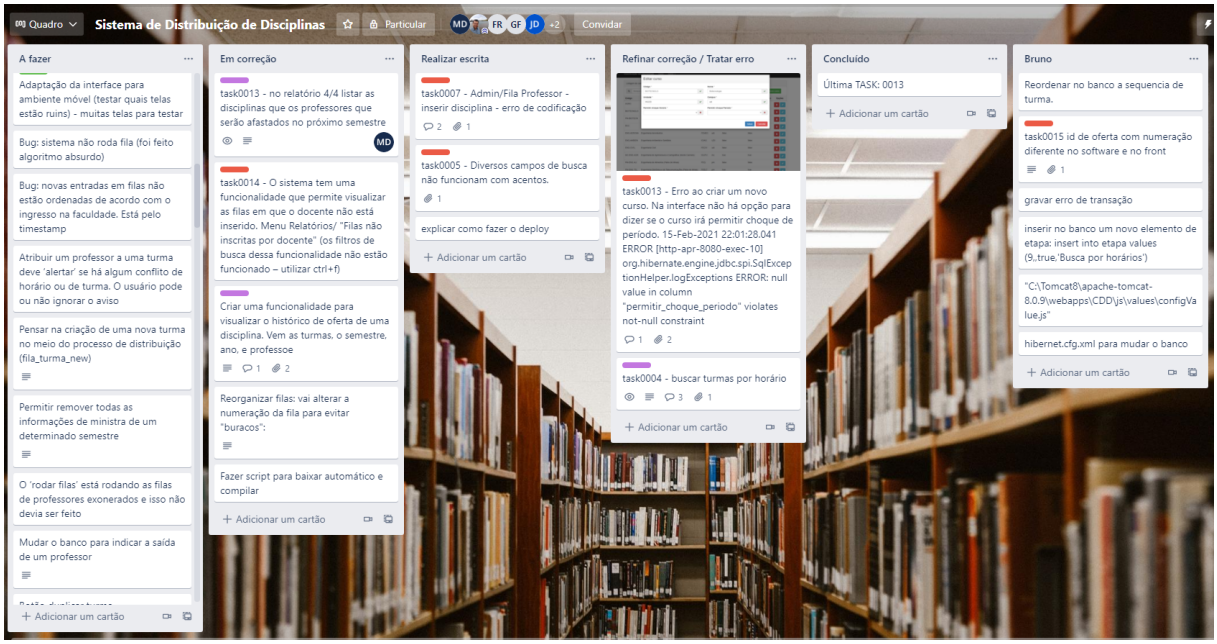

Figura 2 – Tela de trabalho do Trello utilizada para acompanhamento das tarefas (*task*) elencadas para o sistema.

ordem, visto que as tarefas são lançadas no Trello conforme a necessidade, e, nem sempre, essas tarefas são executadas na mesma ordem em que foram lançadas.

3.2.1 Task005 - Acentuação

Problema: Diversos campos de busca não funcionam adequadamente com palavras que possuem acentuação.

Os usuários estavam com dificuldades de buscar palavras com acentos. Por exemplo, se buscasse a palavra "programacao" no campo de disciplina, não encontrava, mas se digitasse "programação" o sistema retornava as disciplinas que tinham programação no nome. Apesar disso não ser um erro, permitir a busca sem utilizar acentuação é uma funcionalidade esperada em qualquer sistema atualmente.

Como solução, foi implementado uma máscara de entrada em todos os campos de busca. Já que existem máscaras prontas para uso, foi utilizada no sistema o contêiner *normalize-diacritics*[1]. Com a aplicação da máscara, as mesmas palavras com acentuação e sem acentuação passaram a ser reconhecidas pelo sistema.

3.2.2 Task0013 - Choque de Período

Problema: Erro ao criar um novo curso. Na interface não há opção para dizer se o curso irá permitir choque de período.

[1] https://www.npmjs.com/package/normalize-diacritics

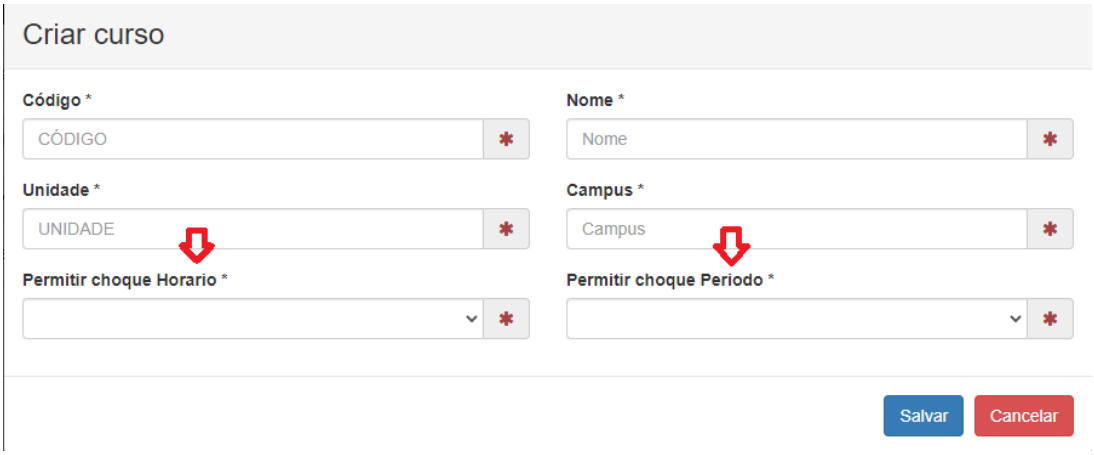

Figura 3 – Adição de novos campos para previsão de choque de período e de horário na funcionalidade “Novo Curso”.

Os usuários não estavam conseguindo criar um novo curso, não era possível inserir as opções de “Permitir choque de Horário” e “Permitir choque de Período”, informações obrigatórias para o sistema. Por causa disso o sistema gerava um erro de violação de restrição no banco de dados e não inseria o curso.

A resolução do problema foi adicionar, na parte do *frontend*, uma caixa de listagem com as opções de “Sim” e “Não” para as opções “Permitir Choque de Período” e “Permitir Choque de Horário” (Figura 3). Depois de adicionada as caixas de listagens no *FrontEnd*, foi necessário, mapear no *backend*, os campos novos, adicionando no DTO os atributos permitir_choque_periodo e permitir_choque_horario do tipo boleano e na camada *repository*, os campos permitir_choque_periodo e permitir_choque_horario foram adicionados para compor a transição com o banco na inserção de novos cursos. Por fim, os campos permitir_choque_periodo e permitir_choque_horario agora são listados quando mostra todos os cursos.

3.2.3 Task004 - Buscar Turma Por Horário

Problema: Necessário uma nova implementação para fazer a busca por horário das disciplinas.

Essa nova funcionalidade se mostrou necessária após várias solicitações dos usuários que precisavam saber em quais dias e horários as turmas seriam ministradas com o intuito de poder organizar sua grade horária e dar prioridade para as matérias que tem mais interesse naquele semestre. Por exemplo, um professor necessita saber quais disciplinas seriam ministradas na quinta-feira no período da manhã, das 7:10h as 12:20h. Até então, o sistema mostrava os dias e horários das disciplinas, mas o usuário tinha que abrir cada disciplina e verificar ele mesmo o dia e horário, aumentando a chances de erros.

Foi implementado um módulo de busca por horário de disciplinas como resolução

dessa funcionalidade, adicionando no sistema uma consulta SQL na camada *repository*, um DTO na camada *view*, criado um *endpoint* na camada *controller* que retorna o DTO criado, além da implementação de *frontend* em *javascript*.

Para criar uma consulta SQL na camada *repository*, foi necessário criar um método chamado "listaTodasTurmasProfessorPorHorario" na classe "TurmaRepositoryImpl" e utilizar as tabelas turma, oferta, disciplina, fila_turma com os atributos codigo, nome, dia, letra e turma do banco de dados para conseguir na consulta SQL os dados necessários para o requisito.

Na camada *view* foi criado um DTO ListaTodasTurmasProfessorPorHorarioDTO que contém os atributos que retornam da busca da camada *repository* (codigo, nome, dia, letra e turma) e criado os métodos padrões da programação orientada a objetos.

Para a criação do *endpoint* na camada *controller*, foi implementado um método "listaTodasTurmasProfessorPorHorario" na classe "TurmaControllerImpl", em que os dados da camada *view* são recebidos e o *controller* mapeia essas informações e repassa para as outras camadas do modelo.

Na parte de *frontend* foi adicionado mais uma opção na pagina principal, com uma tabela em que é possível selecionar os horários em que se tem interesse em ministrar aulas, na Figura 4 mostra a tela de interação da nova funcionalidade implementada.

3.2.4 Task0014 - Funcionalidade dos filtros de busca

Problema: Na seção de relatórios a busca não funciona corretamente.

Uma funcionalidade do sistema, na sessão de relatórios, é exibir a fila dos professores cujo os professores não selecionaram disciplinas para ministrar.

O sistema não estava totalmente funcional, pois o filtro dos campos não funcionavam, investigando o problema, notou-se que um *div*, estrutura do HTML, estava impedindo do filtro funcionar, feito a devida correção, o problema foi sanado.

3.2.5 Task015 - Anomalia no ID de oferta

Problema: Os identificadores de oferta não estavam seguindo uma sequência.

Esse problema, até então, não estava causando grandes impactos, mas estava havendo conflitos de chaves durante a execução do sistema e futuramente causaria estouro de chave primária ou mesmo violação da restrição de chave.

Esse problema ocorria quando executava o sistema, a aplicação criava um ID na tabela oferta e quando ia criar uma oferta, o banco de dados criava um outro ID. Dessa forma, toda vez que o sistema era iniciado estavam sendo gerando IDs de oferta sem necessidade.

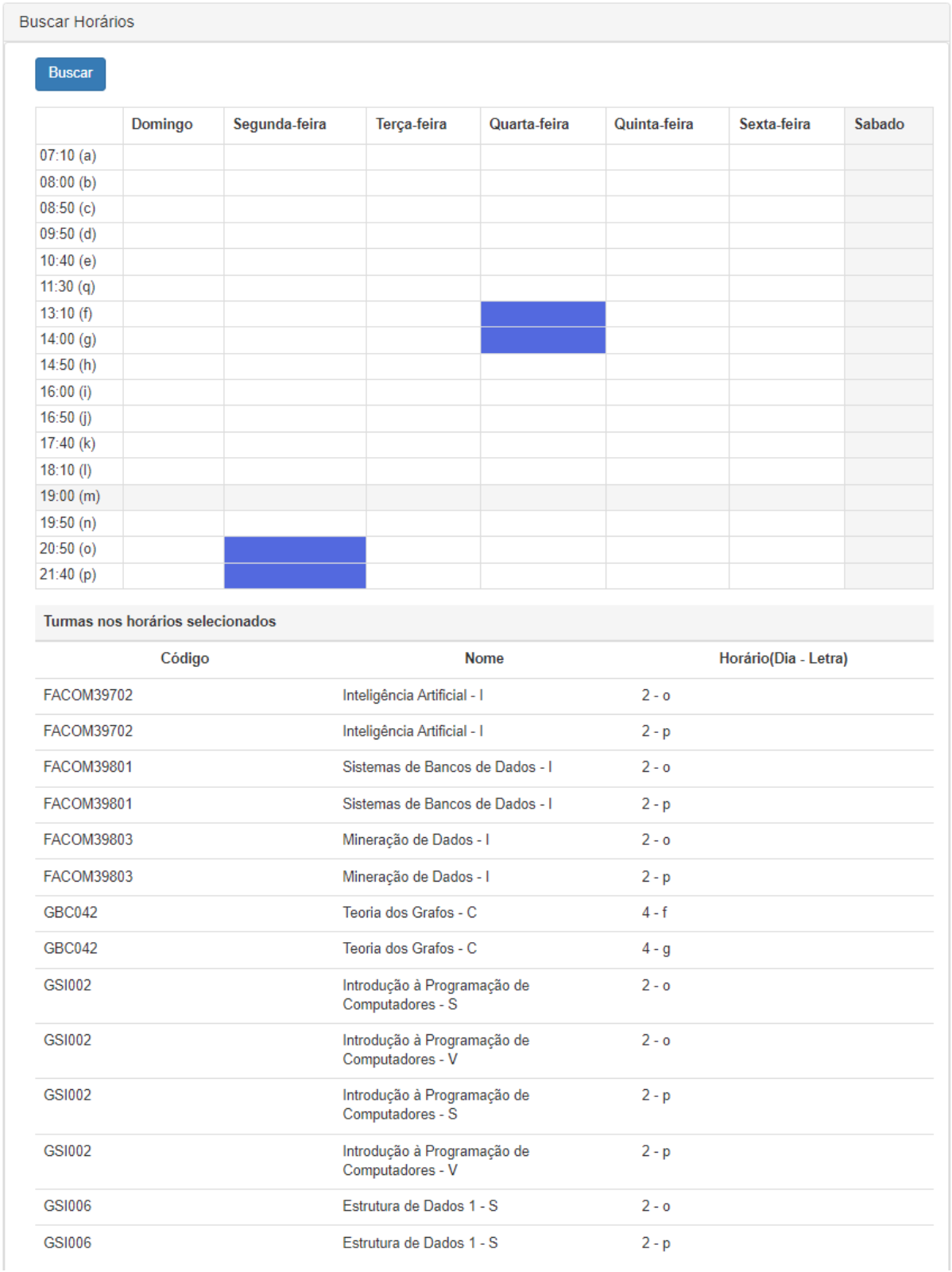

Figura 4 – Nova funcionalidade que permite ao usuário selecionar no quadro de horários os horários que deseja ministrar uma disciplina. O sistema retorna todas as turmas que são ministradas nos horários indicados.

A solução foi deixar para o banco de dados o gerenciamento de IDs de oferta, retirar da aplicação a criação de IDs e atualizar no banco de dados os IDs de oferta por meio de um *script* no banco.

4 Considerações Finais

É incontestável que a manutenção de *software* é indispensável para sua melhoria e otimização. Este trabalho teve foco na manutenção corretiva e evolutiva, corrigindo erros relatados e implementando novas funcionalidades.

Como o SODD começou em 2015, houve muitas melhorias ao longo dos anos e várias ferramentas foram incorporadas durante toda a vida do SODD, vale destacar o uso do GIT, em 2016, quando (SILVA, 2016) e (NAVES, 2016) trabalharam juntos na aplicação. Em 2018, (OLIVEIRA, 2018) trabalhou com um modulo independente em outra linguagem e fez a integração com a aplicação principal e, em 2021, a proposta deste projeto incluiu o uso do Kanban para distribuir as tarefas. As ferramentas agregadas ao projeto continuam em uso e auxiliam os integrantes do projeto nas tomadas de decisões.

A principal dificuldade encontrada durante este projeto foi preparar e configurar o ambiente local. Como o sistema tem mais de 5 anos, algumas tecnologias ficaram datadas, como por exemplo, o uso do servidor de páginas. Nesse sistema, o servidor é configurado manualmente, sendo que nos dias atuais utiliza-se um *framework* que automatiza todo esse processo do servidor. Outro problema encontrado neste projeto foi descobrir como o sistema estava organizado, mesmo seguindo o padrão de arquitetura MVC, em alguns pontos ele não estava bem definido – uma prova disso é o arquivo *main.js* que possui 1847 linhas de código, isso acarreta no desempenho da aplicação. Além disso, entender os requisitos do sistema estava complicado, pois era só um arquivo de texto comum, não existia controle ou categorização, a partir do momento que passou a utilizar a ferramenta Trello.com, os requisitos foram melhor divididos e organizados.

Para os trabalhos futuros, está a eliminação de *stored procedures*. Elas foram implementadas quando o sistema ainda não estava completo, e muitas consultas ainda eram feitas direto no banco de dados. Com o crescimento da aplicação, perdeu-se a necessidade de funções dentro do banco de dados, mas algumas das *stored procedures* ainda desempenham um papel importante no sistema. Outra melhoria é a automatização do *deploy* por meio de *script*, facilitando a disponibilização de atualizações do sistema.

Referências

ERICH, G. et al. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos. Trad. Luiz AM Salgado*. [S.l.]: Bookman, 2000. Citado na página 14.

LOCATELLI, J. A. *Sistema Online para Distribuição de Disciplinas*. 2015. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação – Universidade Federal de Uberlândia, Uberlândia). Citado 2 vezes nas páginas 8 e 10.

NAVES, P. E. de P. *Novas Funcionalidades para o Sistema Online de Distribuição de Disciplinas entre docentes*. 2016. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação – Universidade Federal de Uberlândia, Uberlândia). Citado 2 vezes nas páginas 11 e 24.

OLIVEIRA, A. S. de. *Novas funcionalidades e manutenção no sistema online de distribuição de disciplinas*. 2017. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação – Universidade Federal de Uberlândia, Uberlândia). Disponível em: <https://repositorio.ufu.br/handle/123456789/19437>. Citado na página 12.

OLIVEIRA, I. A. G. de. *Implementação de um Sistema para Distribuição de Disciplinas*. 2018. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação – Universidade Federal de Uberlândia, Uberlândia). Citado 2 vezes nas páginas 14 e 24.

PRESSMAN, R. S. *Engenharia de Software. Terceira edição*. [S.l.]: São Paulo: Pearson Makron Books, 1995. Citado na página 8.

SANTOS, A. V. dos. *Sistema online de distribuição de disciplinas: Manutenção e implementação de novos requisitos*. 2018. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação – Universidade Federal de Uberlândia, Uberlândia). Disponível em: <https://repositorio.ufu.br/handle/123456789/22411>. Citado na página 13.

SILVA, S. S. da. *Implementação de Novos Requisitos para o Sistema Online para Distribuição de Disciplinas*. 2016. Implementação de Novos Requisitos para o Sistema Online para Distribuição de Disciplinas. Citado 2 vezes nas páginas 11 e 24.