Calendário Unificado
Relatório da Primeira Entrega

Guilherme Varela, Karen Pacheco, Larissa Bral, Leonardo Costa

[1]Programação Avançada – Universidade Federal do Rio de Janeiro (UFRJ)

1. Introdução

Ao cursar as disciplinas de graduação da UFRJ os alunos são submetidos a diferentes regimes de avaliação: trabalhos semanais, trabalhos esporádicos, quizes online, provas, entre outros métodos ou uma combinação destes. Alguns desses métodos requerem mais estudo prévio e preparo do que outros, com isso, um bom planejamento e boa organização é essencial. Além disso, é importante que o aluno tenha uma grade de disciplinas balanceadas em número e dificuldade, para que não tenha uma sobrecarga de tarefas ao longo do período, pois muitas disciplinas e diciplinas difíceis demandam mais tempo de estudo.

Mesmo assim, na maioria dos perídos letivos as datas críticas de diferentes disciplinas tendem a coincidir, em especial no meio e fim do semestre, acumulando diversas avaliações na mesma semana, podendo haver mais de uma avaliação por dia. Esse cenário é muito comum a vários estudantes, e fica ainda mais caótico quando o aluno não se organiza e planeja os estudos para não deixar a matéria acumular, o que acontece mais do que gostaríamos.

Tendo em vista que não existe ferramenta ou metologia para aliviar esse conflito, de forma a auxiliar cada professor a escolher a data ideal para suas avaliações dentro das limitações do calendário acadêmico aprovado pelo CEG e que esteja equilibrada com as demais provas dos alunos que cursam a disciplina, foi pensada uma solução para minimizar essas dores.

Como solução para esse problema, elaboramos o Calendário Unificado, uma plataforma onde os professores podem cadastrar provas e entregas/apresentações de trabalhos extensos e verificar dias e semanas que podem sobrecarregar o corpo discente de um determinado curso ou combinação de cursos, enquanto o aluno pode se inscrever nas disciplinas que está cursando e ter disponível um calendário com todas as provas das matérias inscritas.

2. Arquitetura

Após descrever o problema e sugir com uma solução que se adeque, a primeira parte para planejamento e visualização da solução foi simular as telas e fluxo de uso do sistema, criando também uma identidade visual para a aplicação. Com isso, conseguimos mapear as histórias (story map) e então, a partir disso, construir um diagrama UML isolando e relacionando as entidades necessárias. Essas tarefas foram realizadas utilizando a ferramenta Figma, o design das telas pode ser visualizado aqui: https://www.figma.com/file/uyGa48mqOePwbM5q0RvzzC/Untitled?type=design e o mapeamento do banco pode ser visualizado aqui: https://www.figma.com/file/9217M6dGJZEE7trXskBC4t/Untitled?type=whiteboardnode-id=0-1.

Para organização do fluxo de desenvolvimento do projeto, a equipe foi subdividida tal que uns ficassem responsáveis pelo desenvolvimento *backend* da aplicação e outros pelo *frontend* da aplicação. Dessa forma, as partes da aplicação foram desenvolvidas de forma concorrente, para ao final serem unificadas.

O *backend* é responsável por toda parte de persistência de dados, verificações de segurança, validação do fluxo de dados, modelagem das entidades e prover endpoints para o *frontend*, que por sua vez, produz as telas de interação com o usuário, contendo formulários e botões que realizam requisições para o servidor *backend*. De modo grosseiro, o *frontend* fica responsável pelo cliente e o *backend* pelo servidor numa arquitetura cliente-servidor.

No lado do cliente, foram construídas as representações das telas necessárias para login, cadastro de usuário, cadastro de disciplinas e provas, e visualização do calendário. Por parte do servidor, foi utilizada uma arquitetura MVC, que consiste num Modelo(M), responsável pelas regras de negócio e formatação das entidades que serão persistidas no banco de dados, num Controlador(C), que é responsável por tratar as requisições vindas do cliente e enviar de volta as respectivas respostas, bem como passar para o modelo quaisquer fluxo de dados que deva ser persistido no banco, e numa View(V), que seria a parte de visualização que é totalmente relegada ao cliente.

O fluxo de utilização começa numa tela de login onde o usuário pode iniciar uma nova sessão. O usuário pode ter um cadastro com perfil de professor ou perfil de aluno, cada um com funcionalidades específicas no sistema. Uma vez iniciada a sessão o menu é exibido com as opções de acessar as disciplinas, para remover, adicionar ou visuallizar, e de acessar o calendário de provas, para o caso de perfil aluno. Para o caso de perfil professor, o menu inicial exibe as opções de calendário e disciplinas, sendo a primeira o calendário com as provas já marcadas do curso e a segunda oferece opções de cadastra, editar ou excluir disciplina e marcar prova.

3. Implementação

Para o cliente, foi escolhida a biblioteca *VUE.JS*, *framework* voltado para prototipagem rápida, oferecendo uma maneira fácil e flexível de ligação de dados reativos e componentes reutilizáveis, em que consiste numa aplicação javascript composta de componentes, páginas e demais recursos, reativa e, apesar de minimalista, tem uma estrutura definida com arquivos e diretórios-chave formando uma mini-arquitetura pronta para a parte da View. Repositório com a parte *frontend* da aplicação: https://github.com/srtapacheco/calendario-unificado.

Para o lado do servidor, foi escolhido *NestJS*, *framework* que auxilia no desenvolvimento de uma aplicação eficiente, escalável e confiável em cima de Node.js e também tem uma estrutura interna bem definida: o código fonte de uma aplicação Nest é formado por um conjunto de classes internas implementadas via anotações e uma estrutura de diretórios que contém databases(onde ficam as chamadas para o banco de dados, que no caso foi escolhida a ferramenta Prisma para auxiliar no gerenciamento do mesmo), middlewares(onde ficam validadores de requisições, em específico para autenticação) e modules(onde cada entidade tem um subdiretório prório onde são descritas com três arquivos no mínimo: services, module, controller) que já correspondem às partes ”M” e ”C” do MVC. A integração entre o controlador e a view é feita através de requisições

HTTP e suas respostas, respeitando as limitações REST. Repositório com a parte *backend* da aplicação: https://github.com/gui-varela/calendario-unificado.

Para mapeamento das entidades e manipulação dos dados utilizamos *PostgreSQL* como banco de dados, por ser bem completo e de fácil acesso, com a ferramenta de gerenciamento Prisma, que com uma rápida configuração é possível integrar a diversos bancos de dados *SQLike*.

Como futuramente a aplicação será alocada para produção na nuvem, a conteinerização já foi inicializada, uma *DockerFile* já foi criada com esse intuito e os arquivos README dos repositórios contém as instruções de instalação e configuração inicial.

4. Extensões e Refinamento

Para produzir o protótipo da aplicação, o escopo de dados foi reduzido a fim de simplificar o desenvolvimento e testes do projeto. Para próximas versões e atualizações, funcionalidades extras serão implementadas e as existentes serão aprimoradas. Poderá ser possível criar mais perfis de acesso e conexão com outros sistemas institucionas, como o SIGA, por exemplo, para automatizar transferências de dados dos alunos e professores já existentes.

Como ideias e sugestões para futuras funcionalidades, gostaríamos integrar o calendário com ferramentas de análise e recomendação para facilitar a escolha das datas de prova de acordo com quantidade de provas marcadas no dia, e suas respectivas dificuldades, considerando também o calendário dos alunos inscritos na turma ministrada da prova a ser marcada.

Os frameworks escolhidos já contém uma estrutura de testes definida, o VUE possui extensão para navegadores voltada para sua testagem e o NestJS contem diretório para testes automatizados e é integrado com o Jest biblioteca de testes unitários em JavaScript. Como entrega final do projeto, gostaríamos de deixar a aplicação bem segura, estável e de fácil manutenção, para que possa ser continuada e replicada.