Proposta de Construção de um Portal Web para Ensino de Programação Paralela utilizando Blockly e as bibliotecas OpenMP e MPI

João Victor Rocha Brum[1], Ana Paula Canal[1]

[1]Ciencia da Computação – Universidade Franciscana (UFN) –
Santa Maria – RS - Brasil

joao.rocha@ufn.edu.br, apc@ufn.edu.br

Abstract. *The teaching of parallel programming is constantly discussed due to its learning difficulty and lack of tools that enable the study outside the institutional environment. Given this context, this work aims to propose a architecture and to develop a portal for teaching the concepts of OpenMP and MPI with the visual language through blocks. For the implementation of the portal, the JavaScript programming language will be used with the AdonisJS and ReactJS frameworks, along with the MySQL database and the Blockly API, visual language. Thus, aiming to facilitate access to the concepts parallel programming. The work was guided by the best practices of the FDD (Feature Driven Development) agile method.*

Resumo. *O ensino de programação paralela é constantemente discutido por sua dificuldade de aprendizado e a necessidade de ferramentas que viabilizem o estudo fora do ambiente institucional. Diante disso, esse trabalho tem como objetivo propor a arquitetura e desenvolver um portal para auxílio no ensino dos conceitos de OpenMP e MPI com a linguagem visual por meio de blocos. Para o desenvolvimento do portal foi utilizada a linguagem de programação JavaScript com os frameworks AdonisJS e ReactJS, junto ao banco de dados MySQL e a API Blockly, linguagem visual. Esse projeto visa facilitar o acesso aos conceitos sobre programação paralela através de uma ferramenta visual. O trabalho foi dirigido pelo método ágil FDD (Feature Driven Development).*

1. Introdução

A programação paralela, desde o surgimento das arquiteturas de múltiplos núcleos (*multicore*s), vem constantemente se tornando um conhecimento mais requisitado e presente no dia a dia para os profissionais da área da computação [Soares, Nobre e Freitas 2019].

O conceito de computação de alto desempenho ou *High-Performance Computing* (HPC) é um dos temas referentes a arquitetura *multicore* e programação paralela. Este corresponde à supercomputadores ou *clusters* de servidores construídos para o processamento de grandes volumes de dados e em velocidades altas [Oracle 2021]. Dentro deste tema, a programação paralela é essencial para o devido funcionamento dos supercomputadores, pois, para alcançar os níveis de processamento e velocidade desejados se faz necessário a utilização deste paradigma, que nada mais é que uma forma de executar diversas partes da aplicação simultaneamente, de forma

cooperativa, usufruindo da comunicação entre processos e *threads* [Braga, Bezerra e Garcia 2015].

Com isso em mente, Soares, F., Nobre, C., e Freitas, H (2019) apresentaram uma pesquisa referente ao ensino de programação paralela, em cursos de graduação, em diversos locais do mundo, como: Rússia, Brasil, China, Espanha, Estados Unidos da América (EUA), Índia, Suécia, entre outros. Nessa pesquisa, são levantados questionamentos e quantificados artigos relevantes ao assunto. Com essa, foi possível observar que, na maioria das graduações o estudo de paralelismo dá-se ao fim da graduação, por volta do sexto semestre em diante e apenas alguns temas são abordados por serem obrigatórios. Também, que, exceto os EUA, a quantia de pesquisas referentes ao ensino de programação paralela é relativamente baixa, variando de uma a cinco pesquisas por nação.

1.1. Justificativa

O paralelismo, em termos didáticos, é considerado um tema complexo para o ensino, principalmente para os iniciantes na área da computação, pois diversas novas habilidades necessitam serem adquiridas e treinadas [Lima at al. 2018].

Um dos desafios do ensino de programação paralela, dá-se por conta de que, ao contrário da programação que é ensinada durante quase todo o decorrer dos cursos de informática, isto é, a programação sequencial, a programação paralela requer diferentes aspectos para serem desenvolvidos, como processos e *threads*, particionamento de dados, comunicação e sincronização entre processos [Lima at al. 2018].

Segundo Soares, F., Nobre, C., e Freitas, H (2019) a apresentação do conceito e programação em paralelo nos períodos iniciais dos cursos juntamente com a programação sequencial é de suma importância para o desenvolvimento profissional do aluno. Isso se dá pelo fato de que o aluno passará a tratar a programação paralela como algo natural na programação, impedindo assim que este conceito seja interpretado como algo avançando e pouco utilizado. Também é apresentado como meio facilitador a possibilidade de portais educacionais que permitem o estudo dos conteúdos fora do ambiente institucional e é reforçada a necessidade desses estudos e pesquisas, pois poucos trabalhos abordam o tema.

1.2. Objetivo

O projeto tem por objetivo propor a arquitetura e desenvolver uma ferramenta web para auxiliar o ensino de conceitos sobre programação paralela. Abrangendo, a parte teórica e a parte prática, esta, voltada à programação visual dos temas OpenMP e MPI.

2. Referencial Teórico

Esta seção abordará a computação paralela e distribuída, alguns conceitos relacionados ao paralelismo, e a arquitetura computacional na qual o projeto se baseará, ferramentas, trabalhos relacionados e tecnologias que serão utilizadas no desenvolvimento do projeto.

Inicialmente, para melhor contextualizar o assunto, será introduzida, brevemente, a computação paralela e distribuída. Este conceito é de suma importância, pois, sem ela, os recursos de hardware disponíveis não serão utilizados devidamente [Maliszewski et al. 2021].

A computação paralela é um modelo em que as execuções são separadas e processadas individualmente em arquiteturas de memória compartilhada. Já a computação distribuída é um modelo que possibilita a execução em múltiplos processadores e, ou, máquinas, onde a comunicação que ocorre é baseada em troca de mensagens [Maliszewski et al. 2021].

2.1. Conceitos Importantes

Serão definidos conceitos como linguagem visual, processos, threads, arquitetura de memória compartilhada, que será referida como SMP (*Shared-Memory Parallel Computer*) e a arquitetura de memória distribuída, que será referida como DSM (*Distributed Shared Memory*), com o intuito de introduzir os paradigmas de programação paralela do OpenMP e MPI, respectivamente.

No trabalho de Gomes e Brito (2016), onde é apresentado um portal web educativo para ensino da biblioteca OpenCL é utilizada a linguagem de programação visual utilizando a API Blockly. No trabalho em questão, a linguagem visual é caracterizada por ser uma alternativa a programação convencional por ser mais dinâmica, por forcar mais a atenção do usuário na lógica do código do que propriamente a sintaxe.

Inicialmente, para poder, de fato, entrar no assunto de paralelismo, deve-se entender o que são processos e *threads*, pois, o paralelismo baseia-se nestes conceitos. Os processos são abstrações que servem para gerenciar o acesso aos recursos da máquina, possuindo um espaço de endereçamento privado para o armazenamento do código, dados do programa e sua pilha de execução. O modo como um processo é gerado dá-se pelo sistema operacional (SO) no momento de execução de um novo programa. Para cada programa, um ou mais processos serão gerados e cada um atuando independentemente do outro e sendo parcialmente geridos pelo SO. Vale ressaltar também que cada processo pode gerar outros processos [Silva 2019].

Já o conceito de *thread*, caracteriza-se por ser um fluxo lógico de execuções dentro de um processo. Cada processo, pode conter vários *threads*, em casos de mais de um *thread* é caracterizado como *multithreading,* e estas compartilham o mesmo espaço de endereçamento, de arquivos abertos e recursos diversos que compõem o contexto do processo [Silva 2019].

Para que seja possível a devida execução do processo, o SO utiliza alguns fatores que estão dentro do que é chamado, contexto do processo. Esses fatores são definidos como, código do programa, dados armazenados, pilha de execução, conteúdo dos registradores, variáveis de ambiente e descritores de arquivos abertos. Além disso, existem as trocas de contexto, que ocorrem quando o SO transfere o controle de um processo para outro, estas trocas contém os seguintes passos: salvar o contexto atual, restaurar o contexto do novo processo e passar o controle para o novo processo [Silva 2019].

Sobre a execução do *thread,* todo processo inicia sua execução a partir de um *thread* principal, esse *thread* principal gera *threads* secundárias que podem executar concorrentemente com os demais *threads.* Também, da mesma forma que os processos, cada *thread* possui seu próprio contexto vinculado ao contexto do processo que a criou, que se compõe por: identificador, estado da pilha e valores nos registradores [Silva 2019].

A arquitetura SMP apresenta uma única memória física acessível às unidades de processamento via barramento de comunicação de alta velocidade, que as permitem acessar a memória [Chapman, Jost e Ruud 2007], conforme a Figura 1.

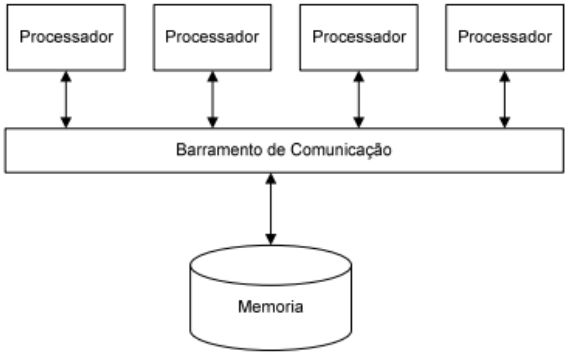


Figura 1 - Arquitetura Geral SMP [Carvalho 2007, p.21]

De acordo com Chapman, Jost e Ruud (2007), os compiladores procuravam realizar o trabalho de identificar e adaptar o código para o paralelismo interno do processador, porém, conforme a escala do computador, ou seja, quanto maior a quantidade de processadores ou núcleos, maior a dificuldade para realizar a função. A razão desta dificuldade, Chapman, Jost e Ruud (2007) definem ser pela necessidade de identificação e verificação de instruções que podem ser executadas em paralelo, e, especificam que para programas mais complexos o compilador não receberá informações suficientes para realizar a decisão, podendo realizar alterações no código, o que não é desejado.

Com isso, na década de 80, foi dado início à pesquisas de formas de acelerar e aperfeiçoar a performance da arquitetura SMP. Contudo, apesar de um aparente sucesso nesse quesito, houve o surgimento de outro problema, a questão da dificuldade de garantir que o resultado foi de fato obtido. Como os processadores dividiam as tarefas entre si, cada um trabalhando independentemente do outro e possuindo tempos de execução diferentes, a tarefa acabou se tornando complicada e trabalhosa. Além disso, apesar dos distribuidores disponibilizarem especificações, ordens, instruções e diretivas possíveis de serem adicionadas nos sistemas, ainda era perceptível a deficiência da execução [Chapman, Jost e Ruud 2007].

Então, a partir destas dificuldades, deficiências e com o esforço do grupo ARB (*OpenMP Architecture Review Board*), surgiu o OpenMP, que é uma API (*Application Programming Interface*) para memória compartilhada e é considerada uma notação. OpenMP foi desenvolvido com o propósito de facilitar e tornar mais ampla a implantação de sistemas SMP, o que realizou com sucesso [OpenMP.org 2021].

O funcionamento do OpenMP dá-se pelo usuário, informando quais instruções serão executadas em paralelo e como serão distribuídas. Dessa forma, o programa será constituído por regiões sequenciais e regiões paralelas, as quais, serão definidas com as diretivas do OpenMP [Chapman, Jost e Ruud 2007].

Por fim, Chapman, Jost e Ruud (2007), definem dois passos para o desenvolvimento com OpenMP. O primeiro sendo a identificação do setor de código sequencial que possua aspectos que viabilizam o paralelismo. Assim, logo que for identificado, será necessário reorganizar o código ou até mesmo substituição de trechos

do código para seu devido funcionamento. O segundo passo é a aplicação das diretivas do OpenMP nos setores identificados como viáveis para aplicação do paralelismo.

Quanto à arquitetura DSM, apresenta memórias fisicamente separadas, ou seja, a memória é distribuída entre os processadores, como ilustrado na Figura 2. Diferenciando-se também das arquiteturas SMP pelo custo de envio de mensagens entre dois nós não depender somente de um processador independente [Foster 1995].

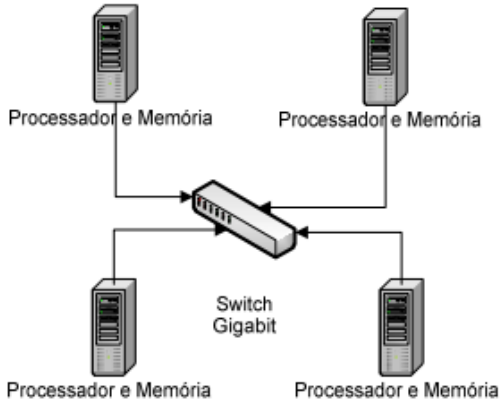


Figura 2 – Arquitetura geral DSM [Carvalho 2007, p.22]

Nessa arquitetura, um dos padrões utilizados é o MPI (M*essage Passing Interface).* Silva (2019) define o MPI como um padrão de troca de mensagens, possuindo diversas funções, que possibilitam uma maior facilidade na implementação de sistemas paralelos.

De acordo com Foster (1995), o desenvolvimento com o MPI compreende um ou vários processos, que, através das funções disponibilizadas, se comunicam entre si enviando e recebendo mensagens. Por praxe, é definido um valor fixo de processos sendo um processo para cada processador e estes podem realizar execuções distintas de aplicações, diferenciando assim o modelo DSM do modelo SMP. Foster (1995) define que a razão pela qual é fixado o número de processos dá-se por utilizarem operações de comunicação ponto a ponto, assim proporcionando ser implementadas comunicações locais e coletivas.

O MPI caracteriza-se por ser considerado um padrão, suportando todas as plataformas de HPC (*High Performance Computing*), estas são as seguintes: ser portátil, ou seja, precisar de mínimas ou nenhuma alteração de código para diferentes plataformas; por sua performance, possuindo suporte nativo do hardware; funcionalidade, contendo diversas rotinas para implementação, definidas em sua última versão MPI-3 (2015); disponibilidade [Barney 2021]; e, por fim, o suporte à programação modular [Foster 1995].

2.2. Tecnologias

O sistema foi desenvolvido no formato de um portal *web*, cuja base do sistema foi constituída das seguintes tecnologias: JavaScript, HTML (*Hypertext Markup Language*) e CSS (*Cascading Style Sheets*). Além da utilização do banco de dados MySQL para o acesso e armazenamento dos dados e, utilizando o conjunto AdonisJS e ReactJS para o desenvolvimento.

O HTML, uma linguagem de marcação, e o CSS, uma linguagem de estilização, são as bases de um *website*. O HTML é responsável por definir a estrutura do website e então auxiliar a interpretação pelo navegador. Já o CSS é responsável para a elaboração do *design* do site, onde os elementos definidos no HTML poderão conter estilizações e aparências distintas [Bortolossi 2012]. O JavaScript é uma linguagem prototipada, que suporta diversos paradigmas da programação e torna as funcionalidades dinâmicas [Mozilla 2021].

Quanto aos *frameworks* AdonisJS e ReactJS. AdonisJS é um *framework*, para desenvolvimento de aplicações web ou para desenvolvimento de API *server*, que utiliza o ambiente de programação NodeJS, usufruindo de um conjunto de diversas ferramentas e bibliotecas, centralizando mais o desenvolvimento [Adonis 2021]. Já o ReactJS, segundo Facebook (2021), é uma biblioteca para o desenvolvimento de interfaces, tendo como principais características ser declarativa, baseada em componentes, de fácil entendimento, assimilação e recordação.

A API *Blockly* é uma interface de desenvolvimento de linguagem visual baseada em blocos, onde é possível gerar trechos de códigos que correspondem aos encaixes. O *Blockly* permite a conversão de diversas linguagens que já estão embutidas na biblioteca, como Javascript e PHP, contudo, para este trabalho, foi realizada a construção dos blocos na linguagem C e das instruções referentes às bibliotecas OpenMP e MPI. Também permite a exportação dos códigos gerados [Google Developers 2021].

O JSON é o formato de conversão de dados atual mais utilizado em aplicações web, que possibilita o fácil entendimento de pessoas e de fácil assimilação da máquina no momento de ser gerado e analisado [Ecma 2017].

2.3. Trabalhos Relacionados

Nesta seção serão apresentados alguns trabalhos correlatos que abordaram os temas de educação, portal web educativo, paralelismo e programação visual baseada em blocos.

O trabalho de Gomes e Brito (2016) trata de um sistema web educativo para o ensino de programação paralela utilizando *Blockly* com OpenCL. Esse padrão de programação, OpenCL, é voltado ao desenvolvimento de aplicações que funcionam em plataformas distintas consistidas de CPUs e GPUs. O sistema levantado por Gomes e Brito (2016) permite ao usuário utilizar o *Blockly* para o desenvolvimento, em blocos, de diversas funções voltadas para a biblioteca OpenCL, podendo, assim, exportar o código para ser testado. O trabalho constitui-se de uma página web que contém a contextualização do projeto, uma seção que contém materiais de apoio referente ao OpenCL e o ambiente do *Blockly*.

O trabalho de Braga, Bezerra e Garcia (2015) trata-se de uma ferramenta que auxilia a aprendizagem de programação concorrente especificamente, o conceito de semáforos. A aplicação consiste em uma interface Java que simula o caso das “Crianças Brincando” cuja premissa dá-se por: existem crianças brincando com bolas e um cesto, cada criança passa um número X de tempo brincando, ao terminar de brincar a bola é guardada no cesto e a criança passa a descansar por um tempo Y. O usuário pode realizar inserções da quantidade de crianças (processos) desejadas na aplicação, os tempos de cada ação, definir se a criança começa com a bola ou não e escolher a

quantidade limite de bolas na cesta. Assim que a primeira criança é inserida outra tela é gerada mostrando as informações de cada criança.

Por fim, o trabalho de Monteiro, Rivero e Barreto (2018) apresentam o processo de desenvolvimento de uma ferramenta desenvolvida em Java *desktop*, que gira em torno do tema "Processo de Estado Finito" (FSP), com o intuito de auxiliar o ensino de modelagem de sistemas distribuídos críticos na prática. A ferramenta permite realizar validações dos modelos gerados pelos alunos como também permite realizar a geração de códigos em Java.

A partir dos trabalhos relacionados citados, pode-se destacar alguns fatores que ressaltam a semelhança dos trabalhos. Começando pelo autor Ramos et al (2016)**,** é apresentado uma plataforma web com o intuito de auxiliar o aprendizado de programação paralela no OpenCL, utilizando o Blockly para o desenvolvimento. No trabalho de Braga, Bezerra e Garcia (2015) é apresentada uma outra plataforma, agora para *desktop,* cujo tema da programação paralela abordado é a sincronização. Por fim, no trabalho de Monteiro, Rivero e Barreto (2018) outra aplicação é apresentada, também em *desktop*, cujo tema abordado é referente a programação de sistemas distribuídos e máquina de estados.

Neste trabalho foi proposta uma arquitetura para um ambiente web e parte de sua implementação para, a partir da programação visual com Blockly, busca-se abordar a programação paralela em C, fazendo o uso das bibliotecas OpenMP e MPI. Dessa forma, a partir do ambiente Blockly, é possível desenvolver algoritmos paralelos, com funções MPI e diretivas OpenMp, no formato da linguagem C, compondo os blocos.

3. Metodologia FDD

A metodologia ágil utilizada para o desenvolvimento do projeto. A metodologia ágil escolhida foi a FDD (*Feature Driven Development)*.

O desenvolvimento dirigido a funcionalidades (FDD) utiliza como filosofia as seguintes premissas: comunicação detalhada tanto visualmente quanto verbalmente; gerência das dependências e complexidade via fracionamento embasados nas funcionalidades; e por fim colaboração da equipe. Além disso é salientado, em sua metodologia, a importância de atividades que garantam a qualidade de software, como: inspeções de código e projeto, desenvolvimento incremental e aplicação de auditorias [Pressman 2010].

Conforme Pressman (2010) existem cinco processos que compõem o FDD, sendo elas: a definição geral do projeto; definição das funcionalidades, de forma que fiquem agregadas em conjuntos e temáticas semelhantes; planejamento baseado em funcionalidades; projetar por funcionalidades, ou seja, um pacote de projeto ou sequência; e desenvolver por funcionalidades. Como ilustrado na Figura 3.

Figura 3 - Processo do FDD, adaptado de Pressman (2010)

Segundo Mazuco (2017), o FDD une as principais vantagens das metodologias disponíveis, possuindo uma única abordagem completa para a Engenharia de Software. Além disso, ela chama a atenção para as suas características, sendo algumas delas: modelagem em objetos de domínio; Desenvolvimento orientado por funcionalidades, facilitando a entrega e descrição delas; Integração regular e hierárquico relacionados aos negócios; Resultados úteis a médio prazo; Planejamento detalhado; Boa visibilidade do projeto.

4. Desenvolvimento

O sistema proposto foi desenvolvido para a plataforma Web. O portal disponibiliza recursos para serem usados no ensino de programação paralela, possuindo teorias dos conceitos propostos e atividades baseadas em programação visual, em blocos. Foi realizada a construção dos blocos de código, na API Blockly, que serão utilizados na aplicação, em linguagem C. A Figura 4 ilustra a arquitetura do projeto.

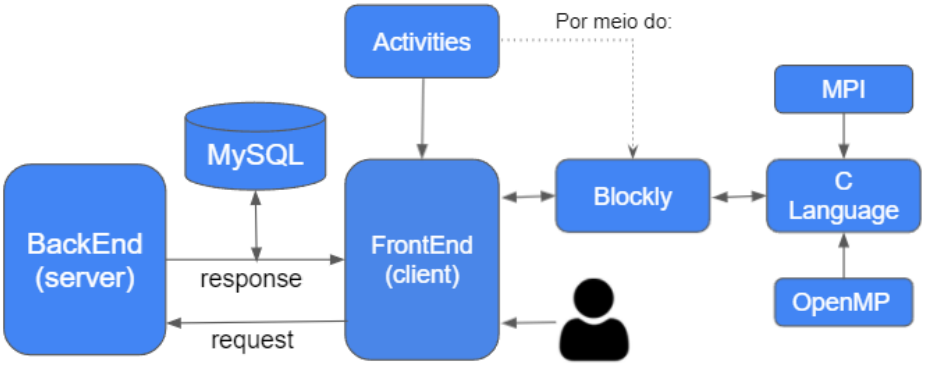


Figura 4 - Arquitetura do Sistema

Há dois tipos de usuários, o usuário Aluno e o usuário Tutor. Ambos os usuários podem interagir com as atividades e com as seções do portal, contudo o usuário tutor pode realizar a revisão das atividades dos alunos.

Os usuários ao acessarem o portal encontram o material de estudo onde são introduzidos os conceitos do paralelismo de modo suscinto e direto, de forma que as atividades possam ser desenvolvidas logo em seguida. Na sequência são apresentadas ao usuário, as atividades referentes ao assunto estudado que consiste inicialmente em realizar a construção do código, via *Codemirror*[1], que é uma ferramenta que simula um editor de texto configurado para a linguagem C.

Assim que a atividade for concluída, ela pode ser colocada para revisão, ao salvar, e realizar o *download*, caso seja desejo do usuário, do arquivo no formato C. As atividades e revisões só podem ser visualizadas e elaboradas quando o usuário estiver logado, caso contrário o usuário convidado, que não realizou o *login*, só possui acesso à documentação e ao *playground*, que é uma área livre para a construção de códigos.

Quanto ao usuário Tutor, além das permissões já descritas, pode realizar a avaliação dos exercícios realizados pelo usuário Aluno, por meio de pareceres

[1] Conforme Haverbeke, *CodeMirror* é um editor de texto versátil implementado em JavaScript para o navegador [Haverbeke 2021].

descritivos que podem auxiliar o aluno no seu aprendizado. O desenvolvimento do trabalho seguiu as etapas da metodologia FDD e é caracterizado nas próximas seções.

4.1. Modelo Abrangente

Seguindo as boas práticas do FDD, a primeira etapa do desenvolvimento é a definição do modelo abrangente que compreende o diagrama de domínio. Como ilustra a Figura 5, nesse modelo é possível visualizar os conceitos e seus relacionamentos.

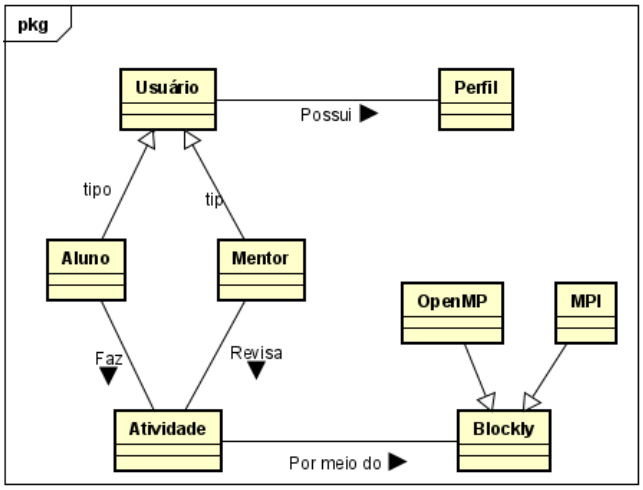


Figura 5 - Diagrama de Domínio

Após o desenvolvimento do diagrama de domínio, o próximo passo da metodologia é o desenvolvimento de uma lista de funcionalidades.

4.2. Listar e Planejar por Funcionalidades

O segundo e terceiro passo do FDD fazem referência à lista de funcionalidades e o planejamento do sistema. Nestas etapas, sob a visão do usuário e desenvolvedor, são analisadas as necessidades de desenvolvimento e a ordem em que estão listadas será a ordem em que serão desenvolvidas. Conforme é demonstrado nos Quadros 1 e 2. Possibilitando assim ilustrar os requisitos funcionais (RF) e requisitos não funcionais (RNF). Estes requisitos representam, respectivamente: as tarefas e comportamento que o sistema precisar ter; e características do sistema quanto à implementação e interfaces que não constituem as funcionalidades do sistema.

Quadro 1 – Requisitos funcionais do sistema

RF 01 – O sistema deverá possibilitar o usuário se cadastrar.	**Complexidade:** Baixa	**Relevância**: Essencial
RF 02 – O sistema deverá possibilitar o usuário realizar o *log in* no portal.	**Complexidade:** Baixa	**Relevância**: Essencial
RF 03 – O sistema deverá permitir o usuário movimentar os blocos e conectar blocos.	**Complexidade:** Baixa	**Relevância**: Essencial
RF 04 – O sistema deverá conter os blocos em linguagem C, OpenMP e MPI	**Complexidade**: Alta	**Relevância:** Essencial
RF 05 – O sistema deverá possibilitar salvar o progresso atual da atividade e retornar no próximo acesso se for desejo do usuário.	**Complexidade**: Alta	**Relevância:** Essencial
RF 06 – O sistema deverá permitir o usuário visualizar as	**Complexidade**: Média	**Relevância:** Essencial

atividades desenvolvidas.		
RF 07 – O sistema deverá possibilitar um usuário, tutor, visualizar a atividade desenvolvida e realizar comentários.	**Complexidade**: Média	**Relevância**: Essencial
RF 08 – O sistema deverá possibilitar o usuário realizar *download* do código como um arquivo no formato C.	**Complexidade**: Média	**Relevância**: Essencial
RF 09 – O sistema deverá possibilitar o usuário realizar alterações no perfil	**Complexidade**: Baixa	**Relevância**: Não - Essencial

Quadro 2 – Requisitos não funcionais do sistema

RNF01 – Linguagem de Programação: O portal será desenvolvido com o uso das linguagens JavaScript, CSS e HTML.
RNF02 – Banco de Dados: Será utilizado para o armazenamento dos dados o banco MySQL.
RNF03 – API Server: O servidor será desenvolvido utilizando NodeJS com o framework AdonisJS.
RNF04 – Layout: O layout do portal será desenvolvido utilizando o framework ReactJS.
RNF05 – Programação em Blocos: Será utilizado a API Blockly do Google.

Com a conclusão da lista de funcionalidades e o planejamento, é possível realizar o desenvolvimento do quarto passo da metodologia que é representada por planejar por funcionalidades.

4.3. Projetar por Funcionalidades

O quarto passo da metodologia corresponde à etapa de projetar por funcionalidades que possibilita uma visão estática do sistema que é representado pelo diagrama de classes, conforme a Figura 6. O diagrama apresenta as classes que fazem parte da implementação do sistema, e seus respectivos relacionamentos com outras classes. Cada classe possui alguns atributos e métodos que serão utilizados.

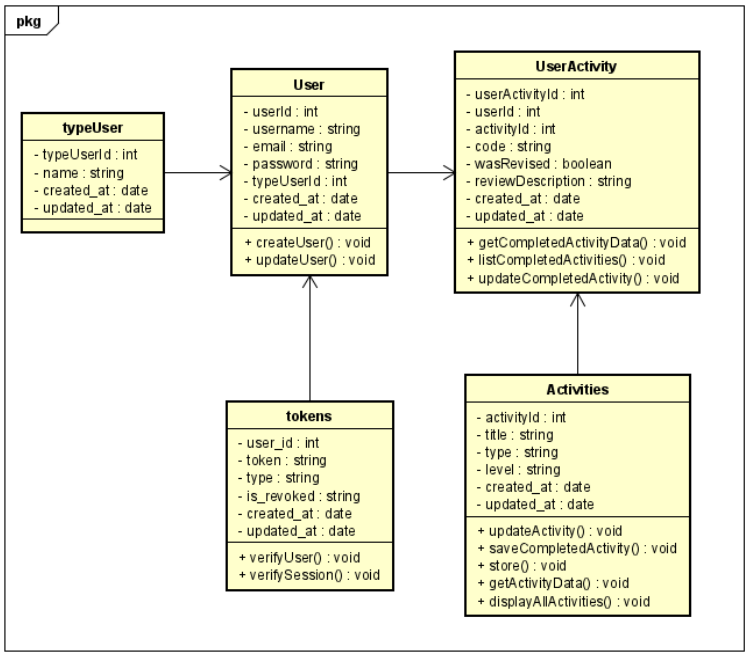
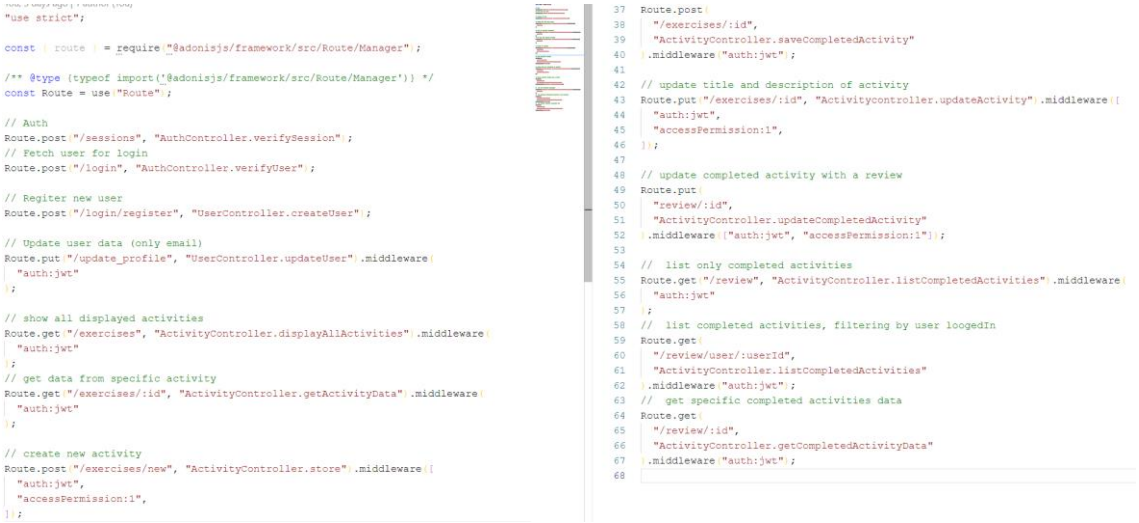


Figura 6 - Diagrama de Classes

4.4. Desenvolver por Funcionalidades

Após a definição dos passos anteriores, foi dado início à implementação que corresponde ao último passo do método ágil FDD. Iniciando-se pelo desenvolvimento do servidor e seguindo para o desenvolvimento das telas. A seguir, a Figura 7 representa as rotas da aplicação.



```
"use strict";

const { route } = require("@adonisjs/framework/src/Route/Manager");

/** @type {typeof import('@adonisjs/framework/src/Route/Manager')} */
const Route = use("Route");

// Auth
Route.post("/sessions", "AuthController.verifySession");
// Fetch user for login
Route.post("/login", "AuthController.verifyUser");

// Regiter new user
Route.post("/login/register", "UserController.createUser");

// Update user data (only email)
Route.put("/update_profile", "UserController.updateUser").middleware([
  "auth:jwt"
]);

// show all displayed activities
Route.get("/exercises", "ActivityController.displayAllActivities").middleware([
  "auth:jwt"
]);
// get data from specific activity
Route.get("/exercises/:id", "ActivityController.getActivityData").middleware([
  "auth:jwt"
]);

// create new activity
Route.post("/exercises/new", "ActivityController.store").middleware([
  "auth:jwt",
  "accessPermission:1",
]);
```

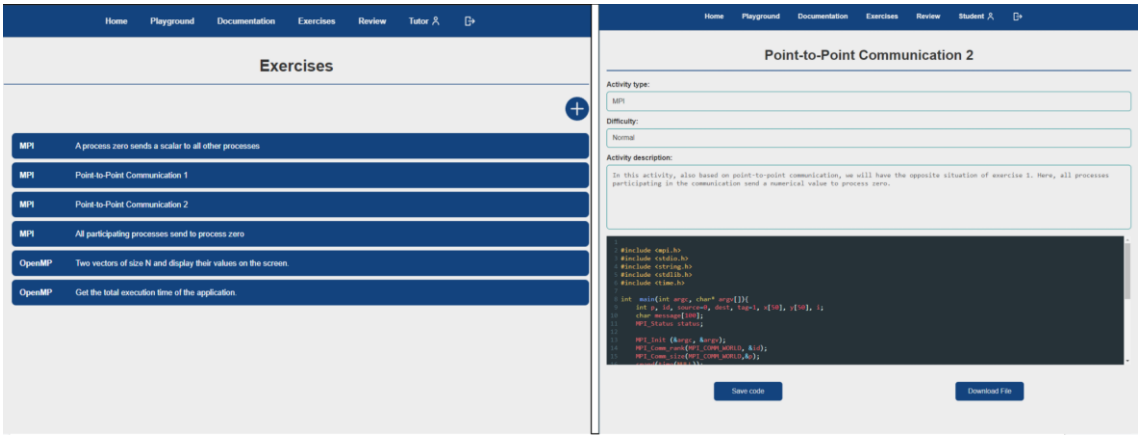
```
Route.post(
  "/exercises/:id",
  "ActivityController.saveCompletedActivity"
).middleware("auth:jwt");

// update title and description of activity
Route.put("/exercises/:id", "Activitycontroller.updateActivity").middleware([
  "auth:jwt",
  "accessPermission:1",
]);

// update completed activity with a review
Route.put(
  "review/:id",
  "ActivityController.updateCompletedActivity"
).middleware(["auth:jwt", "accessPermission:1"]);

//  list only completed activities
Route.get("/review", "ActivityController.listCompletedActivities").middleware([
  "auth:jwt"
]);
//  list completed activities, filtering by user loogedIn
Route.get(
  "/review/user/:userId",
  "ActivityController.listCompletedActivities"
).middleware("auth:jwt");
//  get specific completed activities data
Route.get(
  "/review/:id",
  "ActivityController.getCompletedActivityData"
).middleware("auth:jwt");
```

Figura 7 - Rotas do Portal

No decorrer da implementação das rotas, foram sendo desenvolvidos os métodos e ajustes, conforme necessário. As tabelas do banco foram populadas com as *migrations* e o banco foi populado pelas *seeds*, ambos comandos disponibilizados pelo AdonisJS. Em seguida, com auxílio da ferramenta *Insomnia*[2],, utilizada para realizar requisições ao servidor, os *Controllers* de usuário, atividade e autenticação foram definidos e desenvolvidos.

Com o desenvolvimento do servidor concluído, deu-se início ao desenvolvimento das interfaces gráficas da aplicação, primeiramente sendo desenvolvida as interfaces de *login*, registro, página inicial e documentação. Em seguida, conforme o cronograma definido no FDD, as interfaces das atividades e revisões. A Figura 8 representa a interface da lista de atividades e a interface de resolução da atividade, na visão do aluno. O que difere, neste caso, quanto a visão do tutor, é a existência do botão de edição no lugar do botão de salvar, a interface poderá ser visualizada no Apêndice A.

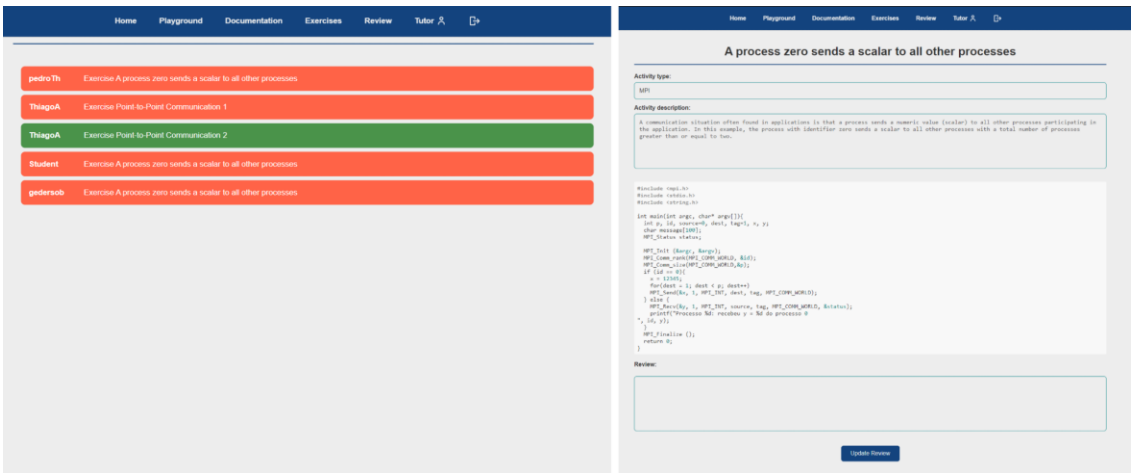
[2] *Insomnia* é um aplicativo, ou ferramenta, desktop para realizar a interação com APIs HTTP [Insomnia, 2021].



(a)Lista de exercícios **(b) Interface de Resolução**

Figura 8 – Interfaces das Atividades, visão do aluno

A seguir, a interface gráfica da Figura 9(a), visão do Tutor, demonstra a área de revisão, onde as atividades concluídas ficam visíveis para o usuário. Atividades já revisadas possuem a coloração verde, enquanto as atividades pendentes para revisão possuem a coloração vermelha. E na interface de Figura 9(b), também na visão do Tutor, representa a tela onde a revisão é aplicada e editada, caso necessário.



(a)Lista de atividade para review **(b) Interface de revisão**

Figura 9 – Interfaces de Revisão, visão do tutor

A ideia inicial do projeto consistia no desenvolvimento do portal utilizando-se da ferramenta Blockly, que é uma ferramenta baseada em blocos, porém ao tentar realizar a geração dos códigos em C foi observado que o desenvolvimento dele envolvia mais questões do que aquelas identificadas inicialmente, o que tornou o escopo do trabalho maior do que o previsto e, consequentemente, requerendo maior tempo para sua execução. Além disso, a atividade se demonstrou mais complexa que o esperado, pois, para melhor aplicação, implica a necessidade de implementação de um pacote para o ReactJS, em C, maior estudo da ferramenta para tornar os blocos criados mais dinâmicos e, por fim, o surgimento de diversos *bugs* ao tentar colocar o quadro do Blockly nas páginas de exercícios, de uma forma satisfatória. Diante disso, optou-se por utilizar outra ferramenta, de implementação mais acessível, chamada *Codemirror*.

Esta ferramenta, *Codemirror* cria uma pequena tela de texto customizável que permite o usuário digitar o código, em C, simulando uma *IDE* (*Integrated Development Environment*). Com isso, foi possível seguir com o desenvolvimento do portal sem maiores barreiras. O *Codemirror*, pode ser visualizado tanto na Figura 8(b).

Para realizar a implementação do Blockly, foi necessário realizar a construção dos blocos em C, através da ferramenta disponibilizada pela biblioteca, chamada "Blockly *Factory*", onde, a partir de blocos conectados ia-se gerando a forma, a definição e uma pré implementação do código do bloco. A outra parte do gerador, que compreende o código gerado, foi realizada no projeto a partir da escrita manual da resposta, referente ao bloco. A Figuras 11 representa o bloco que compreende a inclusão da biblioteca do MPI e a Figura 12 ilustra a criação dos blocos na Blockly Factory.

```
Blockly.JavaScript['includes_mpi'] = function (block) {
  var code = '#include <mpi.h>\n';
  return code;
};
```

Figura 11 – Interface Playground, gerador do bloco "includes_mpi"

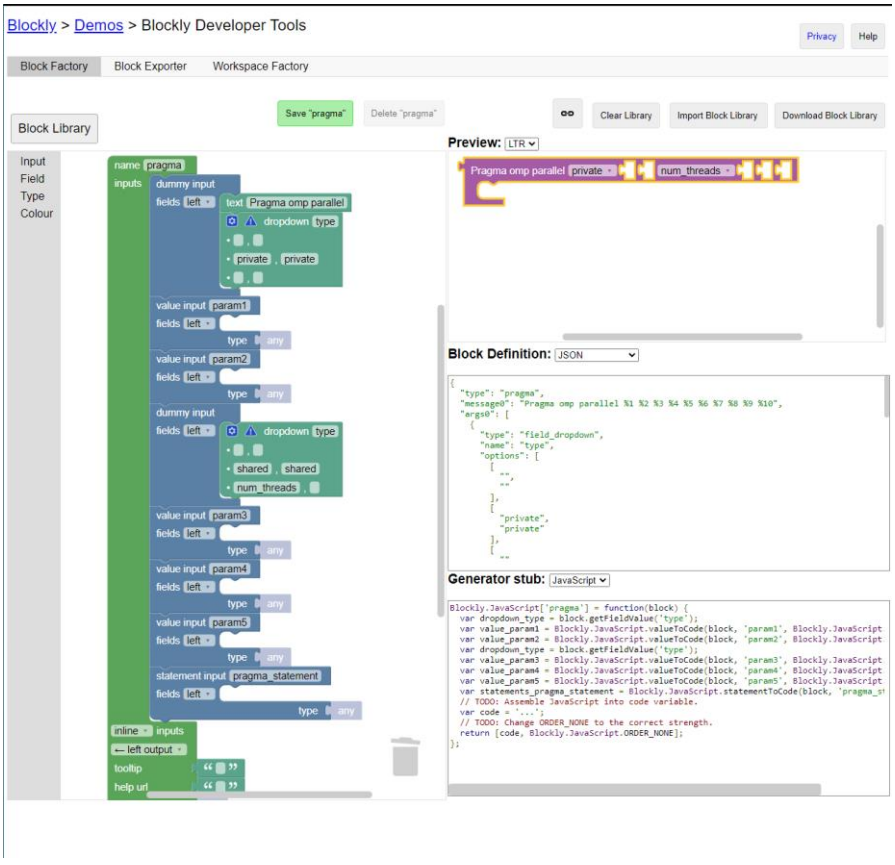


Figura 12 – Blockly Factory, bloco pragma omp

A Figura 13 representa a ferramenta Blockly inserida na aba *Playground*, possuindo alguns blocos, customizados para linguagem C e o código respectivo.

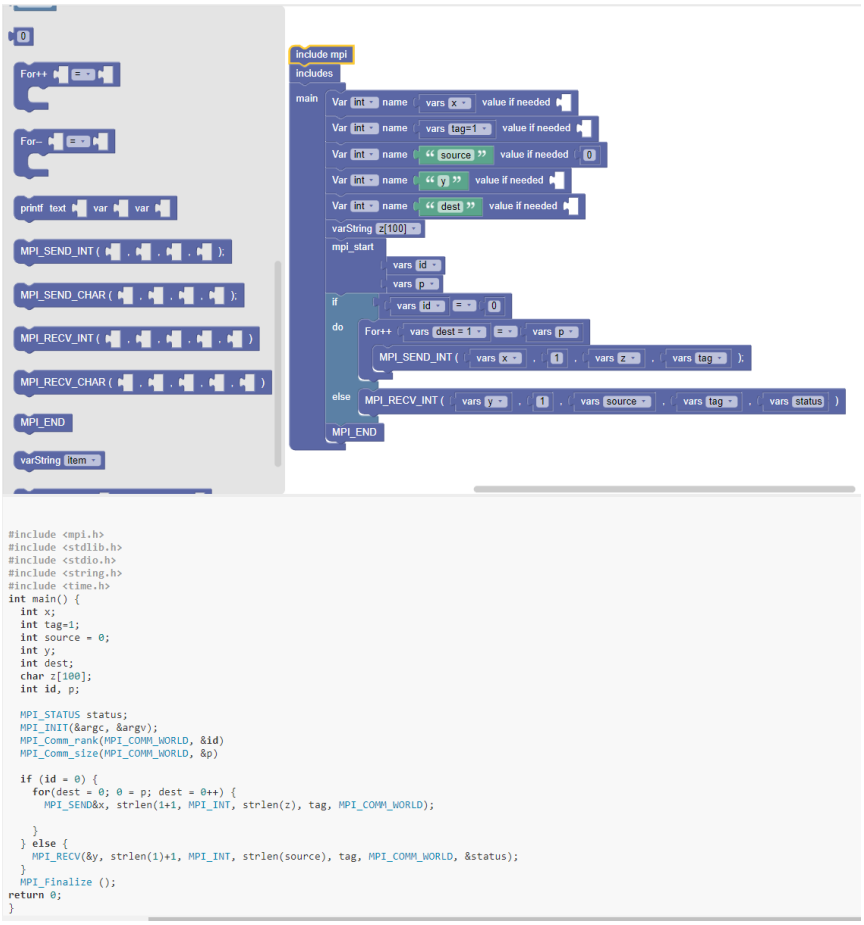


Figura 13 – Interface Playground

Por fim, nesta etapa, foi realizado o desenvolvimento do servidor, com os controladores de atividade, autenticação, e a parte do cliente que compõe as seguintes interfaces: página inicial, *playground*, documentação, lista de exercícios, exercícios, lista de exercícios para revisão e revisão da atividade, que podem ser encontradas no Apêndice A.

5. Conclusões e Trabalhos Futuros

Dessa forma, o presente trabalho apresentou a proposta de desenvolvimento de um sistema web para o ensino de programação paralela dos paradigmas OpenMP e MPI, onde foram descritos os principais requisitos, sua modelagem e o desenvolvimento do portal, sendo apresentados ao fim do trabalho alguns resultados e suas dificuldades.

Os trabalhos referenciados e correlatos demonstraram a importância da programação paralela e de tecnologias, ou sistemas, que auxiliem no aprendizado da matéria, apontando também que sistemas destinados a esta finalidade se mostram escassos.

A metodologia FDD, com seu planejamento, foi de grande auxílio para o desenvolvimento do projeto, tendo em vista que permitiu ter uma visão mais ampla do projeto como um todo e por ser flexível com as mudanças. Sendo assim, uma ótima opção de metodologia para o desenvolvimento do software.

Alguns dos problemas enfrentados durante o desenvolvimento do projeto, além do Blockly, foram relacionados às tecnologias ReactJS, CSS e a parte de autenticação. Dentre estas tecnologias, os conceitos de ReactJS foram os que mais tiveram efeito negativo, pois, foi necessário realizar o estudo quase completo da ferramenta.

Como sugestões para o seguimento do trabalho, para melhor complementá-lo e aperfeiçoá-lo, alguns pontos podem ser citados, como: realizar a implementação do pacote Blockly em C; aplicar o quadro do Blockly nos exercícios; melhorar a aplicação dos blocos; ao cadastrar, o usuário que desejar ser tutor, enviar uma solicitação, no portal, para o representante ou administrador; realizar a adaptação do portal para comportar completamente os dispositivos móveis; realizar o aprimoramento da documentação; por fim, permitir alteração da documentação pelos Tutores.

Referências

Adonis. (2021). *A fully featured web framework for NodeJS.* Disponível em: https://adonisjs.com/, acessado em 06 maio 2021.

Braga, S., Bezerra. H. e Garcia. F. (2015). *"Crianças brincando":* Uma ferramenta para auxílio na aprendizagem de programação concorrente. In. XLIII Congresso Brasileiro de Educação em Engenharia.

Barney, B. (2021) *Message Passing Interface (MPI)*. Lawrence Livermore National Laboratory (LLNL). Disponível em: https://hpc-tutorials.llnl.gov/mpi/, acessado em: 04 junho 2021.

Bortolossi, H. J. (2012). *Criando conteúdos educacionais digitais interativos em matemática e estatística com o uso integrado de tecnologias:* GeoGebra, JavaView, HTML, CSS, MathML e JavaScript. Revista do Instituto GeoGebra Internacional de São Paulo, v. 1, n. 1, p. 28-36.

Carvalho, V. (2007) *Otimização de um cluster de alto desempenho para o uso do programa PGENESIS em simulações biologicamente plausíveis em larga-escala de sistemas neurais*. 102 f. Dissertação (Mestrado em Ciências) – Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto da USP, Ribeirão Preto, SP, 2007.

Chapman. B., Jost, G e Ruud, A. (2007). *Using OpenMP – Portable Shared Memory Parallel Programming*. The MIT Press Cambridge, Massachusetts London, England.

Ecma. (2017) *ECMA-404 The JSON data interchange syntax.* Disponível em: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/, acessado em 07 maio 2021.

Facebook (2021). *React. A JavaScript library for building user interfaces.* Disponível em: https://reactjs.org/, acessado em 06 maio 2021.

Foster, I., (1995) *Designing and Building Parallel Programs.* Disponível em: https://www.mcs.anl.gov/~itf/dbpp/, acessado em 31 junho 2021.

Gomes, J., Brito, A. (2016). *Um Ambiente Baseado em Blocos para Ensino de Programação Paralela com OpenCL.*

Google Developers. (2021) *A Javascript library for building visual programming editors.* Disponível em: https://developers.google.com/blockly, acessado em 23 maio 2021.

Haverbeke, N. (2021) *Codemirror*. Disponível em: https://github.com/codemirror/Codemirror, acessado em 08 de dezembro de 2021.

Insomnia (2021). *Introduction to Insomnia.* Disponível em: https://docs.insomnia.rest/insomnia/get-started, acessado em 08 de dezembro de 2021.

Lima, A. et al (2018). *Uma Oficina para Ensino de Algoritmos Paralelos por Meio de Computação Desplugada*. In: VII Congresso Brasileiro de Informática na Educação, Fortaleza. p. 619.

Mazuco. A. (2017). *Percepções de Práticas Ágeis em Desenvolvimento de Software:* Benefícios e Desafios. Xv,163., il. Dissertação (Mestrado Profissional em Computação Aplicada). p. 95 – Universidade de Brasília.

Maliszewski A. et al (2021). *Ambiente de Nuvem Computacional Privada para Teste e Desenvolvimento de Programas Paralelos.* Minicursos da XXI Escola Regional de Alto Desempenho da Região Sul, p. 104-128.

Monteiro, E., Rivero, L. e Barreto, R. (2018) *Uma Ferramenta de Suporte ao Ensino de Modelagem de Sistemas Distribuídos Críticos:* Uma Experiência Prática. VII Congresso Brasileiro de Informática na Educação 2018.

Oracle (2021) *O que é computação de alto desempenho (HPC)?* Disponível em: https://www.oracle.com/br/cloud/hpc/what-is-high-performance-computing/, acessado em 19 junho 2021.

Pressman, R. S. (2010) *Engenharia de Software*: Uma Abordagem Profissional, 7 ed., McGraw Hill, 2010.

Silva, P. (2019) *Programação Paralela com MPI – Um Curso Prático,* 1 ed.

Soares, F., Nobre, C., e Freitas, H (2019) *Parallel Programming in Computing Undergraduate Courses:* A Systematic Mapping of the Literature. IEEE Latin America Transactions, Vol, 17, NO. 8.

APÊNDICE A – Demais interfaces do sistema

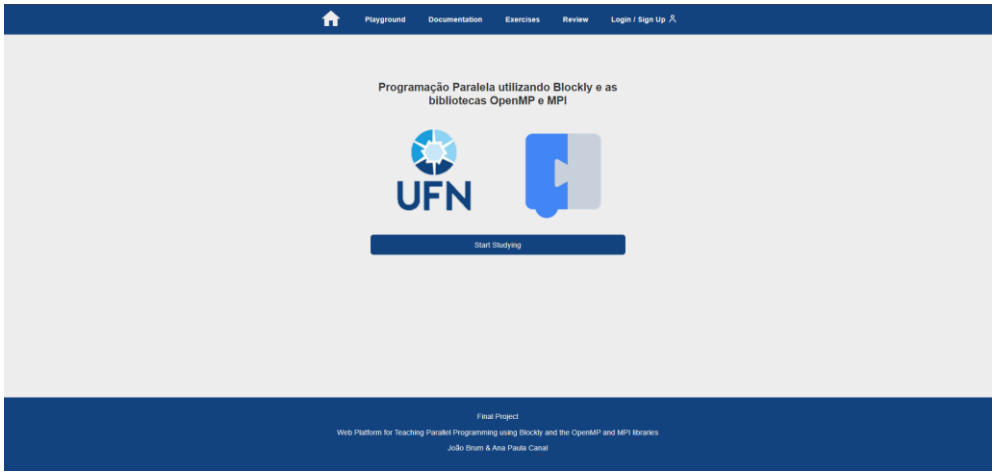


Figura 14 – Interface: Página inicial

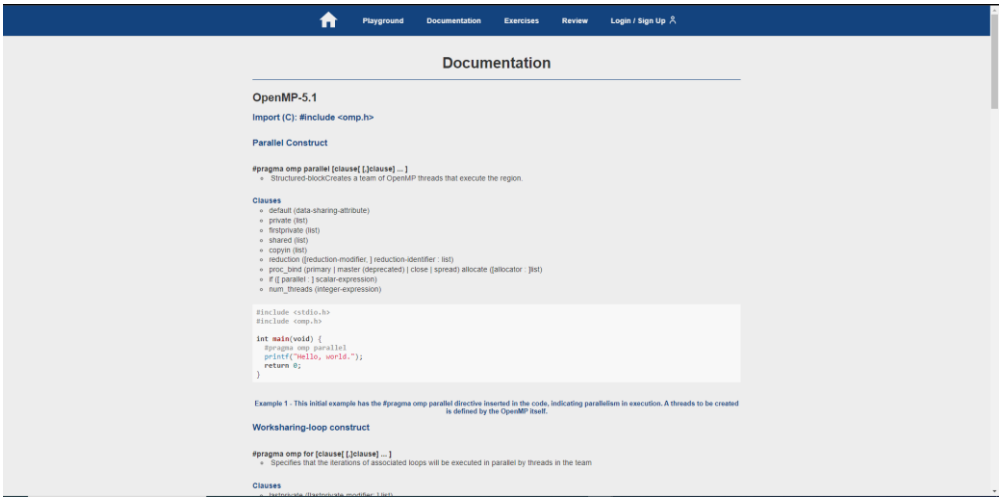


Figura 15 – Interface: Documentação 1

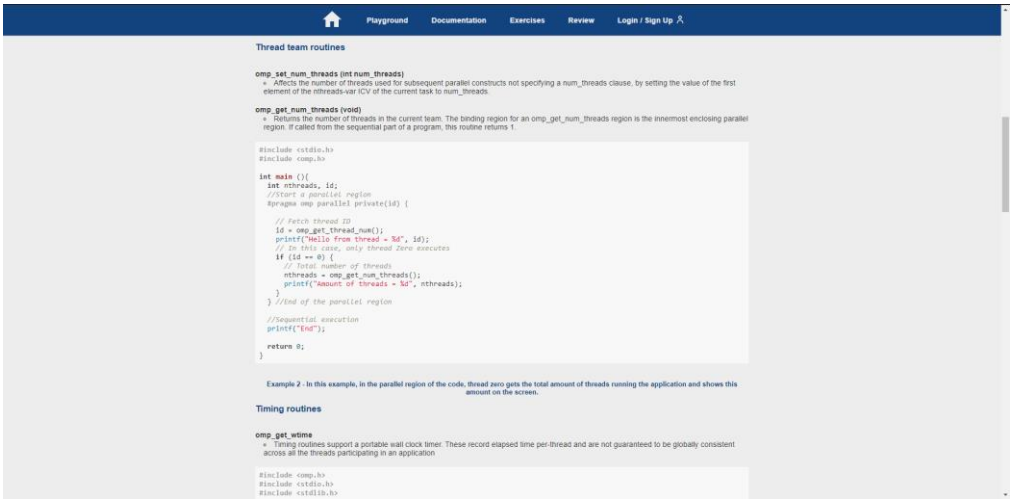


Figura 16 – Interface: Documentação 2

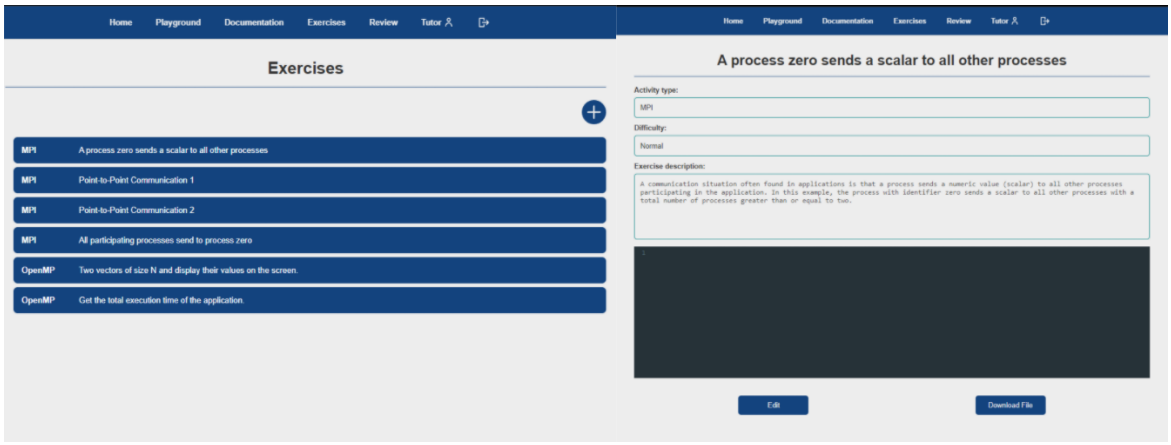


Figura 17 – Interface: Exercícios, visão tutor

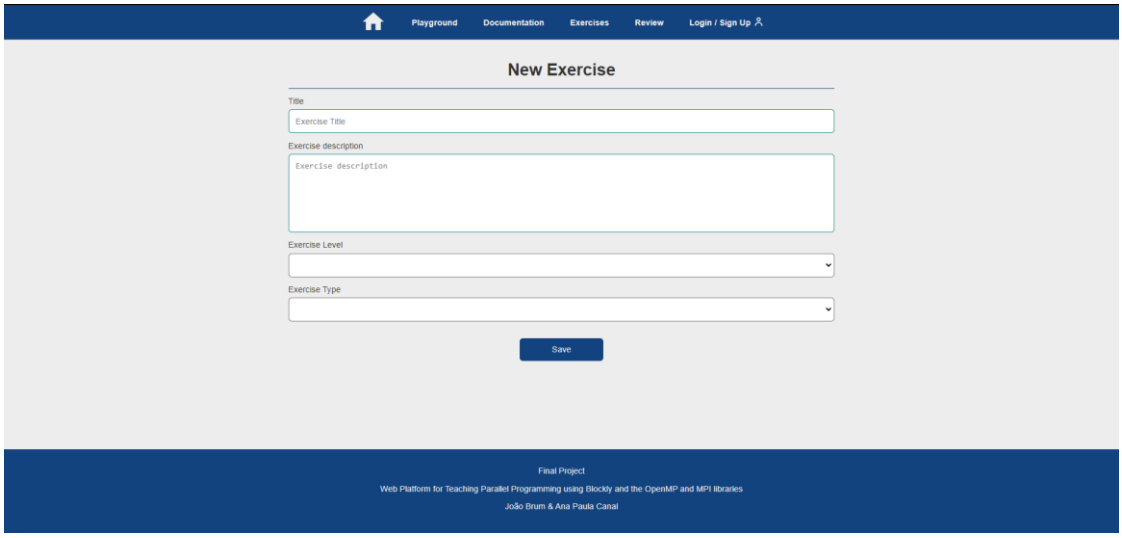


Figura 18 – Interface: Novo Exercício, visão tutor

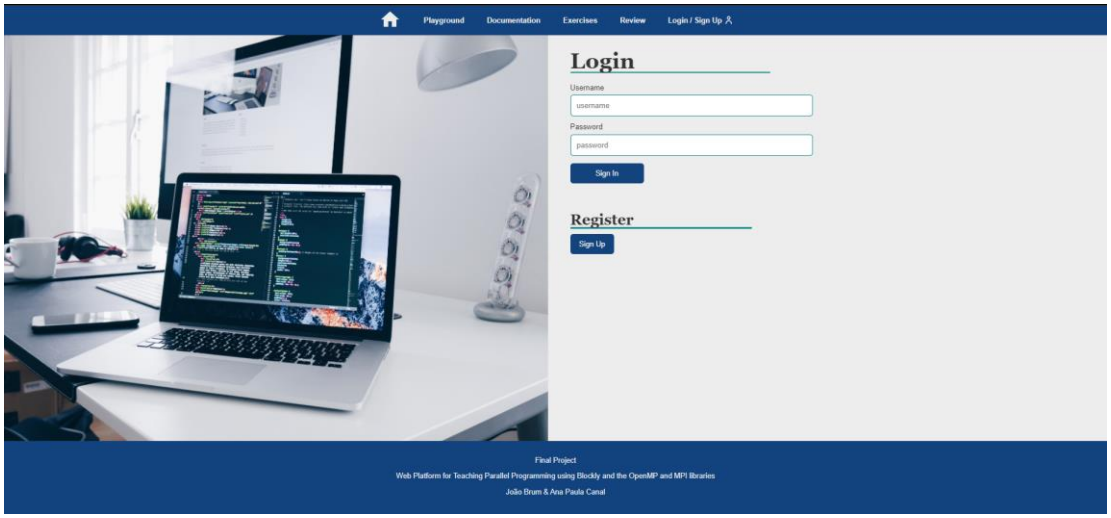


Figura 19 – Interface: Acesso

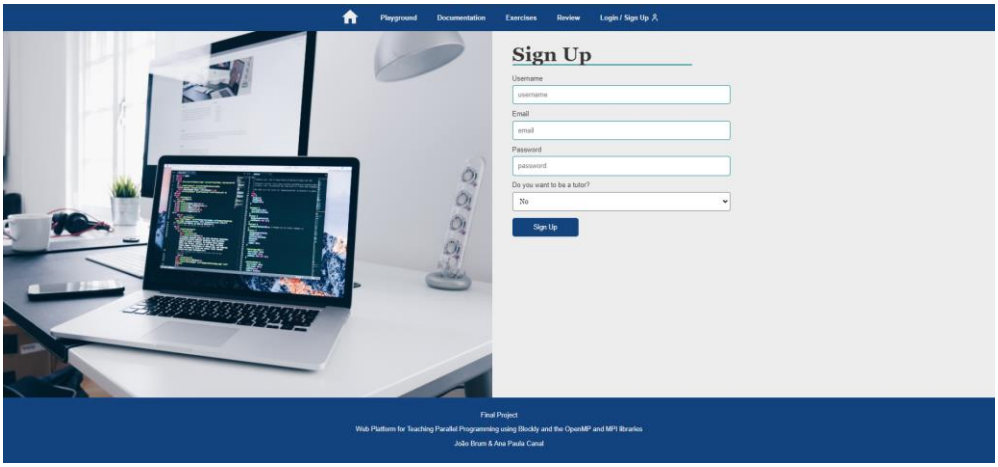


Figura 20 – Interface: Cadastro

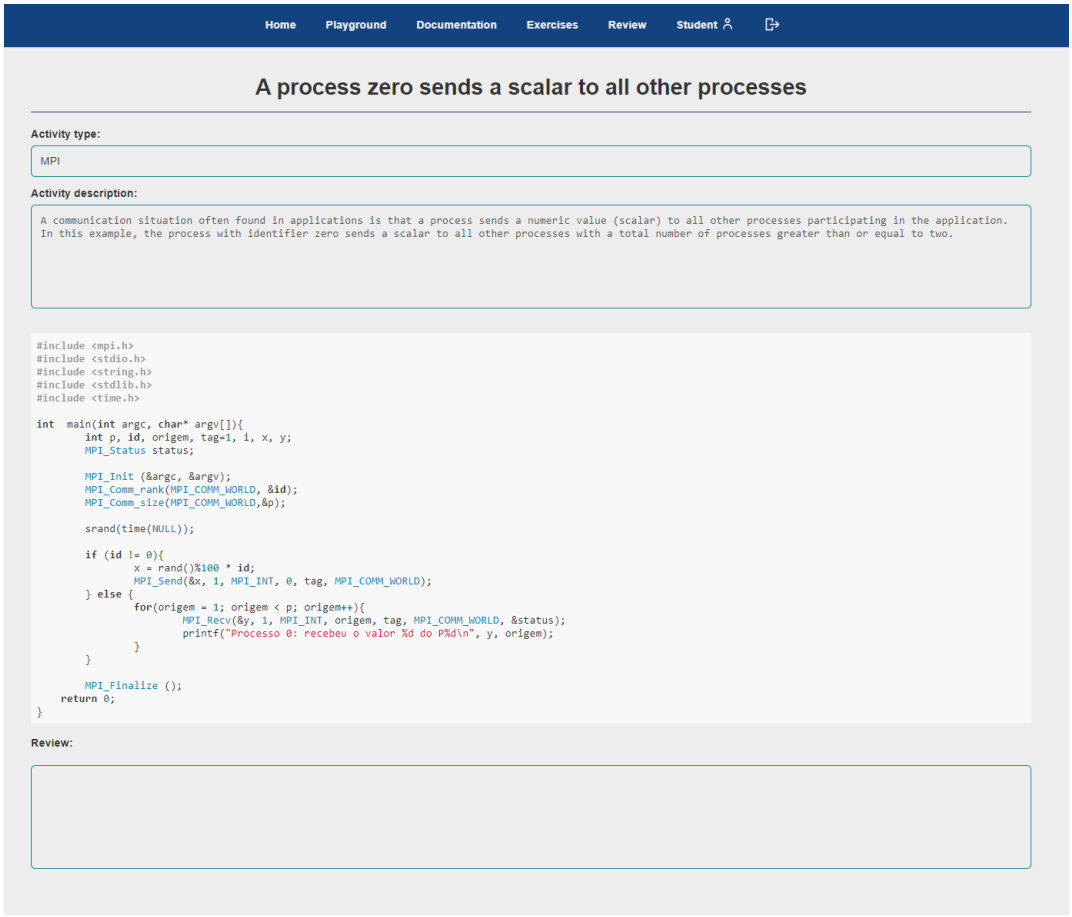


Figura 21 – Interface: Revisão da atividade, visão do aluno

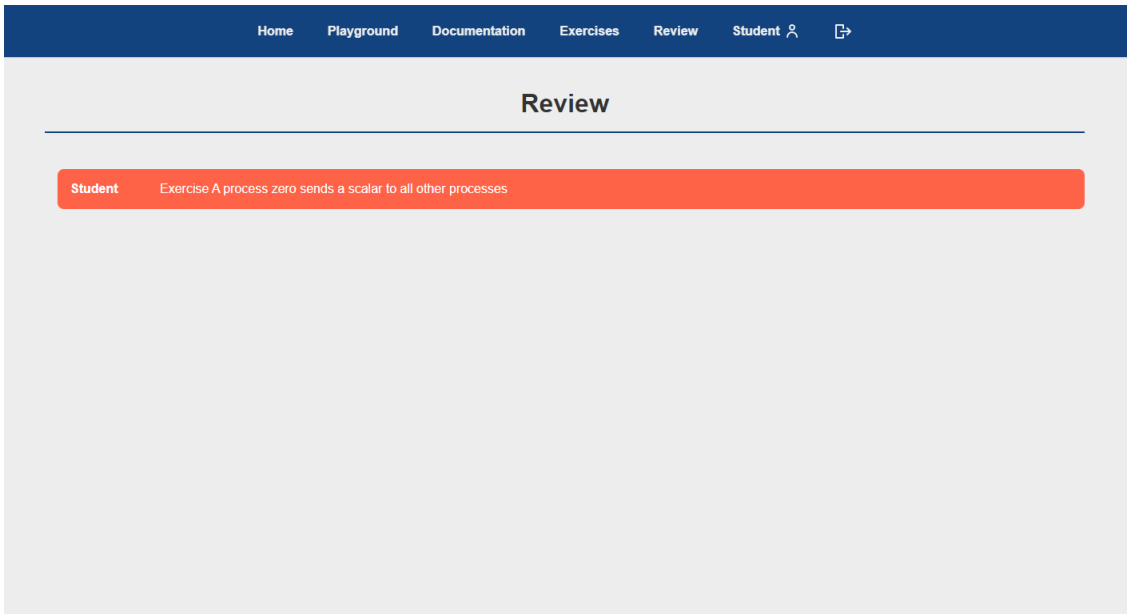


Figura 22 – Interface: Lista de revisão, visão do aluno