Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Departamento Acadêmico de Informaática– DAINF
Engenharia de Computação
Oficina de Integração 3 (EEX23) – S71 – 2018/1

Relatório Técnico

Implementação de sistema de gerenciamento inteligente de tempo

Anderson Antonio Campanha andersoncampanha@alunos.utfpr.edu.br
Brendon Leonam Pasquim brendonpasquim@gmail.com
Isabella Mika Taninaka isabellamika@gmail.com
Matheus Machado Cassol matheus_cassol@hotmail.com

Abril de 2018

Resumo

Este trabalho apresenta o desenvolvimento de um produto capaz de monitorar o tempo gasto em diferentes atividades através da interação com um objeto físico, um cubo, em que cada face representa uma atividade realizada pelo usuário. Para marcar o início de uma atividade, basta que o usuário troque a face voltada para cima. Este objeto foi desenvolvido utilizando um NodeMCU como núcleo do sistema, controlando diversos displays gráficos, contadores e identificando a face voltada para cima. Junto a ele, há uma página web com relatórios dos tempos de atividade. A principal motivação para o desenvolvimento deste trabalho é reduzir o tempo que o usuário precisa gastar para monitorar as próprias atividades e disponibilizar essas informações para que o mesmo possa analisar os próprios hábitos e tomar decisões mais conscientes. Para a elaboração deste projeto, foi feita uma pesquisa teórica de componentes e tecnologias para a solução deste problema e o desenvolvimento foi feito em quatro frentes principais: sistema embarcado, processamento e armazenamento de dados, comunicação e a página web.

1 Introdução

Este projeto tem como objetivo desenvolver um produto capaz de tornar física a tarefa de monitorar nossos gastos de tempo. Desta forma, será possível criar o ciclo de trabalho definido pela "Técnica do Pomodoro" e gerar visualizações posteriores desdes ciclos de trabalho, dando acesso ao usuário a informações sobre seus hábitos e consciência sobre sua rotina.

A solução proposta é o Smart Pomodoro: um cubo que coleta dados de tempos e atividades que geram resultados que podem ser acessados em uma página web.

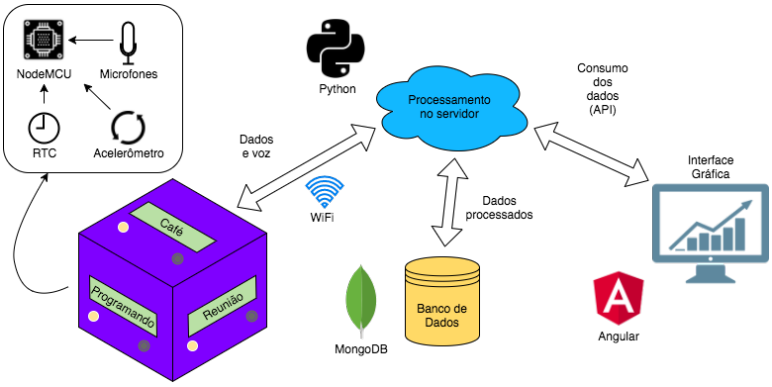

Figura 1: Visão Geral do Smart Pomodoro

O Smart Pomodoro é um cubo, com cinco faces representando atividades diferentes e uma face usada para desligar. A face virada para cima representa a atividade atual, ou desligamento. Cada face de atividade contém um display de LCD onde estão escritos o nome da atividade e o tempo decorrido desde que a face foi selecionada. Em cada face também há um botão, que é utilizado para indicar que o usuário está ciente do fim de um ciclo de trabalho e, um LED para indicar o início ou o fim de um ciclo de trabalho.

Para alterar a atividade representada por uma face o usuário poderá utilizar a aplicação web, bem como acompanhar análises dos dados referentes as atividades. Essas análises contêm gráficos e tabelas com os valores coletados pelo Smart Pomodoro.

Dentre as informações fornecidas na página web estão dados relacionados a data em que a atividade foi executada, a duração da atividade, número de pausas entre atividades, histórico da sequência de atividades que foram executadas, entre outros.

O sistema *Smart Pomodoto* é baseado na amplamente conhecida "Técnica do Pomodoro", desenvolvida originalmente na década de 1980, pelo italiano Francesco Cirillo. [1]

2 Componentes e Tecnologias

A partir do objetivo e requisitos estabelecidos para o projeto, foram feitas pesquisas de componentes e tecnologias que possibilitariam o desenvolvimento e cumprimento destes critérios. As soluções encontradas para as diversas partes do projeto estão apresentadas ao longo desta seção, dividida em três subseções, referentes ao sistema embarcado, à estação base e à comunicação.

2.1 Sistema Embarcado

O sistema embarcado é composto por quatro partes principais: NodeMCU, o processador, o acelerômetro, para a identificação das faces, os LCDs, para exibir as informações e a alimentação do sistema. Todas as partes mencionadas estão descritas nas subseções a seguir.

2.1.1 NodeMCU

O sistema embarcado foi desenvolvido usando um kit de desenvolvimento baseado no ESP8266, um módulo *Wi-fi* e usando o *firmware* da NodeMCU. Esse kit é ideal para o desenvolvimento de objetos *IoT*, internet das coisas. Tais objetos usam tecnologia embarcada, sensores e atuadores para coletar dados e se conectar a rede, dando a objetos do cotidiano habilidades de comunicação e computação.

Por isso, o kit se encaixa nos requerimentos do projeto, além de possuir as seguintes características:

- WiFi integrada nativamente com o protocolo TCP/IP
- ADC (Analog Digital Converter - Conversor analogico digital) com resolução de 10 bits
- Suporte a I2C
- Memória flash com 4MB
- Tensão de operação de 3.3V
- Baixo consumo energético
- Tamanho reduzido
- Programação via usb usando a interface de programação do Arduino

Um dos principais motivos para esta escolha é o consumo de energia deste ser baixo quando comparado às outras opções do mercado. Um Raspberry Pi 3 B+ consome cerca de 400 mA quando não está executando nenhum código e esse consumo pode passar de 1A ao realizar processamentos mais intensos. Um Arduino Mega possui um consumo menor, cerca de 80 mA constantemente, contudo não possui suporte nativo para WiFi necessitando de um módulo/shield que aumentaria o gasto energético. Já o NodeMCU possui um consumo de 80 mA enquanto envia dados pela WiFi, consome cerca de 10 mA com a função WiFi completamente desabilitada e, segundo o datasheet, chega a consumir 60uA no modo Deep Sleep.

2.1.2 Acelerômetro

Para realizar a identificação de qual face se encontra voltada para cima utilizamos um acelerômetro, que é um dispositivo que mede o ângulo de inclinação em que o mesmo se encontra. A partir de ângulos medidos em testes foi possível encontrar faixas que definem qual face se encontra ativada.

2.1.3 LCDs

Para mostrar a contagem de tempo e o nome da tarefa ao usuário no sistema embarcado (Pomodoro) foram utilizados displays LCD. LCDs foram escolhidos ao invés de displays de LED para essa aplicação principalmente pelo menor custo e maior facilidade de programação do display LCD.

A comunicação entre os displays e o microcontrolador é realizada por meio do protocolo I2C, escolhido por permitir que usando apenas dois pinos seja possível realizar a comunicação com todos os 5 LCD's apenas comunicando o endereço do display ao qual a mensagem se destina.

Os displays não possuem nativamente a capacidade de se comunicar utilizando I2C, por isso foi utilizado um adaptador que é responsável por traduzir o I2C para algo que o display compreenda. Este adaptador é o componente que permite que seja atribuído um endereço para cada LCD, de modo que poderíamos utilizar até 8 displays usando este método.

2.1.4 Alimentação

Para a alimentação do sistema foi escolhida a utilização de um power bank de 2500 mAh. Este tipo de bateria foi selecionado ao invés de uma bateria comum pela facilidade de recarga: no caso de uma bateria seria necessário abrir a estrutura para retirá-la, já com o power bank basta conectar um cabo micro usb e o carregamento já se inicia.

2.2 Estação Base

Neste projeto a estação base tem o objetivo de processar os dados coletados no sistema embarcado, exibi-los de forma útil ao usuário e enviar alguns comandos simples ao embarcado. Foram pesquisadas tecnologias capazes de cumprir este objetivo de forma eficiente. Esta seção está subdividida em duas subseções, referentes ao processamento e o banco de dados, e à página web.

2.2.1 Processamento e Banco de Dados

O processamento dos dados fornecidos pelo embarcado e a disponilização deles para serem utilizados pela página web estão descritos nesta seção. A solução de banco de dados escolhida foi o MongoDB e para o processamento, a linguagem de programação Python.

O MongoDB é um banco de dados que não utiliza tabelas, mas documentos BSON, representações binárias JSON. O JSON é um tipo de documento que guarda informações chave-valor[2].

Dentre as principais vantagens encontradas na utilização deste banco de dados, listam-se algumas:

- Não há necessidade de se declarar o esquema do banco de dados e, a estrutura dos campos de diferentes documentos não precisa ser a mesma. Novos campos, portanto, podem ser adicionados aos documentos sem afetar documentos antigos.
- Os objetos dentro do código podem ser modelados naturalmente a documentos do MongoDB. Estruturas hierárquicas como vetores podem ser representados diretamente dentro do banco de dados. Além disso, é um padrão que se comunica bem com a página web, podendo responder à requisões da mesma sem precisar realizar grandes modificações nos atributos, mantendo um padrão que é entendido por todo o sistema e facilitando a programação.
- O MongoDB foi desenvolvido tendo em mente arquiteturas distribuídas e pode ser facilmente escalado sem mudanças na API e distribuído em diversos *data centers*, caso houvesse necessidade.

O Python foi escolhido devido as diversas bibliotecas e *frameworks* disponíveis para ser realizado o desenvolvimento. Vale destacar as principais utilizadas no sistema:

- Flask [3] para o desenvolvimento WEB;
- PyMongo [4] para a integração com o MongoDB.

2.2.2 *Página Web*

Para o desenvolvimento da página web (*front-end*), foi escolhida a tecnologia Angular. O Angular é um conjunto de ferramentas (*framework*) para desenvolvimento na linguagem de programação Typescript [5]. Este *framework* é ideal para desenvolver SPAs (*single-page applications*), que possuem diversas vantagens:

- O usuário tem uma experiência melhor, pois não precisa esperar recarregamentos do site completo. Além disso a navegação fica mais parecida com a de uma aplicação desktop (no computador) ou aplicativo (no celular).
- O sistema ganha muito desempenho, pois ele é carregado na primeira requisição de forma assíncrona que permite o usuário já consumir o conteúdo sem esperar que tudo seja carregado por completo e as requisições

seguintes são responsáveis por trafegar apenas os dados brutos entre o cliente e o servidor, normalmente no formato JSON.

- O tempo de requisição é diminuído nas SPAs, pois a lógica fica mais concentrada no lado do cliente e processada no navegador.
- A manutenção do sistema completo fica mais fácil, pois existe uma separação integral entre *back-end* (lado do servidor) e *front-end* (lado do cliente). Toda a lógica de navegação, exibição de dados e interação está do lado do cliente, então o lado do servidor fica apenas com a tarefa de prover e salvar os dados no banco.

Além disso, o Angular oferece facilidades para a produção de um código organizado. Cada componente (tabela, menu, formulário, etc) é criado separadamente, podendo ser usado em diferentes telas. Isso reduz a necessidade de código redundante para o desenvolvedor. O *framework* também contém uma interface de comandos em terminal chamada AngularCli. O AngularCli oferece diversos comandos para facilitar a criação e *debug* do sistema.

2.3 Comunicação

O formato da comunicação entre a estação base e o sistema embarcado é apresentado nessa seção.

A comunicação aqui apresentada tem por objetivo integrar os dados coletados no sistema embarcado (Smart Pomodoro) e os dados armazenados e processados na estação base (servidor HTTP hospedado na nuvem). A tecnologia escolhida para troca de dados entre o sistema embarcado e a estação base foi o padrão WiFi IEEE 802.11n.

Os dados entre o sistema embarcado e a estação base são trocados utilizando conexão com a Internet, através de requisições HTTP do tipo GET, ao solicitar dados e, POST ao submeter dados. Esse modelo de comunicação foi escolhido para que houvesse uma fácil integração com a API RESTful em Python implementada na estação base.

O formato JSON foi escolhido como estrutura de dados padrão para a comunicação, principalmente devido seu desempenho e facilidade de leitura (tanto humana quanto computacional), bem como integração direta com o formato de dados do MongoDB.

3 Desenvolvimento

O desenvolvimento do projeto está descrito nesta seção. Cada etapa do desenvolvimento do sistema embarcado, estação base e comunicação está descrita, assim como a integração entre as partes e a estrutura física do Smart Pomodoro.

3.1 Estrutura Física

Inicialmente, a estrutura física foi desenvolvida utilizando 6 placas de papelão, coladas em forma de cubo com fita adesiva. Estas placas foram cortadas com aberturas para os displays de LCD. O modelo inicial não tinha aberturas para os botões e LEDs do Smart Pomodoro, pois foi criado apenas para o teste do projeto. A placa onde o circuito foi montado foi colada na estrutura, usando duas tiras de papelão como apoio. Este apoio foi necessário para que houvesse espaço entre a placa e a face de baixo, pois a face abaixo da placa contêm um display de LCD. Após a montagem do modelo inicial, uma versão definitiva foi modelada em 3D utilizando o software Fusion 360. Esta versão completa foi planejada para ser impressa em 3D, utilizando a impressora do laboratório LASER na UTFPR.

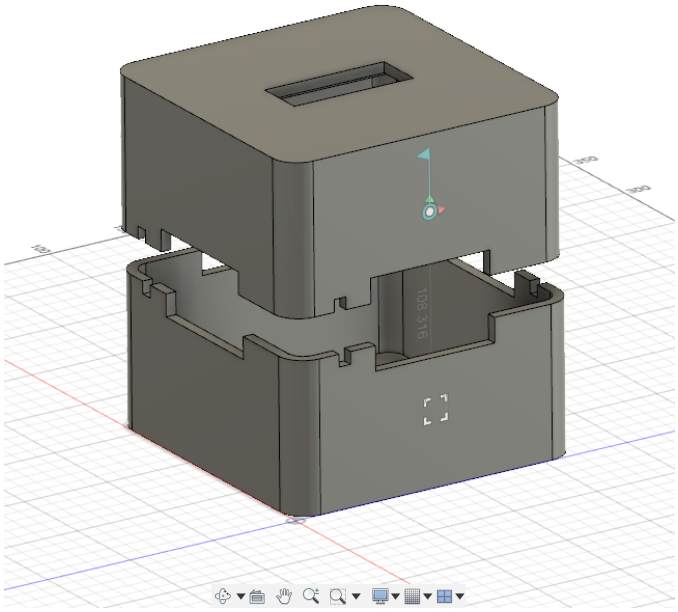

Figura 2: Modelo em 3D da estrutura do Smart Pomodoro

Infelizmente, após algumas tentativas de impressão mal sucedidas, foi decidido que a estrutura inicial de papelão seria utilizada para a entrega do projeto. Foram adicionados os detalhes que faltavam na estrutura inicial e feito um acabamento para que fosse mais robusta.

3.2 Sistema Embarcado

Para o desenvolvimento do sistema embarcado foram feitos o diagrama esquemático mostrado na Figura 3 e os testes individuais dos componentes necessários.

Foram feitos testes do processador, garantindo a gravação dos programas no NodeMCU, além dos testes da comunicação TCP/IP.

Então, foram soldados os adaptadores I2C aos displays, de modo que estes pudessem ser usados de forma fácil por meio da interface dos adaptadores.

O próximo passo foi testar o acionamento dos LCDs e do acelerômetro ligados na I2C, botões e LEDs associados a cada face. Primeiro individualmente, depois integrado ao sistema geral.

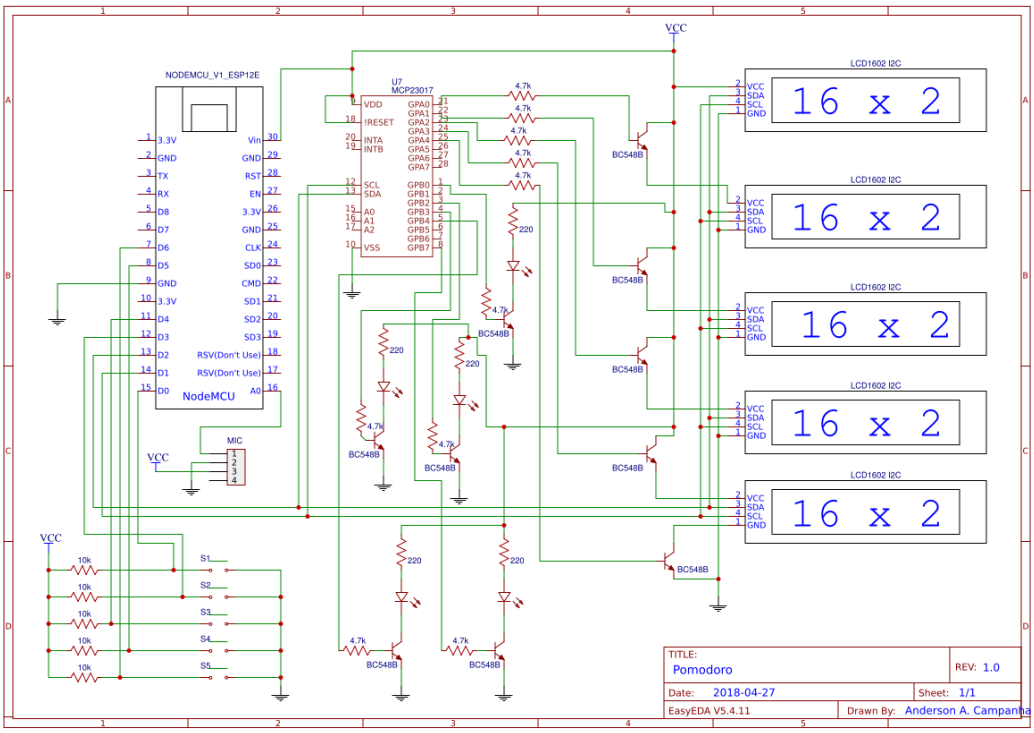

Figura 3: Diagrama esquemático da parte eletrônica do Smart Pomodoro

Após a garantia de funcionamento dos componentes integrados, os mesmos foram soldados a uma placa perfurada de 10cm X 10cm seguindo o esquemático da Figura 3, excetuando os displays, botões e LEDs que precisavam ser anexados às paredes do cubo e, portanto, foram soldados apenas aos *jumpers*. Todas as conexões entre os componentes foram feitas usando *jumpers*.

3.3 Estação Base

Os *softwares* desenvolvidos para estação base, neste projeto, estão descritos nesta subseção. A estação base, neste caso, é o computador do usuário que irá acessar o site do Smart Pomodoro. O site está hospedado em um servidor e pode ser acessado pelo usuário via *browser*.

3.3.1 *Processamento e Banco de Dados*

Foram definidos os casos de uso que seriam atendidos pelo sistema e o esquema que seria seguido no banco de dados para o usuário, as atividades realizadas por ele e uma listagem de todos os pomodoros produzidos.

Os objetos da coleção de usuários seguem o formato da tabela 1.

Tabela 1: Usuário

```
{ "_id": "username",
  "password": "hash(password)",
  "email": "email@email.com",
  "pomoID": ObjectID("ID"),
  "faces": ["Desligado", "Face 1", "Face 2", "Face 3"
     , "Face 4", "Face 5", "Pausa"]}
```

Os objetos da coleção de atividades seguem o formato da tabela 2.

Tabela 2: Atividade

```
{ "_id": ObjectId("ID"),
  "userID": "username",
  "name": "Atividade",
  "timeStarted": timestamp,
  "timeEnded": timestamp,
  "timeSpent": Integer}
```

Os objetos da coleção de pomodoros seguem o formato da tabela 3.

Tabela 3: Pomodoro

```
{ "_id": "ObjectID("ID"),
  "userID": "username"}
```

Uma vez tendo os dados definidos, foram programadas os acessos para cada um deles. Utilizando o *framework* Flask, foram criados caminhos para fazer acessos usando métodos HTTP. As rotas, com os métodos disponíveis para cada uma delas e a resposta gerada estão disponíveis na tabela 4.

Tabela 4: URIs disponíveis

URI	Método	Ação
/api/login	GET	Autoriza ou não autoriza um usuário a utilizar o sistema
/api/users	GET	Retorna a lista de todos os usuários cadastrados
/api/users	POST	Cadastra um novo usuário
/api/users/<user_id>	GET	Retorna um único usuário identificado por user_id
/api/userfield/<field>	GET	Retorna o campo identificado por field do usuário logado
/api/userfield/<field>	PUT	Modifica o campo identificado por field do usuário logado
/api/activity	GET	Retorna todas as atividades cadastradas e o tempo total gasto em cada uma delas pelo usuário logado
/api/activity	POST	Cadastra uma nova atividade para o usuário logado
/api/activity/<activity_name>	GET	Retorna o tempo total gasto pela atividade identificada por activity_name pelo usuário logado
/api/activity/<activity_name>	DELETE	Remove a atividade identificada por activity_name pelo usuário logado
/api/activitylist	GET	Retorna o nome de todas as atividades já realizadas pelo usuário logado

As URIs disponibilizadas permitem o acesso ao banco de dados tanto a partir da página Web quanto a partir do sistema embarcado, servindo de peça fundamental para a integração do sistema. Uma vez tendo a API definida, ela foi disponibilizada em um serviço online, permitindo que os demais componentes do sistema se conectassem a ela para desponibilizar ou utilizar os dados.

3.3.2 *Página Web*

Primeiramente foi criado um desenho básico de como seria o site (*wireframe*) de acordo com a necessidade de interação. As *views* definidas para o site foram:

- *Home*: apenas exibição das principais atividades do usuário em gráfico e tabela.
- Análise: tabelas e gráficos mais complexos a serem escolhidos por periodo e tipo.

- Configurações: configurações de usuário, criação e remoção de atividades.

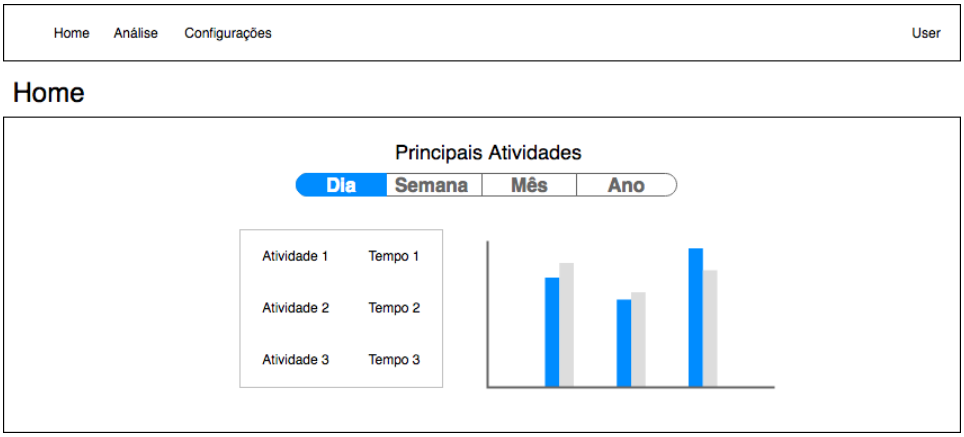

Figura 4: *Wireframe* da tela *Home*

A partir do *wireframe* foi desenvolvida a parte visual e de navegação. Esta etapa apenas envolveu a criação do projeto Angular, programação do roteamento (navegação entre as *views*) e criação do *layout* com HTML. Para o desenvolvimento do *layout* foi utilizado o conjunto de templates Bootstrap. O *Bootstrap* contém templates de componentes e animações que livram o programador da preocupação com o design da aplicação.
Após esta etapa, foi desenvolvida a aplicação utilizando dados falsos. Todo o funcionamento das exibições de dados foi programado, de forma que os dados falsos pudessem ser substituídos por dados obtidos a partir de requisições para o servidor. A tela *home* foi programada com uma tabela e um gráfico de barras estático exibindo as três principais atividades do usuário e seus tempos totais do dia, semana, mês e ano. A tela de Análise foi desenvolvida com gráficos dinâmicos (o número de entradas pode variar). Foi programada a seleção de tipos de gráficos:

- Todas as atividades: tempos totais de todas as atividades em gráfico de pizza.
- Tempo total e Tempo total %: tempos totais em três intervalos do periodo selecionado em gráfico de linha.
- Média diária: média diária dos tempos em três intervalos do periodo selecionado em gráfico de barra.

Os gráficos foram desenvolvidos utilizando a biblioteca ng2chart. Esta biblioteca, feita para Angular, possibilita a criação de vários tipos de gráficos com

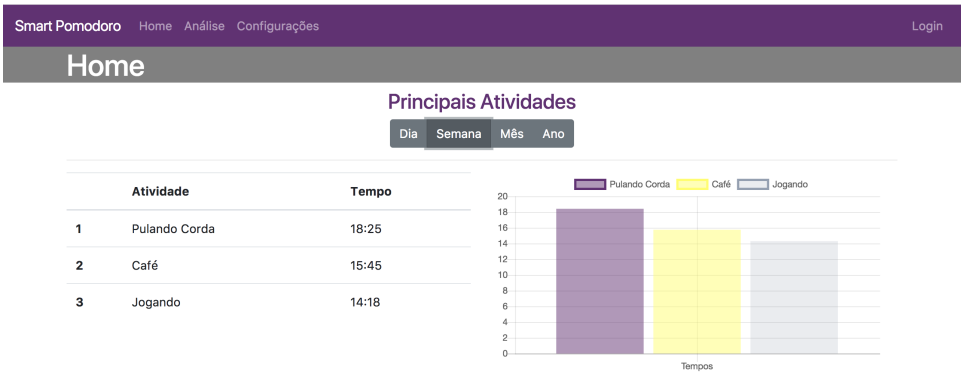

Figura 5: Tela *Home* do site

visual e animações interessantes. Apesar de ser muito bem feita, e parecer relativamente simples, o uso desta biblioteca foi mais complicado que o esperado. A implementação dos gráficos dinâmicos (número de entradas variável) se provou uma tarefa complexa, mas foi executada por fim.

3.4 Comunicação

A comunicação do sistema foi dividida em 2 partes:

- Comunicação implementada na Estação base;
- Comunicação implementada no sistema embarcado;

A comunicação na estação base foi implementada com a linguagem Python e o *framework Flask* (que é voltado para aplicações *WEB* com *HTTP*).

Na estação base os dados são enviados ao sistema embarcado através de requisições *HTTP POST*, que tem como principal objetivo criar ou alterar configurações de comportamento do sistema embarcado. A autenticação do usuário que esta usando o sistema embarcado é um exemplo de uso da comunicação entre estação base e sistema embarcado.

Já no sistema embarcado podem ocorrer requisições do tipo *GET* ou *POST*. Isso ocorre porque o sistema embarcado além de gerar dados e submete-los a Estação base (requisições *POST*) pode solicitar informações a estação base, como por exemplo, verificar se existe um usuário autenticado para uso do sistema embarcado (requisições *GET*).

3.5 Integração

Além do desenvolvimento das partes do SmartPomodoro que foram descritas, o desenvolvimento do produto final envolve um trabalho de integração entre

as partes. Nesta subseção estão descritos os processos de integração entre as partes do projeto.

3.5.1 Página Web e Banco de Dados

A integração entre página web e o banco de dados foi criada através do desenvolvimento de uma *API*: uma aplicação que responde a um conjunto de requisições web. Na programação da página web, o comando é enviado para a *API*, que retorna os dados no banco.

Primeiramente foi desenvolvida toda a programação da *API* em Python. Essa aplicação foi criada para receber requisições e devolver texto no formato JSON, que pode ser lido facilmente pela pelo *front-end*. Após a criação das respostas as possíveis requisições, a *API* foi publicada no endereço: http://smartpomodoro-backend2-smartpomodoro.1d35.starter-us-east-1.openshiftapps.com/.

Com a aplicação publicada, a parte da programação da página que utilizava dados falsos foi substituída por requisições apontando para este endereço. Da mesma forma a autenticação de usuário foi desenvolvida, utilizando a *API* para bloquear acessos não autorizados.

4 Conclusão

O desenvolvimento deste projeto mostrou que se podem utilizar conhecimentos técnicos e científicos para facilitar aspectos da vida cotidiana como o monitoramento do tempo. Através do uso de tecnologias de *hardware* e *software*, assim como métodos de comunicação, foi desenvolvido um produto que atende a necessidade das pessoas de ter consciência sobre seus próprios hábitos.

Durante o desenvolvimento, notou-se uma grande complexidade ao trabalhar com os diferentes sistemas (embarcado, estação base e comunicação) de forma integrada. O trabalho em equipe e integração entre as partes individuais exige um esforço de organização e comunicação entre os membros. A documentação, cronograma e análise dos riscos tiveram um papel importante para a execução das tarefas.

A equipe teve a oportunidade de aprender sobre as tecnologias escolhidas por cada membro, que podem ser úteis no mercado de trabalho ou vida acadêmica. As dificuldades encontradas em cada parte do projeto possibilitaram um aprendizado mais profundo em determinadas áreas. O trabalho de integração também foi importante para o aprendizado, pois cada membro da equipe trabalhou também com outras áreas além da sua própria.

Conclui-se então que a proposta do projeto foi executada e os objetivos foram atingidos, apesar das dificuldades encontradas pela equipe.

Referências

[1] Francesco Cirillo. The pomodoro technique, 2018. Disponível em: https://francescocirillo.com/pages/pomodoro-technique.

[2] MongoDB. Mongodb docs, 2018. Disponível em: https://docs.mongodb.com/.

[3] Armin Ronacher. Flask docs, 2018. Disponível em: http://flask.pocoo.org/.

[4] MongoDB. Pymongo docs, 2018. Disponível em: https://api.mongodb.com/python/current/.

[5] Angular. Angular docs, 2018. Disponível em: https://angular.io/docs.