PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

PUC Minas Virtual

Pós-graduação *Lato Sensu* em Arquitetura de *Software* Distribuído

Projeto Integrado

Relatório Técnico

PyAgenda

Luiz Felipe Basile Ribeiro Filho

Belo Horizonte
04/2023

Projeto Integrado – Arquitetura de Software Distribuído

Sumário

1. Introdução

Ferramentas de gestão de tarefas são ferramentas que ajudam a organizar o trabalho e a aumentar o foco e o desempenho pessoal. Elas permitem que os usuários organizem e priorizem as tarefas, gerenciem seu tempo, monitorarem seu progresso e se mantenham responsáveis. Ao usar essas ferramentas, os usuários podem aumentar a produtividade, melhorar a qualidade e reduzir o estresse. Elas também ajudam a manter um alto nível de foco e fornecem feedback instantâneo sobre o progresso das tarefas. Ao usar essas ferramentas, os usuários também podem aumentar o desempenho, pois elas permitem que eles se concentrem nos aspectos mais importantes de seu trabalho.

No entanto, muitas pessoas enfrentam dificuldades para instalar programas em seus dispositivos. Alguns dos principais problemas enfrentados incluem a incompatibilidade do sistema, a falta de espaço de armazenamento, a falta de acesso à internet e a instalação incorreta. Além disso, muitas vezes, as pessoas não sabem como instalar um programa corretamente, o que pode resultar em problemas de segurança ou outros problemas. É importante que os usuários sigam as instruções fornecidas com o programa para garantir que o processo de instalação ocorra sem problemas. Se o usuário tiver problemas para instalar o programa, ele deve entrar em contato com o suporte técnico para obter ajuda.

Dessa forma, um sistema web para gerenciar as tarefas a serem cumpridas oferece uma maneira simples, rápida e eficiente de realizar tarefas relacionadas ao gerenciamento de projetos. Não requer o uso de instalações, permitindo que usuários acessem o sistema de qualquer lugar com acesso à internet. Além disso, ele permite que tarefas sejam realizadas de maneira mais eficaz, pois os usuários podem acompanhar o progresso de suas tarefas em tempo real, além de partilhar informações importantes em tempo real. Isso ajuda a manter os usuários informados sobre o que está acontecendo em seu projeto e pode ajudar a reduzir o tempo gasto com a realização de tarefas. Além disso, não existe nenhum custo de instalação e sua manutenção é mínima, tornando-o uma opção acessível para a maioria das organizações.

O objetivo deste trabalho é desenvolver um sistema web para gerenciamento de tarefas e eventos que possibilite a organização e o acompanhamento das atividades de maneira ágil e eficaz aos usuários.

Os objetivos específicos propostos são:

- Criar um sistema web com interface intuitiva e responsiva, permitindo o acesso ao sistema em diversos dispositivos;
- Desenvolver funcionalidades que permitam ao usuário cadastrar e gerenciar tarefas e eventos, estabelecendo descrições e datas;
- Realizar testes de funcionalidade no sistema para garantir a funcionalidade e a usabilidade do sistema.

2. Cronograma do Trabalho

A seguir é apresentado o cronograma proposto para as etapas deste trabalho.

Datas		**Atividade / Tarefa**	**Produto / Resultado**
De	**Até**		
03 / 03 / 2023	10/ 03 / 2023	1. Definir os requisitos funcionais	Tabela de requisitos funcionais
03 / 03 / 2023	10/ 03 / 2023	2. Definir os requisitos não funcionais	Tabela de requisitos não funcionais
11 / 03 / 2023	22/ 03 / 2023	3. Definir os mecanismos arquiteturais	Tabela de mecanismos arquiteturais
11 / 03 / 2023	22/ 03 / 2023	4. Modelagem arquitetural	Diagramas arquiteturais
23 / 03 / 2023	10 / 04 / 2023	5. Desenvolvimento do protótipo funcional	Protótipo funcional do sistema proposto
23 / 03 / 2023	10 / 04 / 2023	6. Testes da funcionalidade do protótipo	Resultados dos testes
23 / 03 / 2023	10 / 04 / 2023	7. Hospedagem do protótipo	Acesso web à ferramenta
11 / 04 / 2023	20 / 04 / 2023	8. Gravação do vídeo	Vídeo explicativo
21 / 04 / 2023	23 / 04 / 2023	9. Elaboração do relatório	Relatório técnico

3. Especificação Arquitetural da solução

3.1 Restrições Arquiteturais

R1: O software deve ser desenvolvido em Python, com o *microframework* Flask
R2: O software usa o banco de dados relacional SQLite

3.2 Requisitos Funcionais

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	O sistema deve permitir o auto cadastramento de usuários novos	B	A
RF02	O sistema deve permitir o acesso aos usuários cadastrados	B	A
RF03	O sistema deve permitir a criação de tarefas/eventos;	B	A
RF04	O sistema deve permitir a consulta de tarefas/eventos;	B	A
RF05	O sistema deve permitir a atualização de tarefas/eventos;	B	A
RF06	O sistema deve permitir a exclusão de tarefas/eventos;	B	A
RF07	O sistema deve permitir a exclusão da conta do usuário;	M	M
RF08	O sistema deve permitir o logout dos usuários;	B	B
RF09	O sistema deve permitir a exclusão de tarefas/eventos;	B	A
RF10	O sistema deve permitir a autenticação dos usuários;	M	A
RF11	O sistema deve impedir contas com e-mail já cadastrado.	M	A

*B=Baixa, M=Média, A=Alta.

3.3 Requisitos Não-funcionais

ID	Descrição	Prioridade B/M/A*
RNF01	O sistema deve ser apresentar disponibilidade 24 X 7 X 365;	A
RNF02	O sistema deve apresentar desempenho de no máximo 2 segundos;	M
RNF03	O sistema deve ter mecanismos de autenticação de usuários para restringir o acesso a dados confidenciais;	A
RNF04	O sistema deve ter backups regulares para garantir a integridade dos dados;	M
RNF05	O sistema deve possuir uma interface simples e intuitiva;	M
RNF06	O sistema deve ser capaz de se adaptar ao crescimento do número de usuários;	A
RNF07	O sistema deve possuir documentação clara e de fácil compreensão para facilitar a manutenção.	B

*B=Baixa, M=Média, A=Alta.

3.4 Mecanismos Arquiteturais

Análise	*Design*	Implementação
Persistência	Banco de dados relacional	SQLite
Front end	Multi Page Application	Html5 e CSS3
Back end	Servidor web	Python
Integração	Banco de dados SQLite	Python 3.9
Log do sistema	Registro de erros	Pythonanywhere
Teste de Software	Verificação de funcionalidade	Teste de funcionalidade
Deploy	Hospedagem web	Pythonanywhere

4. Modelagem Arquitetural

Para esta modelagem arquitetural optou-se por utilizar o modelo C4 para documentação de arquitetura de software. Dos quatro níveis que compõem o modelo C4 três serão apresentados aqui e somente o Código será apresentado na próxima seção (5).

4.1 Diagrama de Contexto

A solução proposta pode ser compreendida através do diagrama de contexto apresentado na Figura 1.

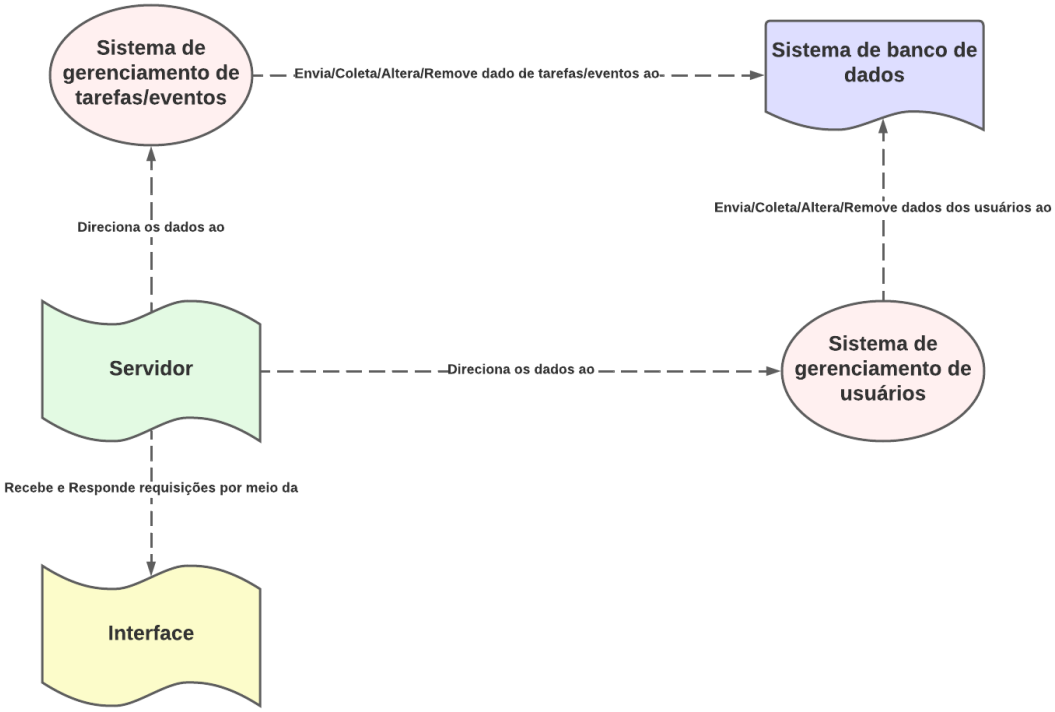

Figura 1 - Diagrama de Contexto do sistema proposto. Fonte: Próprio autor (2023).

A figura 1 mostra a especificação o diagrama geral da solução proposta, com todos seus principais módulos e suas interfaces. Sendo centralizado no servidor que gerencia os sistemas de gerenciamento de usuários e tarefas/eventos. Recebendo e respondendo as requisições provenientes da interface, direcionando ao sistema devido que se comunica com o banco de dados, que, armazenará os dados dos usuários e suas tarefas e eventos.

4.2 Diagrama de Container

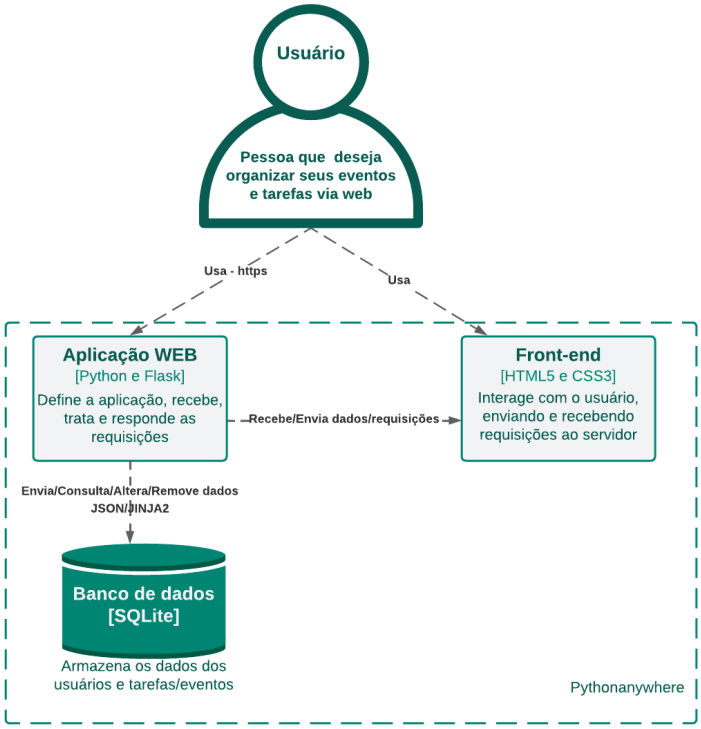

Figura 2 – Diagrama de container do sistema proposto. Fonte: Próprio autor (2023).

A figura 2 apresenta os *containers* da aplicação, sendo a aplicação web, onde se concentra os arquivos de programação com a linguagem Python e seu *microframework* Flask. Assim como, o *front-end*, que concentra os arquivos referentes ás páginas web, com arquivos HTML5 e *estilos* feitos com CSS3, além dos ícones usados nas páginas. Já a parcela referente ao banco de dados, é responsável pelo armazenamento dos dados, usando o banco relacional SQLite.

4.3 Diagrama de Componentes

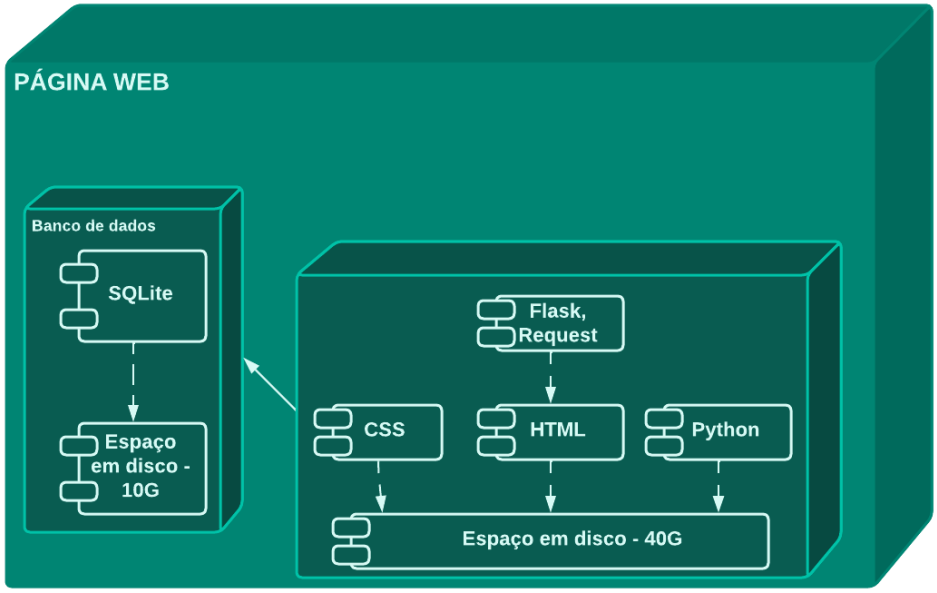

Figura 3 – Diagrama de Componentes do sistema proposto. Fonte: Próprio autor (2023).

O sistema foi desenvolvido na Linguagem Python, usando o *microframework* Flask, que permite a renderização e indexação de páginas HTML. Dessa forma, o Python importa a biblioteca Flask que constrói e inicializa a aplicação, indicando a rota e sua respectiva página html, alocada no servidor. A estética das páginas html são ajustadas via CSS.

Destaca-se que, toda interação entre dados (GET ou POST) é feito via Flask e Request, um dos módulos da linguagem. O Python possui bibliotecas nativas (instaladas nativamente) que permitem a conexão e manejo de banco de dados relacionais, no caso deste projeto, foi adotado o banco SQLite, alocado junto ao servidor. Todos os componentes são gratuitos e compatíveis entre si, apresentando funções e módulos internos que automatizam o desenvolvimento, abstraindo cada vez mais a complexidade dos elementos.

Portanto, conforme o diagrama apresentado na figura 3, as entidades participantes da solução são:

- Componente 1 – Espaço em disco, a aplicação necessita de espaço de memória para alocar os dados dos usuários do sistema, de suas tarefas e eventos. Assim como, se faz necessário memória para a hospedagem do serviço.
- Componente 2 – SQLite, o banco de dados relacional SQLite é um banco de dados que armazena os dados em modelo de tabelas, com colunas e linhas.

Sendo um banco compatível com a linguagem Python e de leve instalação e manuseio.

- Componente 3 – Linguagem Python na versão 3.9 ou superior, é a linguagem adotada para a construção do servidor, de instalação e uso gratuito, com módulos documentados, uma comunidade ativa e de desenvolvimento simples, frente a natureza de alto nível de abstração da linguagem.

- Componente 4 – HTML 5, é uma linguagem de marcação de texto, que permite a construção da interface interativa, que intermedia a comunicação entre o usuário e o servidor.

- Componente 5 – CSS 3, é a linguagem de folha de estilos usada para formatar a visualização de documentos HTML, permitindo a criação de designs mais avançados e sofisticados. Além de permitir que os usuários naveguem mais facilmente na página.

- Componente 6 – Flask é um framework de aplicação web de código aberto escrito em Python, que permite desenvolver aplicações web rapidamente e facilmente, pois oferece um conjunto de ferramentas prontas para uso. É possível criar uma página HTML usando o Flask, mas também é possível criar aplicações web mais complexas, como APIs e aplicações de banco de dados. Além disso, o Flask oferece uma variedade de recursos para aprimorar a experiência do usuário, tais como métodos de autenticação, sessões seguras, URLs amigáveis, temas, entre outros. De forma que, apenas poucos componentes são necessários para a construção de uma página amigável e funcional.

5. Prova de Conceito (PoC)

5.1 Integrações entre Componentes

A forma de comunicação entre os componentes é completamente detalhada à seguir. Foi utilizada ferramenta de prototipagem para o código da aplicação.

5.2 Código da Aplicação

Nesta seção você encontra indicado o padrão arquitetural C4, a estrutura de código da aplicação proposta.

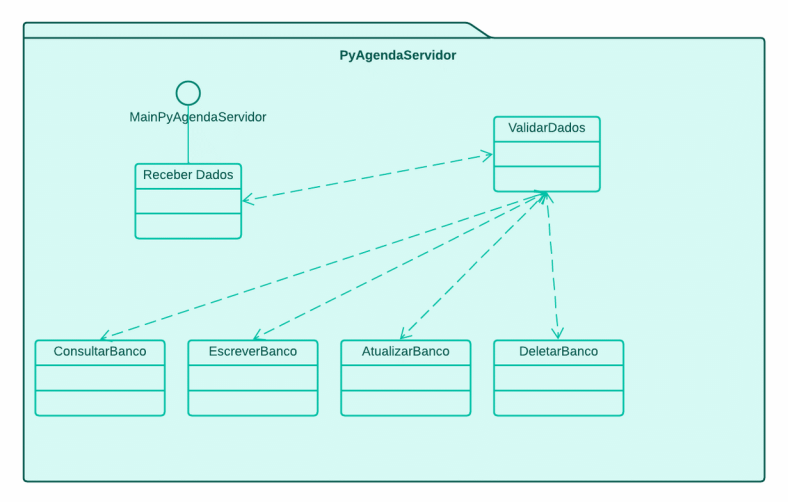

Figura 4 – Estrutura de código da aplicação do sistema proposto referente ao servidor. Fonte: Próprio autor (2023)

A estrutura da aplicação mostrada na figura 4 apresenta os componentes de código e suas funções no software implementado:

- ReceberDados (que recebe os dados presentes nas requisições, formulários html, dados do banco, dados validados, sendo estes tratados pelo servidor e encaminhados para o banco ou para o usuário);

- ValidarDados (que testa e valida os dados recebidos, por exemplo: se o campo e-mail foi preenchido de forma correta);

- ConsultarBanco (função que permite realizar consultas no banco de dados referenciado);

- EscreverBanco (função que permite realizar escritas no banco de dados referenciado);

- AtualizarBanco (função que permite realizar atualizações em tabelas no banco de dados referenciado);

- DeletarBanco (função que permite realizar remoções no banco de dados referenciado);

Neste sentido, foi feito um vídeo explicativo e disponibilizado n*esse link. O repositório com o código pode ser acessado no GitHub*.

Uma conta de teste pode acessar com o e-mail ‘projectaluno@gmail.com’ e senha ‘senha’

6. Avaliação da Arquitetura (ATAM)

A avaliação da arquitetura desenvolvida neste trabalho é abordada nesta seção visando avaliar se ela atende ao que foi solicitado pelo cliente, segundo o método ATAM.

6.1. Análise das abordagens arquiteturais

Para o sistema web de gerenciamento de tarefas feito em Python e Flask, o ATAM identifica três cenários principais:

Atributos de Quailidade	Cenários	Importância	Complexidade
Funcionalidade	Cenário 1: O sistema deve oferecer funções básicas de gerenciamento de tarefas/eventos, como criar, editar, excluir e listar tarefas/eventos.	A	M
Usabilidade	Cenário 2: O sistema deve ser intuitivo para usuários iniciantes, com uma interface limpa, intuitiva e fácil de usar.	M	B
Performance	Cenário 3: O sistema deve fornecer respostas rápidas aos usuários, manter os dados seguros e ser escalável para atender a um número crescente de usuários.	A	M

6.2. Cenários

Cenário 1 – Funcionalidade: O sistema deve oferecer funções básicas de gerenciamento de tarefas/eventos, como criar, editar, excluir e listar tarefas/eventos.

Cenário 2 - Usabilidade: O sistema deve ser intuitivo para usuários iniciantes, com uma interface limpa, intuitiva e fácil de usar.

Cenário 3 – Performance: O sistema deve fornecer respostas rápidas aos usuários, manter os dados seguros e ser escalável para atender a um número crescente de usuários.

6.3. Evidências da Avaliação

Atributo de Qualidade:	Funcionalidade
Requisito de Qualidade:	O sistema deve oferecer funções básicas de gerenciamento de tarefas/eventos, como criar, editar, excluir e listar tarefas/eventos.
Preocupação:	
O sistema deve coletar os dados, tratá-los e conectá-los com o banco de dados, associando corretamente com seu respectivo usuário.	
Cenário(s):	
Cenário 1	
Ambiente:	
Sistema em operação Windows 10. Navegador edge	
Estímulo:	
O sistema recebe uma requisição e recebe os dados e a operação solicitada, validando-os e gerando a resposta.	
Mecanismo:	
O *microframework* Flask recebe as requisições, o Request permite o recebimento dos dados em formato de dicionário, sendo possível tratar os dados e encaminhá-los ao banco de dados, após a validação destes.	
Medida de resposta:	
Retornar a página solicitada pelo cliente.	
Considerações sobre a arquitetura:	

Riscos:	Não há
Pontos de Sensibilidade:	Não há
Tradeoff:	Não há

Atributo de Qualidade:	Usabilidade
Requisito de Qualidade:	O sistema deve ser intuitivo para usuários iniciantes, com uma interface limpa, intuitiva e fácil de usar.
Preocupação:	
O sistema deve permitir navegação fluida e intuitiva, evitando poluir com elementos gráficos desnecessários ou não apresentar as opções mínimas para as tarefas necessárias, ou apresentar de forma ineficiente, que o cliente não deduza sua função.	
Cenário(s):	
Cenário 2	
Ambiente:	
Sistema em operação Windows 10. Navegador edge	
Estímulo:	
O sistema recebe uma requisição e recebe os dados e a operação solicitada, validando-os e gerando a resposta.	
Mecanismo:	
O sistema estará hospedado na plataforma Pythonanywhere, que será acessado por seu link, recebendo as requisições por meio do Flask e Request.	
Medida de resposta:	
Retornar a página solicitada pelo cliente.	
Considerações sobre a arquitetura:	
Riscos:	Poluição gráfica por excesso de elementos ou ineficiência destes.
Pontos de Sensibilidade:	Não há
Tradeoff:	Não há

Atributo de Qualidade:	Performance
Requisito de Qualidade:	O sistema deve fornecer respostas

	rápidas aos usuários, manter os dados seguros e ser escalável para atender a um número crescente de usuários.
Preocupação:	
O sistema deve ser acessado com delay de no máximo 2 segundos, para não comprometer a experiência do usuário.	
Cenário(s):	
Cenário 3	
Ambiente:	
Sistema em operação Windows 10. Navegador edge	
Estímulo:	
O sistema recebe uma requisição e recebe os dados e a operação solicitada, validando-os e gerando a resposta.	
Mecanismo:	
O sistema estará hospedado na plataforma Pythonanywhere, que será acessado por seu link, recebendo as requisições por meio do Flask e Request. Esse processo não deve ultrapassar os 2 segundos.	
Medida de resposta:	
Retornar a página ou operações solicitadas pelo cliente.	
Considerações sobre a arquitetura:	
Riscos:	Delay proveniente de instabilidades na rede, perda de pacotes ou acesso via rede wireless.
Pontos de Sensibilidade:	Não há
Tradeoff:	Não há

Os testes feitos podem ser conferidos conforme as figuras apresentam, de forma que, nas Figuras 5, 6, 7, 8 e 9.

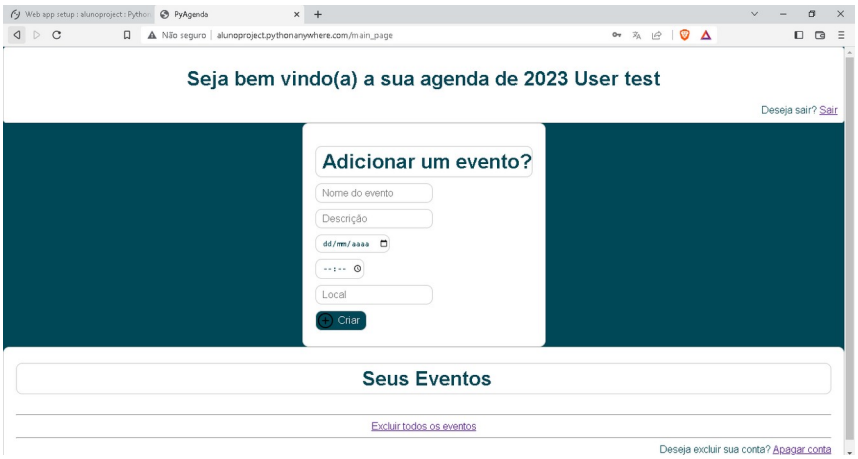

Figura 5 – Tela principal do usuário sem tarefas/eventos. Fonte: Próprio autor (2023).

Como visto, a página principal evita poluição gráfica, com descrições mínimas, tornando a terefa de criação de tarefas e eventos simples.

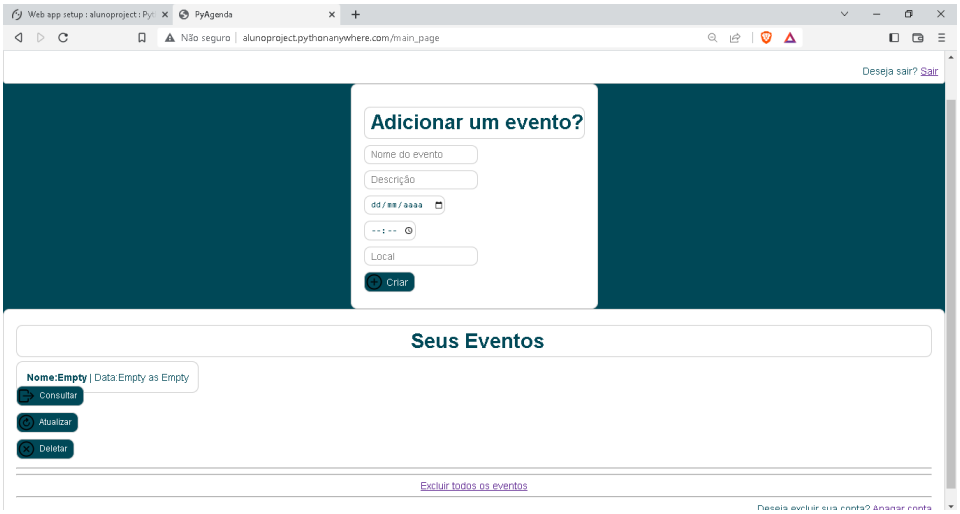

Figura 6 - Tela principal do usuário após criar um evento/tarefa. Fonte: Próprio autor (2023).

Na imagem a cima, é visto que as tarefas criadas ficam listadas, sendo possível sua consulta, atualização ou exclusão.

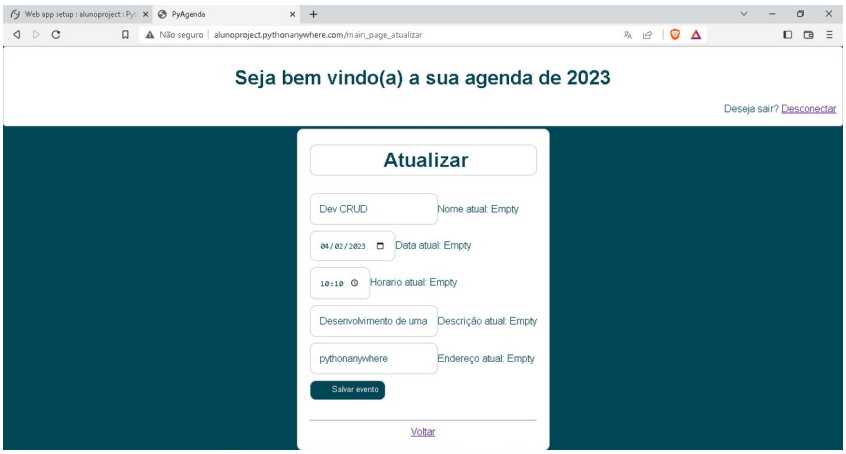

Figura 7 - Tela de atualização de tarefas/eventos. Fonte: Próprio autor (2023).

O processo de atualização é feito selecionando a opção e preenchendo os campos com os novos dados, caso não sejam preenchidos, serão mantidos os dados atuais.

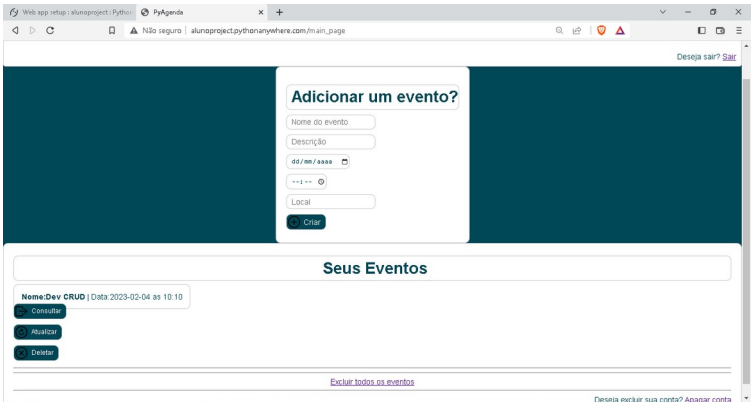

Figura 8 - Tela principal do usuário após atualizar tarefas/eventos. Fonte: Próprio autor (2023).

A consulta é feita de forma semelhante à atualização, selecionando esta opção, sendo direcionado à página com as informações da tarefa/evento, como visto na figura

Figura 9 - Tela de consulta de dados de tarefas/eventos. Fonte: Próprio autor (2023).

6.4. Resultados Obtidos

O sistema web de gerenciamento de tarefas desenvolvido em Python e Flask foi projetado para ser usado por usuários autenticados. O sistema oferece uma interface intuitiva, com a capacidade de criar, editar, excluir e gerenciar tarefas/eventos e suas informações associadas.

Além disso, o sistema fornece um mecanismo de autenticação de usuários para garantir que apenas usuários autorizados possam acessar as tarefas, sendo necessário criar e cadastrar uma conta para acessar. O sistema foi testado com sucesso e demonstrou cumprir os requisitos não funcionais, conforme a tabela abaixo apresenta:

Requisitos Não Funcionais	Teste	Homologação
RNF01: O sistema deve ser apresentar disponibilidade 24 X 7 X 365;	OK	OK
RNF02: O sistema deve apresentar desempenho de no máximo 2 segundos;	OK	OK
RNF03: O sistema deve ter mecanismos de autenticação de usuários para restringir o acesso a dados confidenciais;	OK	OK
RNF04: O sistema deve ter backups regulares para garantir a integridade dos dados;	OK	OK
RNF05: O sistema deve possuir uma interface simples e intuitiva;	OK	OK
RNF06: O sistema deve ser capaz de se adaptar ao crescimento do número de usuários;	OK	OK
RNF07: O sistema deve possuir documentação clara e de fácil compreensão para facilitar a manutenção.	OK	OK

7. Avaliação Crítica dos Resultados

O Python tem uma sintaxe simples, que permite que qualquer programador possa começar a trabalhar rapidamente com o código. Ele também possui uma ampla comunidade de usuários que ajudam os desenvolvedores a resolver problemas e fazer melhorias. Semelhantemente, o Flask é um framework web leve, ágil e extensível. Ele também fornece várias ferramentas para ajudar a desenvolver e implantar aplicativos web complexos.

Contudo, foi visto que o Flask é um excelente framework web, mas também tem seus limites. Ele não oferece muitas ferramentas avançadas para desenvolver aplicativos web complexos. Assim como, a plataforma adotada para hospedar o sistema, a PythonAnywhere, é uma plataforma de hospedagem de aplicativos web eficiente, mas não oferece muitas opções de personalização, o que pode ser um problema para usuários avançados, limitando a complexidade da proposta.

8. Conclusão

Conclui-se com este projeto que, desenvolver um sistema web com Python e Flask exige consideração de todos os aspectos de arquitetura necessários para um sistema de gerenciamento de tarefas eficaz. De forma que, é importante compreender os trade-offs entre desempenho, custo, segurança e outras questões para otimizar o desempenho do sistema.

Assim como, conhecer as características da plataforma Pythonanywhere para configurar o ambiente de hospedagem de forma apropriada. Sendo fundamental dimensionar o ambiente de acordo com o volume de tráfego esperado no sistema e entender as limitações da plataforma para garantir que o sistema possa funcionar de forma eficaz, considerando as limitações da plataforma na modalidade gratuita.

Foi visto que é importante testar e monitorar o desempenho do sistema para garantir que este seja adequado, atendendo aos requisitos estabelecidos. Entre os quais, destaca-se as medidas de segurança adequadas para proteger o sistema e seus usuários.

Contudo, é evidente a necessidade de ajustes e melhorias, entre as quais cita-se a implementação de um sistema de integração com outras plataformas, como o Google Calendar ou o Slack, assim como, implementação de um sistema de notificação push para lembrar os usuários de tarefas com datas próximas. Neste sentido, lista-se as principais lições aprendidas com a execução deste projeto:

1. Como configurar um servidor web para hospedar um aplicativo Python/Flask na plataforma PythonAnywhere;
2. Como escrever código Python para criar um sistema de gerenciamento de tarefas;
3. Como conectar o banco de dados do aplicativo ao servidor web;
4. Como criar formulários HTML para interagir com o usuário;
5. Como usar a biblioteca Flask para criar rotas e manipular os dados do usuário;
6. Como definir as regras de acesso ao aplicativo;
7. Como implementar a autenticação e autorização de usuários;
8. Como monitorar e solucionar problemas no servidor web;

9. Como usar as ferramentas fornecidas pelo PythonAnywhere para monitorar e gerenciar aplicativos;
10. Como automatizar o processo de deploy.