**Status de Serviço** (MeAjudaAqui)

**Documento de Arquitetura de Software**

**Versão 1.3**

**Allan Breno Ferreira Pereira**

**Daniel de Andrade Teixeira**

**Danilo Siqueira Oliveira**

**Douglas Caldeira de Souza**

**Eduardo da Motta Saldumbides**

**Gabriel Barbedo Fernandes**

**Glauber Guimarães Batista Silveira**

**Gustavo da Silva Santos**

**Leonardo Menezes**

**Paulo Augusto de Macena Pereira**

**Histórico da Revisão**

| **Data** | **Versão** | **Descrição** | **Autor** |
| --- | --- | --- | --- |
| 19/04/2022 | 1.0 | Início do projeto | Gustavo Santos |
| 20/04/2022 | 1.0 | Criação dos Repositórios | Paulo Augusto |
| 01/06/2022 | 1.0 | Organização do Documento | Gustavo Santos |
| 08/06/2022 | 1.0 | Atualização de Escopo, Metas, Requisitos e  e Visões da Arquitetura | Grupo Completo |
| 10/06/2022 | 1.1 | Diagrama de Classe Status Service | Paulo, Eduardo e Douglas |
| 10/06/2022 | 1.1 | Diagrama de Classe Telegram Service | Paulo e Eduardo |
| 10/06/2022 | 1.1 | Diagrama de Classe User Service | Gustavo Santos |
| 11/06/2022 | 1.1 | Diagrama de Caso de Uso | Paulo, Eduardo e Leonardo |
| 11/06/2022 | 1.1 | Diagrama de Sequência (Cadastrar Usuário) | Paulo e Gabriel Barbedo |
| 11/06/2022 | 1.1 | Código Telegram Service | Paulo Augusto e Eduardo |
| 11/06/2022 | 1.1 | Código Status Service | Paulo Augusto e Douglas |
| 11/06/2022 | 1.1 | Modelo Arquitetural Projeto | Paulo e Leonardo |
| 12/06/2022 | 1.1 | Desenvolvimento Back-end | Paulo e Gustavo |
| 12/06/2022 | 1.1 | Desenvolvimento Front-end Inicial | Danilo e Guilherme |
| 13/06/2022 | 1.2 | Mecanismos de Arquitetura | Gabriel Barbedo |
| 13/06/2022 | 1.2 | Visões de Arquitetura | Gabriel Barbedo |
| 14/06/2022 | 1.2 | Qualidade | Gabriel Barbedo |
| 15/06/2022 | 1.2 | Ajustes | Grupo Completo |
| 15/06/2022 | 1.2 | Atualização da Documentação | Eduardo e Allan Breno |
| 15/06/2022 | 1.2 | Formatação da Documentação | Eduardo e Allan Breno |
| 15/06/2022 | 1.2 | Revisão da Documentação | Eduardo e Allan Breno |
| 12/07/2022 | 1.3 | 8.3.3 Ajuste 5 | Paulo |
| 12/07/2022 | 1.3 | 8.3.3 Ajuste 6 | Eduardo |
| 12/07/2022 | 1.3 | 8.3.3 Ajuste 3 | Douglas |
| 12/072022 | 1.3 | Orquestrador Report Service | Paulo e Eduardo |
| 12/07/2022 | 1.3 | API Gateway | Paulo e Douglas |
| 13/07/2022 | 1.4 | Camadas Arquiteturais | Paulo e Glauber |
| 13/07/2022 | 1.4 | Estilos Arquiteturais | Grupo Completo |
| 13/07/2022 | 1.4 | Diagrama de Sequência (Registrar problemas em um serviço) | Allan Breno, Daniel e Glauber |
| 13/07/2022 | 1.4 | Diagrama de Sequência (Solicitar Relatório) | Allan Breno, Daniel e Glauber |

| 13/07/2022 | 1.4 | Ajuste dos casos de uso de acordo com as classes | Allan Breno, Daniel e Glauber |
| --- | --- | --- | --- |
| 13/07/2022 | 1.4 | Desenvolvimento Front-end Final | Paulo Augusto |

**Índice Analítico**

1. Introdução

1.1 Finalidade

1.2 Escopo

2. Metas e Restrições da Arquitetura

3. Suposições e Dependências

4. Requisitos Arquiteturalmente Significativos

5. Decisões, Restrições e justificativas

6. Mecanismos Arquiteturais

1. Persistência
2. Comunicação
3. Tratamento de Exceções

7. Camadas da Arquitetura

8. Visões da Arquitetura

9. Qualidade

10. Telas Interface Web

**Documento de Arquitetura de Software**

1. **Introdução**
   1. **Finalidade**

O sistema tem por objetivo geral auxiliar os usuários a verificar o status em tempo real de serviços e aplicações na web.

* 1. **Escopo**

O sistema tem propósito, mostrar um conteúdo dinâmico onde veremos os últimos casos de instabilidades/falhas e ou serviços e aplicações mais pesquisados no momento. O sistema deve permitir que qualquer usuário navegue por todas as páginas, e consiga buscar e selecionar o serviço e/ou aplicação que deseja verificar o status. O usuário terá como opção ver o status atual, ver o status semanal e ou por determinado período que assim desejar.

1. **Metas e Restrições da Arquitetura**

O sistema irá possibilitar o usuário reportar caso tenha passado por uma instabilidade ou problemas relacionados, efetuando seu cadastro, para assim efetuar o seu reporte. Ao realizar o cadastro no sistema, o usuário terá a opção de criar reportes indicando título, aplicação e uma breve descrição do ocorrido, além, da possibilidade de anexar prints, por fim poderá também marcar para receber um e-mail a partir do momento que for detectado uma normalidade no sistema.

O sistema deverá permitir a geração de relatórios específicos para cada aplicação/serviço, relatórios por períodos e relatórios por tipos de falhas. A implementação do sistema trará aos usuários um melhor retorno para identificação de problemas, além de conter um histórico datado das falhas ocorridas detectadas.

Vale salientar que o sistema não entregará certeza absoluta pois funcionará em sua grande forma analisando relatórios em tempo real e também baseado em relatos de usuários

1. **Suposições e Dependências**

*[Liste as suposições e dependências que dirigem as decisões arquiteturais. Isto pode incluir áreas sensíveis ou críticas, dependências e interfaces com sistemas legado, a habilidade e experiência da equipe, a disponibilidade de recursos importantes, e assim por diante ]*

1. **Requisitos Arquiteturalmente Significantes**

*[Insira uma referência ou link para os requisitos que exploram aspectos relevantes da arquitetura.]*

Requisitos Funcionais:

* Cadastro de conta
* Login de usuário cadastrado
* Alteração de dados do usuário
* Reportar incidente
* Consulta de sistemas cadastrados
* Listagem de sistemas cadastrados
* Solicitar Relatórios
* Enviar mensagem no telegram

Requisitos Não Funcionais

* Funcionar nos principais navegadores (Google Chrome,Mozilla Firefox e Internet Explorer)
* Garantir pelo menos taxa de 99,9% de integridade nos dados
* Estar na Língua Portuguesa
* Conexão com o Banco de dados (Ainda não foi definido)
* Deve proteger dados do usuário (Nome, idade, IP e e-mail.)
* Funcionamento 24h por dia, durante os 7 dias da semana
* Autenticar conta

1. **Decisões, Restrições e Justificativas**

**5.1 Listas de Decisões**

**5.2 Lista de Restrições**

* Restringir que o usuário efetue um *report* inválido.
* Restringir que o usuário cadastre um e-mail inválido.
* Restringir que o usuário cadastre uma senha inválido.

1. **Mecanismos Arquiteturais**

**6.1 Persistência**

Os bancos de dados (Profile DB e Status DB) entregarão uma melhor persistência, garantindo integridade e confiabilidade nos dados trabalhados.

Para o Status DB será utilizado o [MongoDB](https://www.mongodb.com/), já que por armazenar em forma de documento e ser não relacional, torna mais eficiente a geração de relatórios, consultas e os próprios inserts.

Para o Profile DB será utilizado o [PostgreSQL](https://www.postgresql.org/), por ser apto a escalabilidades verticais, o que se torna interessante caso o número de usuários se torne maior do que o esperado. Também é um banco de dados relacionais que oferece forte suporte para a funcionalidade NoSQL. Permitindo que os usuários definam seus próprios tipos de dados e o sistema de gerenciamento de banco de dados. PostgreSQL oferece suporte a ferramentas adicionais, tanto gratuitas quanto comerciais.

**6.2 Comunicação**

As comunicações com os micro-serviços vão ser através de uma comunicação REST, fazendo a comunicação enviando e recebendo um JSON conseguimos tratar de uma forma melhor os dados.

**6.3 Tratamento de Exceções**

O tratamento de exceções será responsável por tratar as execuções de interações diferentes envolvendo os usuários e o sistema.

1. **Camadas da Arquitetura**

**Diagrama

Descrição gerada automaticamente**

* 1. **Estilos Arquiteturais** 
     1. **Facade**

O padrão facade – em português, fachada – foi usado com o objetivo de simplificar mais a coleta de dados pelas classes Service, escondendo a complexidade do código real através de classes úteis.

**7.1.2 Rest**

Utilizamos REST na comunicação entre o Gateway e os nossos micros Serviços para podermos conseguir ter uma comunicação de uma forma fácil e conseguirmos ter tratar os dados da maneira mais segura possível

* + 1. **Micro-Serviços**

Utilizando a arquitetura de Micro-Serviços, nós conseguimos ter serviços de forma separada, para podermos ter soluções em diferentes tipos de códigos, pensando em produção, um micro-serviço pode ficar fora sem impactar todo o pacote.

* + 1. **Orchestrator**

O modelo arquitetural

1. **Visões da Arquitetura**

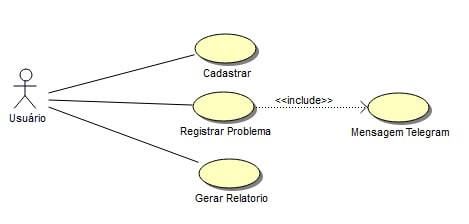
**8.1 Casos de Uso**

| Nome | Cadastro no sistema |
| --- | --- |
| Ator(es) | Usuário |
| Pré-condições | Plataforma funcionando |
| Pós-condições | Usuário cadastrado na plataforma |
| Fluxo principal | 1. Usuário acessa a opção "Cadastre-se" na tela  2. Usuário preenche os campos com seus dados  3. Usuário confirma seus dados  4. Usuário é cadastrado |
| Fluxo alternativo | 3.1 Usuário preenche algum campo com dados inválidos  3.2 Sistema apresenta a mensagem de que há dados inválidos e destaca o campo  3.3 Sistema retorna o usuário para o passo 2 do fluxo principal |

| Nome | Registrar problemas em um serviço |
| --- | --- |
| Ator(es) | Usuário |
| Pré-condições | Plataforma funcionando  Usuário cadastrado e logado Serviço cadastrado no sistema |
| Pós-condições | Registro concluído sobre incidente no serviço |
| Fluxo principal | 1 Usuário acessa a opção "Reporte um incidente"  2 Usuário fornece os dados relacionados ao serviço (nome do serviço, duração, tipo de incidente) e descreve o incidente  3 O sistema cadastra os dados do incidente do serviço  4 O sistema envia uma notificação via Telegram para um grupo |
| Fluxo alternativo 1 | 3.1 Usuário fornece os dados relacionados a um serviço não cadastrado no banco  3.2 Sistema apresenta a mensagem "Sistema não cadastrado"  3.3 Sistema retorna o usuário para o passo 2 do fluxo principal |
| Fluxo alternativo 2 | 3.1 Usuário preenche algum campo com dados inválidos  3.2 Sistema apresenta a mensagem de que há dados inválidos e destaca o campo  3.3 Sistema retorna o usuário para o passo 2 do fluxo principal |

| Nome | Solicitar relatório |
| --- | --- |
| Ator(es) | Usuário |
| Pré-condições | Plataforma funcionando  Usuário cadastrado e logado  Serviço cadastrado no sistema |
| Pós-condições | Registro concluído sobre incidente no serviço |
| Fluxo principal | 1 Usuário acessa a página do serviço desejado no sistema  2 Usuário solicita um relatório através da opção "Gerar relatório"  3 Um relatório é gerado e enviado para o email do usuário |
| Fluxo alternativo 1 | 3.1 Usuário acessa a página do serviço desejado no sistema  3.2 Usuário solicita um relatório através da opção "Gerar relatório"  3.3 Sistema apresenta a mensagem de que não há registros de incidentes. |

**8.1.2 Diagrama de Casos de Uso**

****

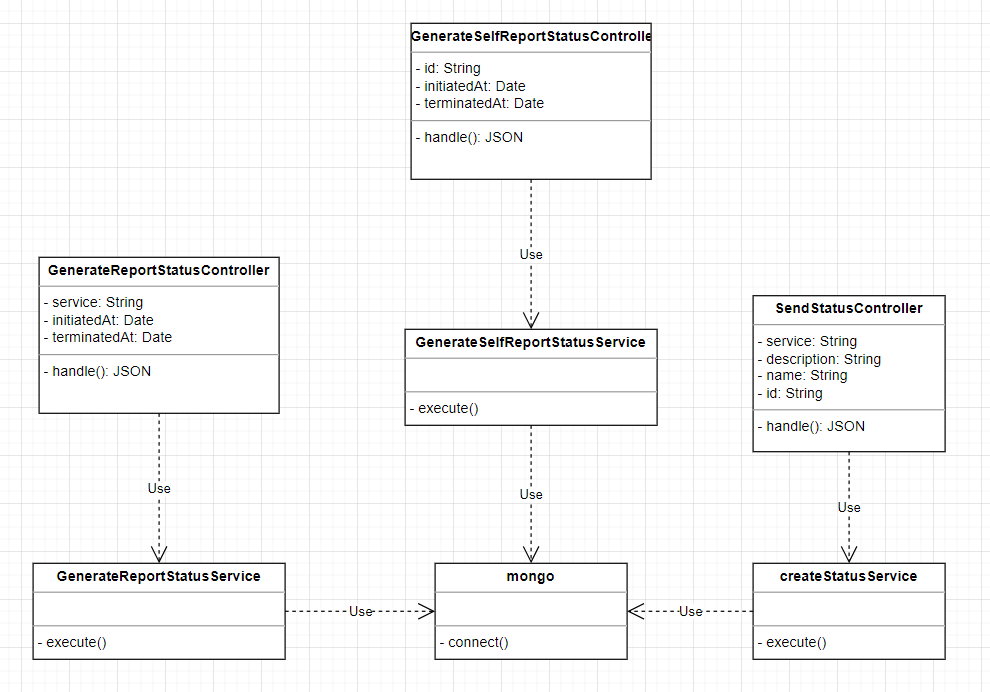
**8.3 Lógica**

**O sistema irá utilizar a arquitetura em micro serviços, os quais serão:**

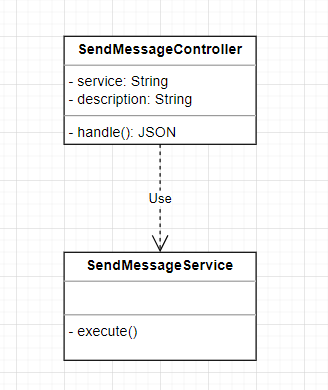
* + **User service :** Responsável por criar login para o usuário e autenticar o mesmo
  + **Status service :** Responsável por encaminhar o status para o sistema / também responsável pela busca de relatório
  + **Orquestrador de relatório :** Trata informações obtidas através do status service e utiliza um outro serviço para encaminhar as informações
  + **Telegram service:** Responsável por encaminhar as inserções do status service em um grupo no Telegram

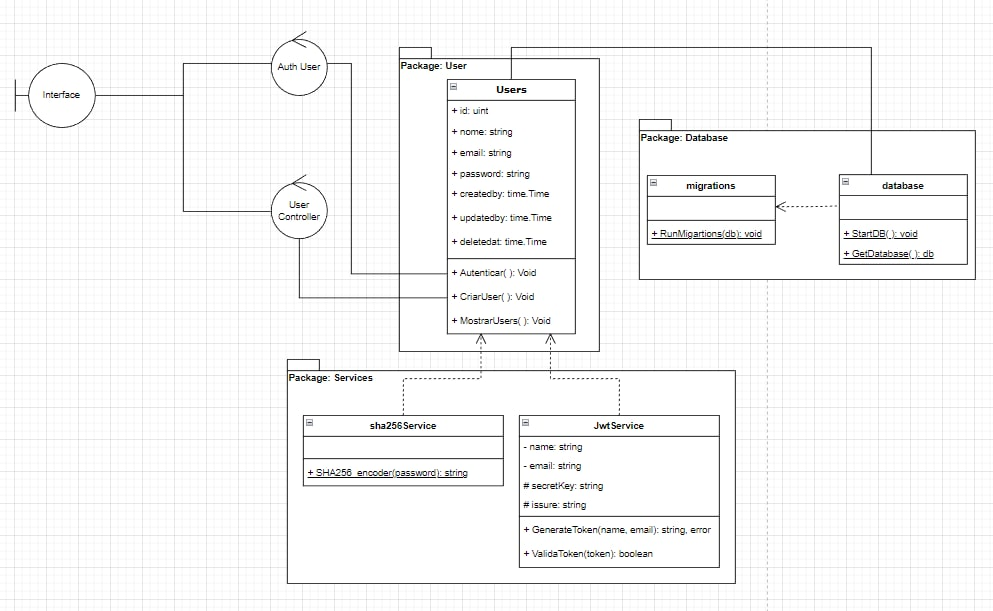
**8.3.1 Diagrama de Classes**

* **Status Service:** Serviço para inserir Status de serviços e utilizado pelo Orquestrador de Relatórios para pegar histórico dos serviços.

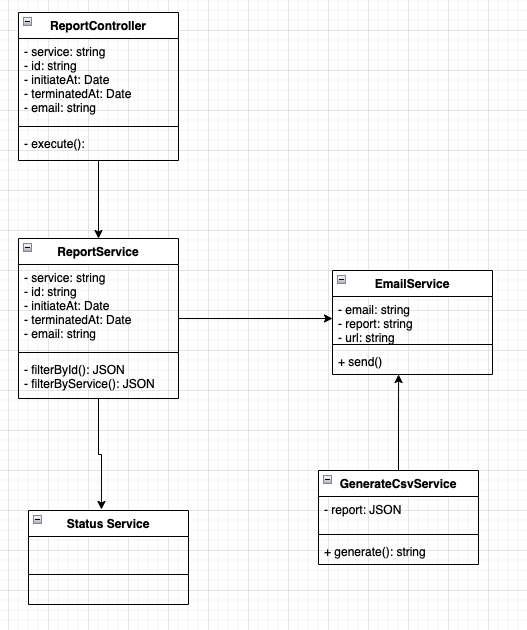
****

* **Telegram Service:** Responsável por enviar as notificações para o telegram

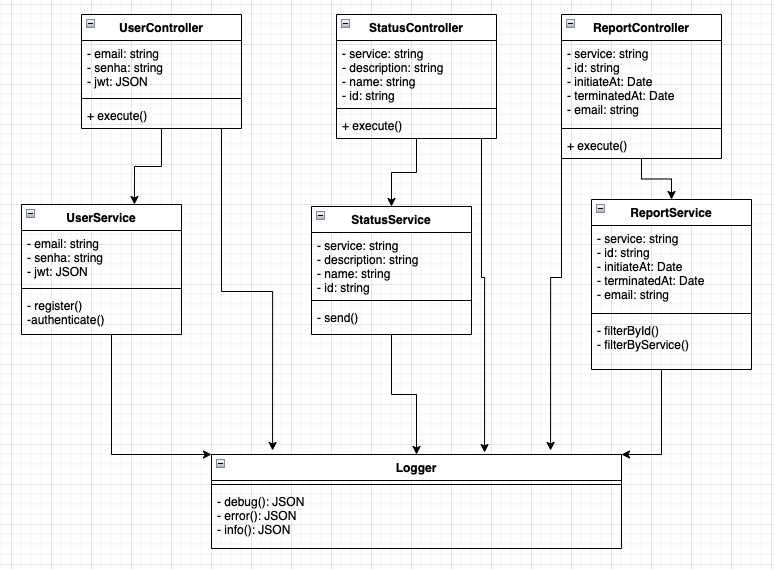
****

* [**User Service**](https://github.com/meajudaaqui/user-service)**:** Usado para cadastrar e autenticar os usuários

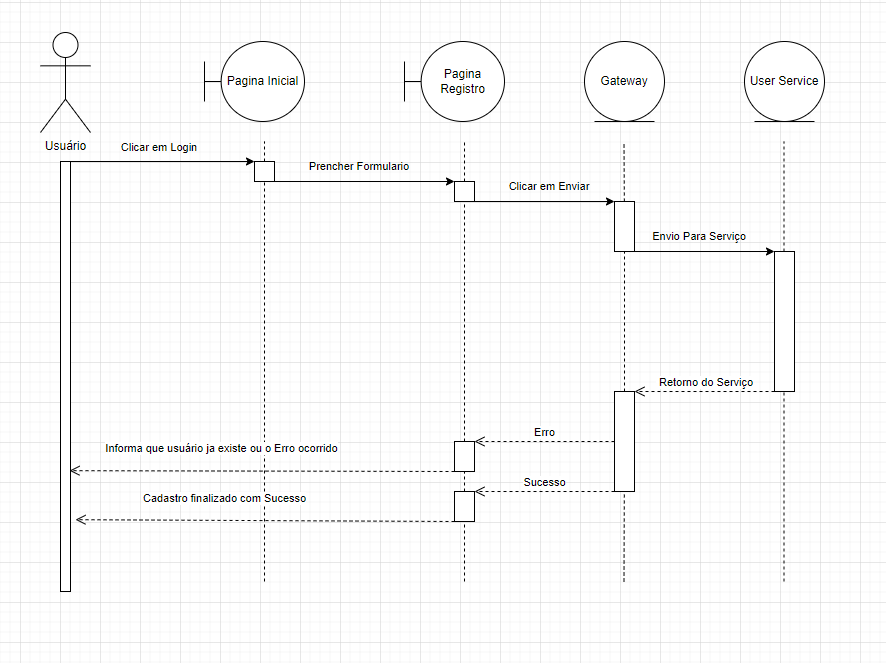
**- Orquestrador Report Service :** Gerar relatórios e enviar via email

****

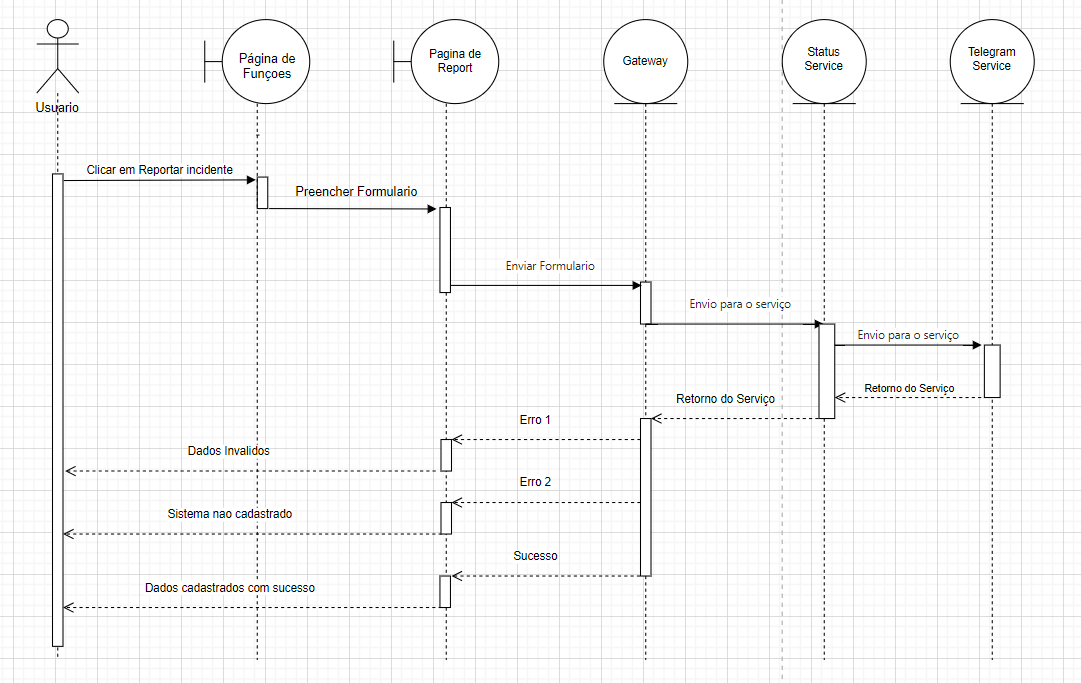
**-API Gateway:** Responsável por se comunicar com todos os micro-serviços.

****

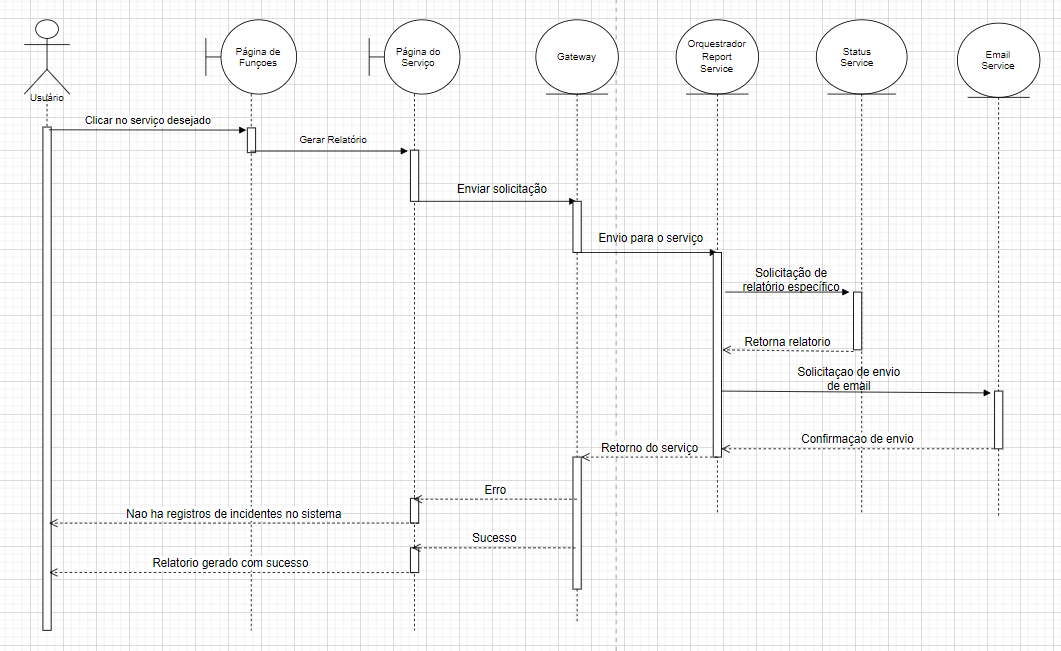
**8.3.2 Diagramas de Sequência**

**8.3.2.1 Cadastrar Usuário**

**8.3.2.2 Registrar um problema em Serviço**

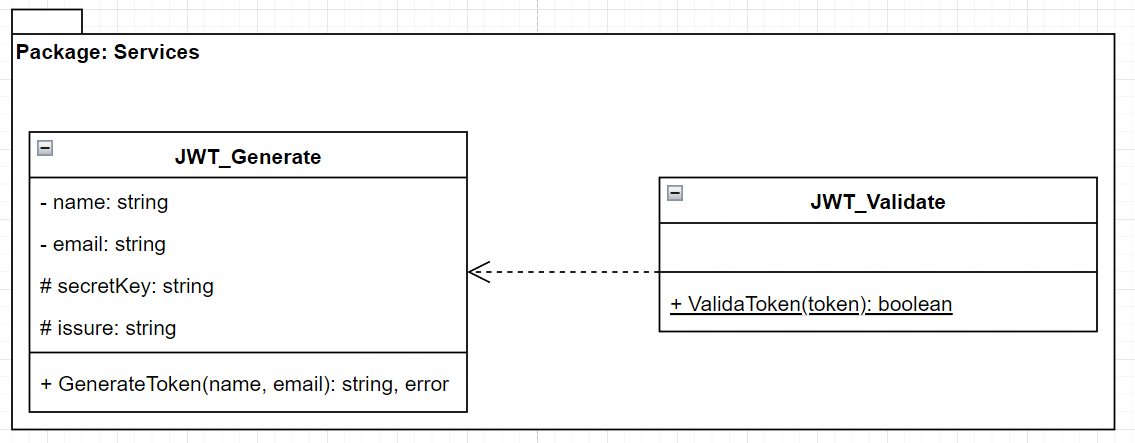
****

**8.3.2.3 Solicitar Relatório**

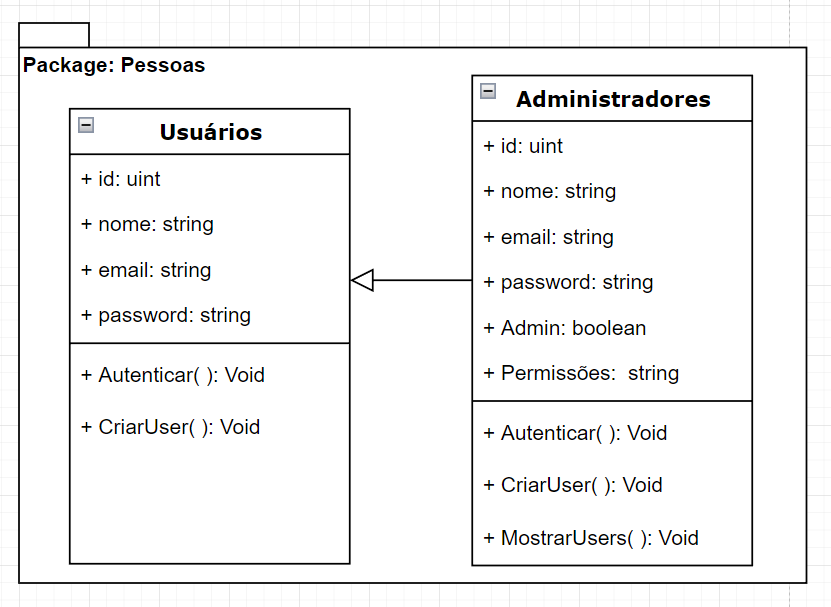
****

**8.3.3 Ajustes**

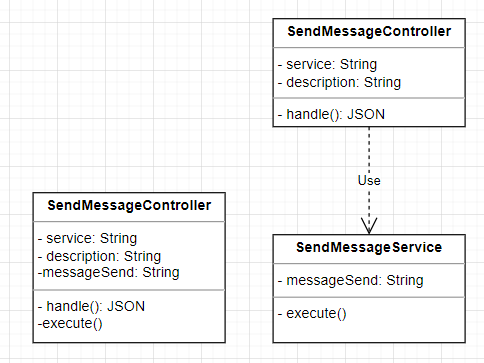
* **Ajuste 1:** Separação do serviço JWT em classe para gerar e outra classe para validar.  
  Motivação: Princípio da responsabilidade única, a classe que gera, seria chamada inutilmente sempre que fosse necessário validar, com isso, estaria quebrando o primeiro princípio da SOLID

****

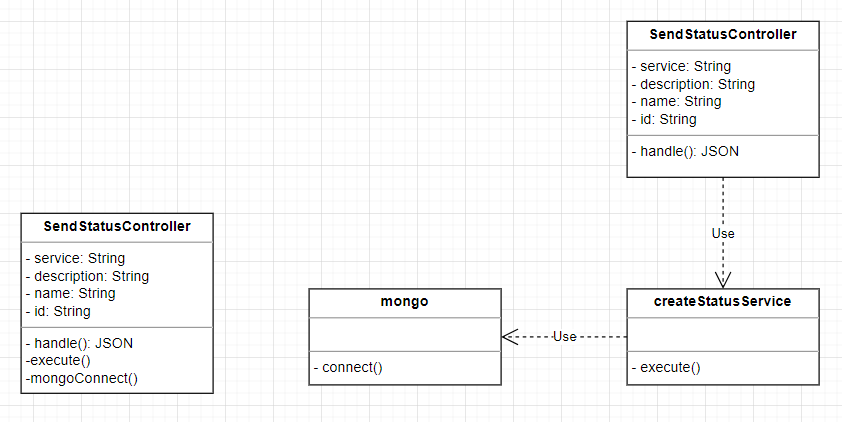
* **Ajuste 2:** Administradores como uma classe à parte, tendo como herança a classe usuários.  
  Motivação: Princípio da responsabilidade única, a classe Usuários recebendo usuários e administradores estaria quebrando o primeiro princípio da SOLID

****

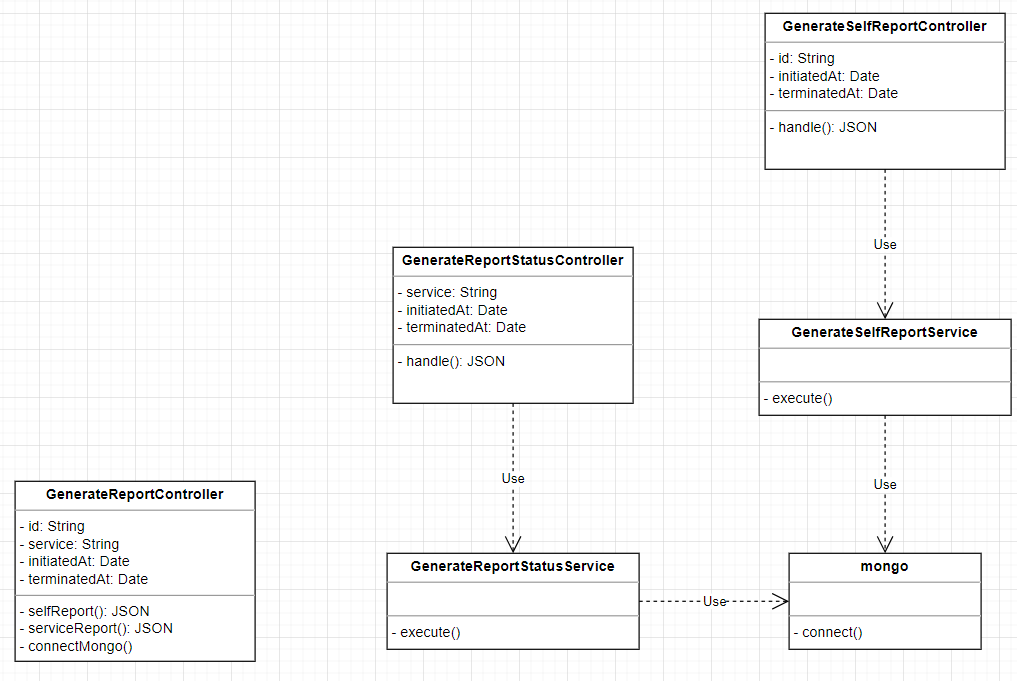
* **Ajuste 3:** Dividindo o SendMessageController e transformando em SendMessageController e SendMessageService.  
  Motivação: Trazer mais facilidade na hora da manutenção do código, trazendo mais confiabilidade e ajudando nos testes

****

* **Ajuste 4:** Retirando o mongoConnect() do controller para evitar criar várias sessões, criando o mongo com a função de connect para poder fazer a conexão com o banco e todo o serviço utiliza a mesma, separando o que for controle e service para ajudar em teste e melhorar o código

****

* **Ajuste 5:** Desacoplar funções para facilitar a manutenção de código, como no outro exemplo, isolando o mongoConnect para não ficar repetitivo

****

**- Ajuste 6:** No Telegram Service não utilizamos quase nada de SOLID apenas Princípio de Responsabilidade Única e Princípio de inversão de dependências, como ele era um micro-serviço simples e prático era a melhor forma de trabalhar com ele.

**Diagrama

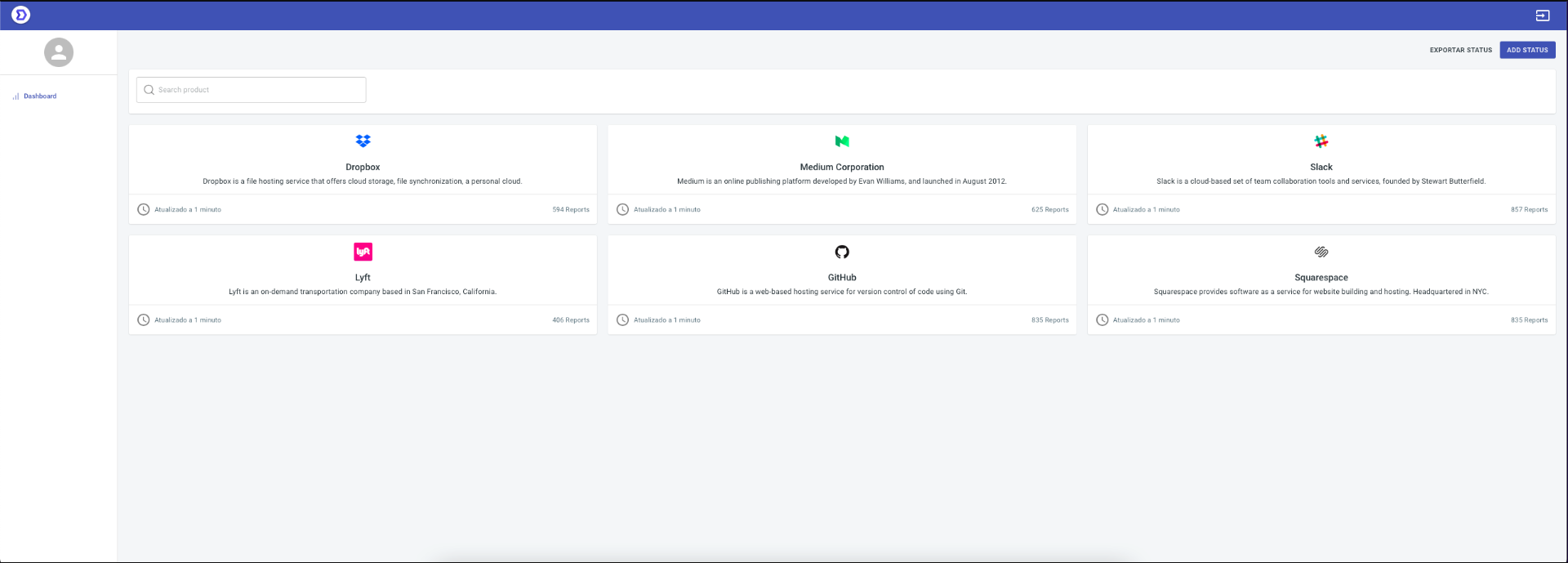
Descrição gerada automaticamente**

1. **Qualidade**

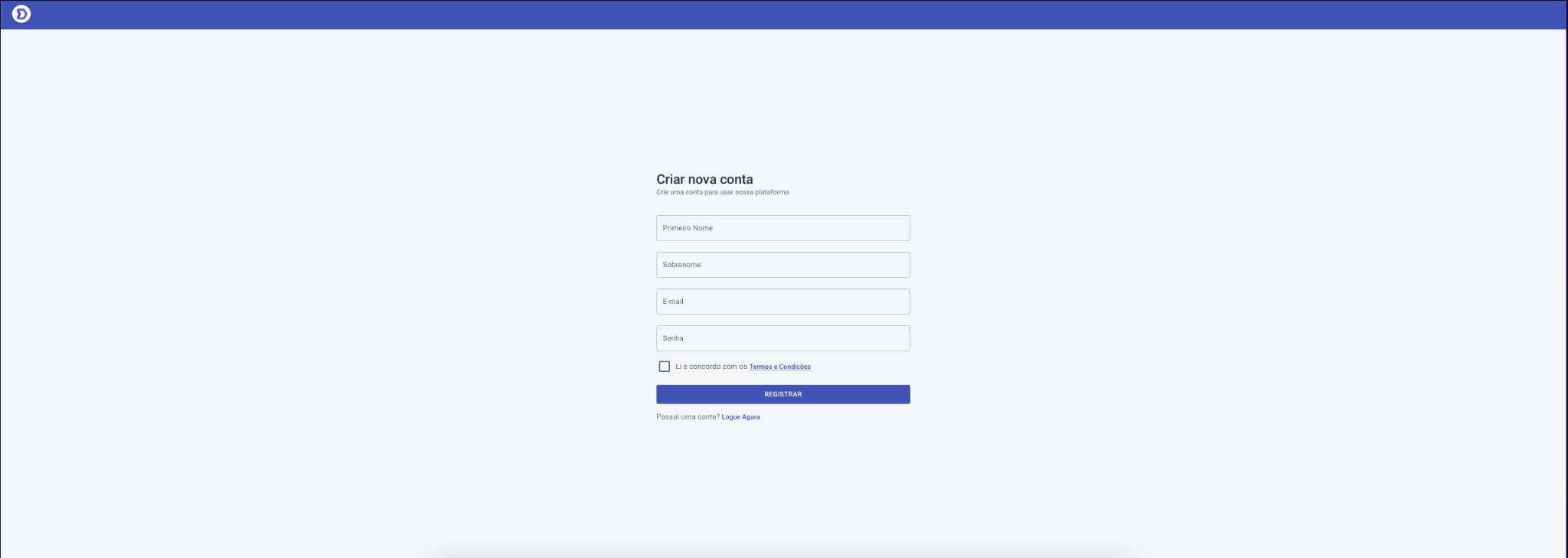
A arquitetura em micro serviços permite uma boa escalabilidade, flexibilizando a adaptação do sistema, tendo em vista que os deploys são feitos em máquinas virtuais organizadas de maneira independente. O que também valida uma manutenção e evolução dos serviços de maneira mais fluída, tendo em vista que o objetivo é individualizar funções de acordo com os serviços, permitindo consequentemente uma manutenção que não afete diretamente outras funcionalidades do sistema.

A arquitetura em questão também viabiliza uma maior flexibilidade de tecnologia, permitindo uma maior liberdade para que os desenvolvedores possam trabalhar com tecnologias distintas para atender cada uso de maneira mais adequada.

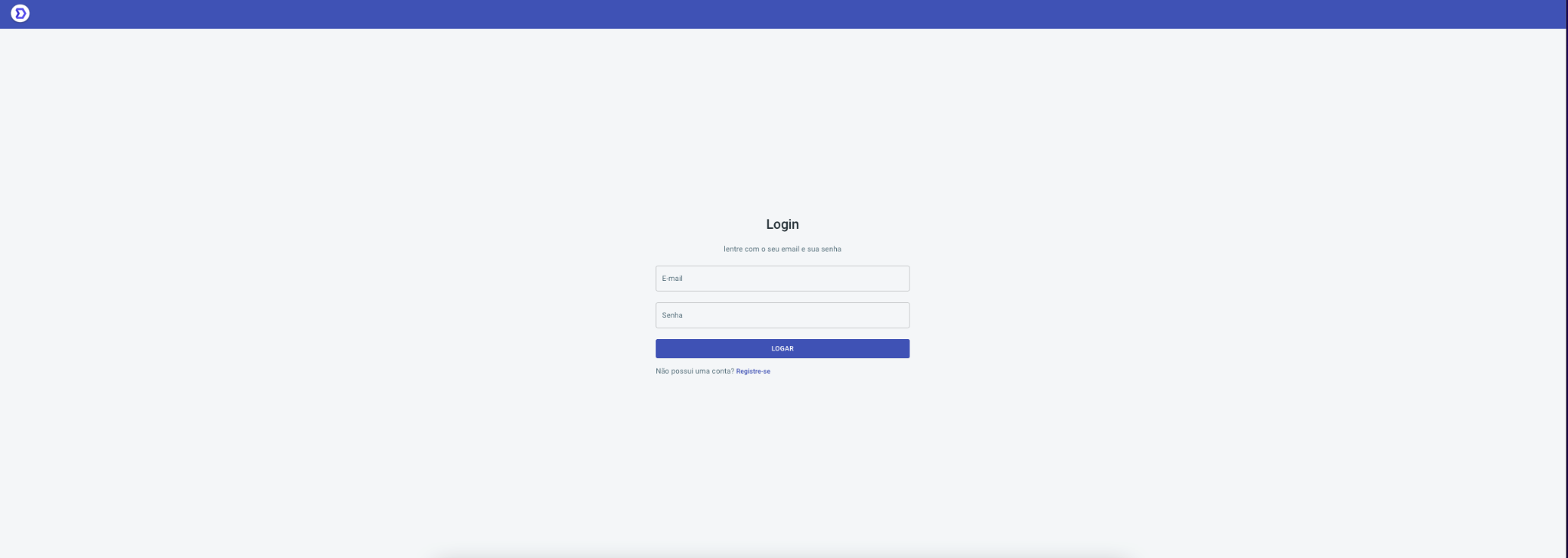
1. **Telas Interface Web**



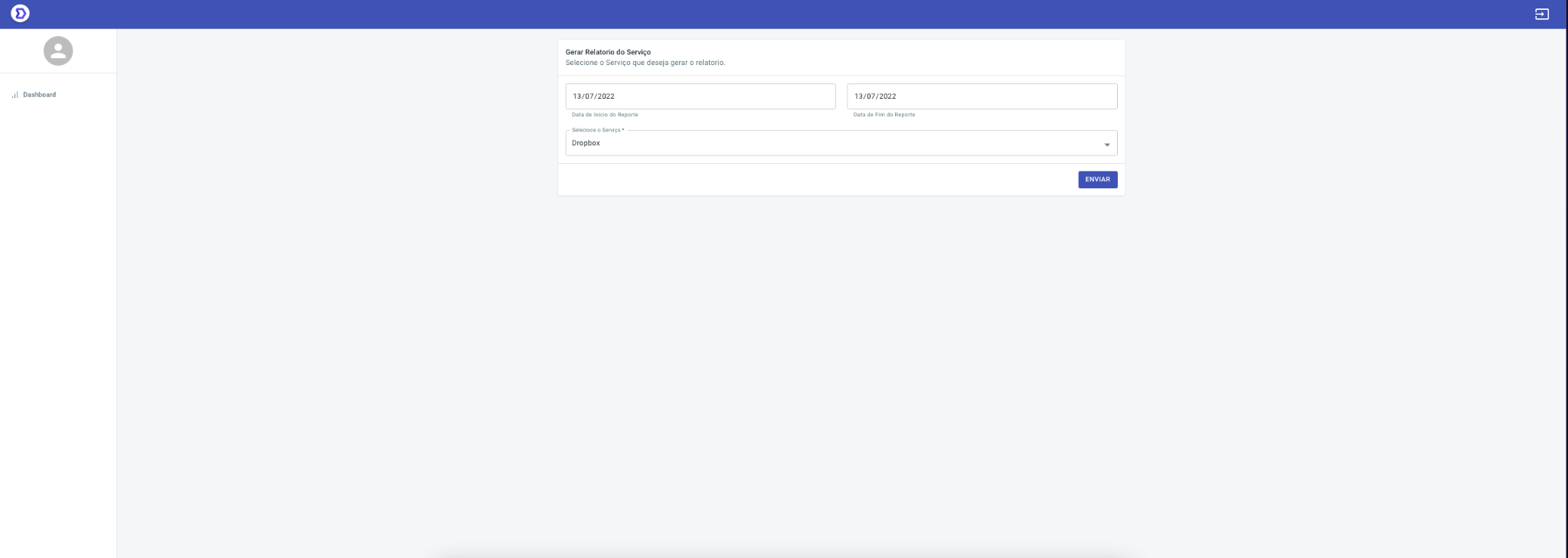
Tela Inicial



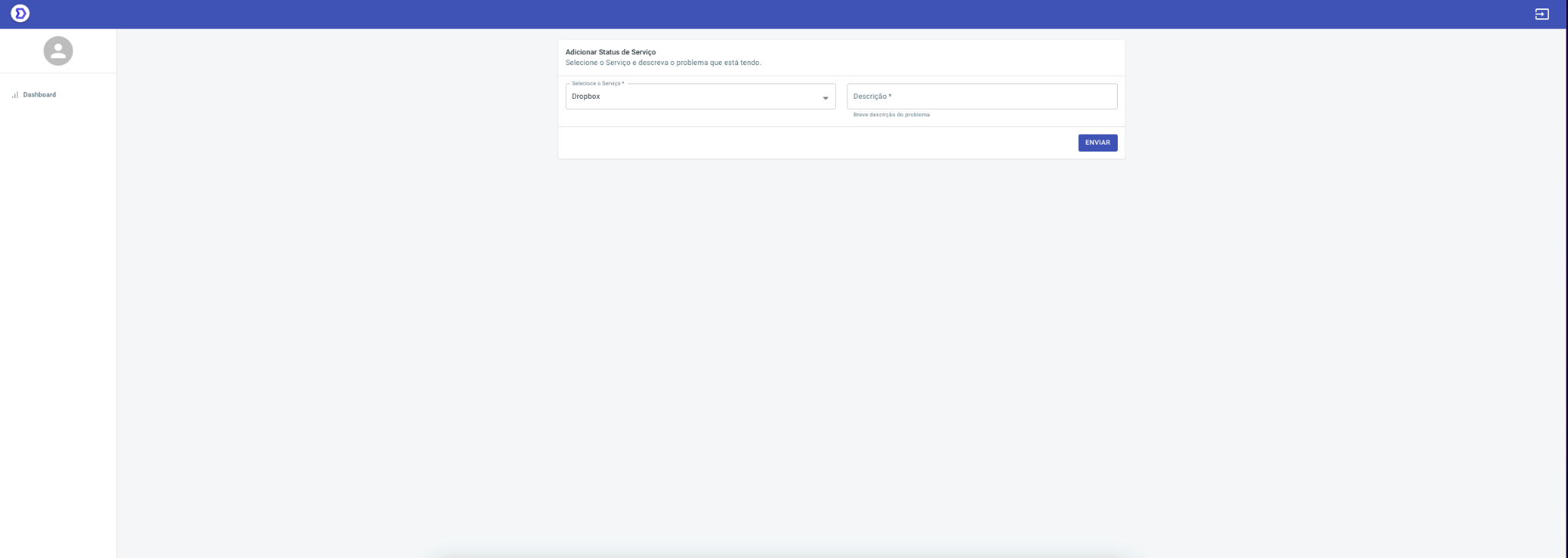
Tela de Criação de Conta



Tela de Login



Tela de Status de Serviço 1



Tela de Status de Serviço 2