Grupo 1

Is Down?

Documento de Arquitetura de Software

Versão 0.7

Histórico da Revisão

Data	Versão	Descrição	Autor
09/12/2021	0.1	Início do desenvolvimento da Introdução e Metas e Restrições da Arquitetura	Gabriel Inácio
21/01/2022	0.2	Modificação na descrição da arquitetura do sistema e desenvolvimento dos tópicos	Gabriel Inácio
25/01/2022	0.3	Revisão dos 6 primeiros tópicos e desenvolvimento dos próximos	Gabriel Inácio
26/01/2022	0.4	Padrões arquiteturais adicionados	Gabriel Inácio
28/01/2022	0.5	Modificação nos estilos arquiteturais	Gabriel Inácio
29/01/2022	0.6	Revisão de todo o documento, alteração do escopo, dos requisitos, inclusão de um tópico em qualidade, inclusão de um novo termo.	Rafael Lins
29/01/2022	0.7	Revisão de todo o documento, alteração nos estilos arquiteturais.	Ricardo Carletti

Índice Analítico

Documento de Arquitetura de Software

1. Introdução

Esse documento tem o objetivo de definir a visão arquitetural de um sistema de monitoramento de serviços da internet de nome “Is Down ?”.

1.1 Finalidade

Este documento oferece uma visão geral arquitetural abrangente do sistema, usando diversas visões arquiteturais para representar diferentes aspectos do sistema. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema.

1.2 Escopo

A arquitetura do projeto é fundamentada para suprir a necessidade de fornecer em tempos reais as disponibilidades de diversos serviços da internet para os usuários, bem como o monitoramento, solicitação de monitoramento de novos serviços previamente não contemplados. A fim de mitigar o impacto que problemas nesses serviços podem causar. Além de permitir que usuários possam adquirir os dados dessa aplicação para fazer testes e pesquisas com aplicações próprias.

1.3 Termos

A tabela a seguir define os termos usados ao longo do documento e seu significado.

Termo	Significado
SGBD	Acrônimo de Sistema de Gerenciamento de Banco de Dados.
API	Acrônimo de *Application Programming Interface*.
Query	Palavra “Consulta” em inglês.
Back End	Parte da aplicação que não é diretamente acessada pelo usuário. Tipicamente responsável pelo acesso ao banco de dados.
Front End	Parte da aplicação que o usuário usa diretamente.
Twitter	Twitter é uma rede social e um serviço de microblog.
Tags	Etiqueta que mostra o estado atual de um determinado serviço

2. Metas e Restrições da Arquitetura

2.1 Motivação

Diversos serviços da internet, como redes sociais e correios eletrônicos, sofrem com períodos de instabilidade e inoperabilidade. Assim, é muito comum que usuários dessas aplicações não saibam se o problema ocorre exclusivamente com ele – podendo ser um problema com o dispositivo e/ou sua conexão com a internet – ou é um problema do serviço.

Além disso, existem também os usuários corporativos, que dependem parcialmente ou totalmente de um ou mais desses serviços. Logo, é fundamental que saibam quais serviços são mais estáveis.

Portanto, com o objetivo de solucionar esse problema, o “Is Down ?” busca monitorar os principais serviços da internet e oferecer ferramentas capazes de mitigar o impacto que problemas nesses serviços podem causar.

2.2 Metas e Restrições

2.2.1 A solução deve ser robusta

Como o objetivo da aplicação é monitorar os serviços e ajudar nossos usuários quando um problema ocorrer, a solução deve sempre estar disponível. Assim, deverão ser adotadas medidas para que isso seja possível.

2.2.2 A coleta de dados deverá ser regular e assíncrona

Um período de instabilidade pode ocorrer a qualquer momento, logo os serviços deverão ser monitorados regularmente e ações automáticas disparadas sempre que um problema for detectado.

2.2.3 Deverão ser usadas mais de uma fonte de dados

A solução deverá utilizar diversas fontes de dados para coletar reclamações de usuários. A dependência de uma única fonte de dados poderá comprometer a disponibilidade do sistema e a confiabilidade dos dados.

2.2.4 A solução poderá ser integrada a outros sistemas via API

Um dos objetivos do “Is Down ?” é permitir que outros sistemas utilizem dos dados coletados por nossa solução.

3. Suposições e Dependências

3.1 Introdução

Em uma pesquisa realizada pela nossa equipe em busca de soluções que realizam o monitoramento de serviços da internet e o comportamento de usuários em redes sociais, foi definido que o sistema usará APIs para analisar comentários e publicações em redes sociais para identificar e registrar reclamações.

3.2 API do *Twitter*

A principal fonte de dados para monitoração dos serviços será a API do *Twitter*. Para usá-la, o sistema deverá ser capaz de realizar requisições HTTP e tratamento de objetos JSON.

4. Requisitos Arquiteturalmente Significantes

4.1 Requisitos Funcionais

RF1 - O usuário externo deverá obter um gráfico com o histórico do comportamento dos serviços monitorados.
RF2 - O usuário externo deverá ser informado do status dos serviços por meio de tags.
RF3 - O usuário externo deverá ter acesso a um catálogo dos serviços monitorados.
RF4 - O usuário externo poderá se cadastrar para ter acesso a privilégios de usuário interno.
RF5 - O usuário interno poderá solicitar que um novo serviço seja monitorado.
RF6 - O administrador poderá cadastrar, editar e excluir serviços.
RF7 - O administrador poderá cadastrar, editar e excluir administradores.
RF8 - O sistema deverá ter uma função de notificar usuários internos de problemas nos serviços monitorados.

4.2 Requisitos Não Funcionais

RNF1 - O sistema deverá usar o SGBD PostgreSQL.
RNF2 - O sistema deverá usar o framework Hibernate.
RNF3 - Períodos de manutenção só podem ser realizados fora do horário comercial.
RNF4 - A interface web deverá ser responsiva.
RNF5 - O monitoramento dos serviços deverá manter um mínimo de 48 horas de histórico.

5. Decisões, Restrições e Justificativas

5.1 O sistema deverá usar mais de uma representação do mesmo objeto

O objetivo da solução é monitorar serviços da internet e possibilitar o acesso dos dados coletados por duas formas: Interface Web e API. Assim, o objeto que será usado para persistência de dados será diferente do objeto retornado pela API, mesmo que representem o mesmo objeto abstrato.

5.2 Abstração do acesso à base de dados

As classes usadas pela Interface Web e API deverão usar classes que representem os registros da base de dados. Em nenhuma circunstância essas classes poderão fazer *queries* diretamente a base de dados.

O objetivo dessa restrição é diminuir o acoplamento de camadas usadas diretamente pelos usuários da camada de persistência de dados.

6. Mecanismos Arquiteturais

6.1 Correio

Uma das funcionalidades críticas da ferramenta é poder enviar mensagens aos usuários de maneira automática.

- **Latência**: O intervalo de tempo entre um evento e a ação automática de envio de mensagens deve ser otimizado.
- **Confiabilidade**: O número de mensagens que não chegam aos usuários deve ser o mínimo possível.
- **Volume**: O sistema deverá suportar um grande número de mensagens enviadas simultaneamente.

6.2 Gestão de recursos

A solução deve armazenar uma grande quantidade de dados sobres os serviços monitorados por longos períodos de tempo, assim, deverão ser utilizados recursos para mitigar o uso excessivo da base de dados.

- **Manutenção**: Dados antigos devem ser excluídos ou consolidados regularmente.

6.3 Gráficos

Deverão ser utilizados gráficos como recurso de apresentação dos dados.

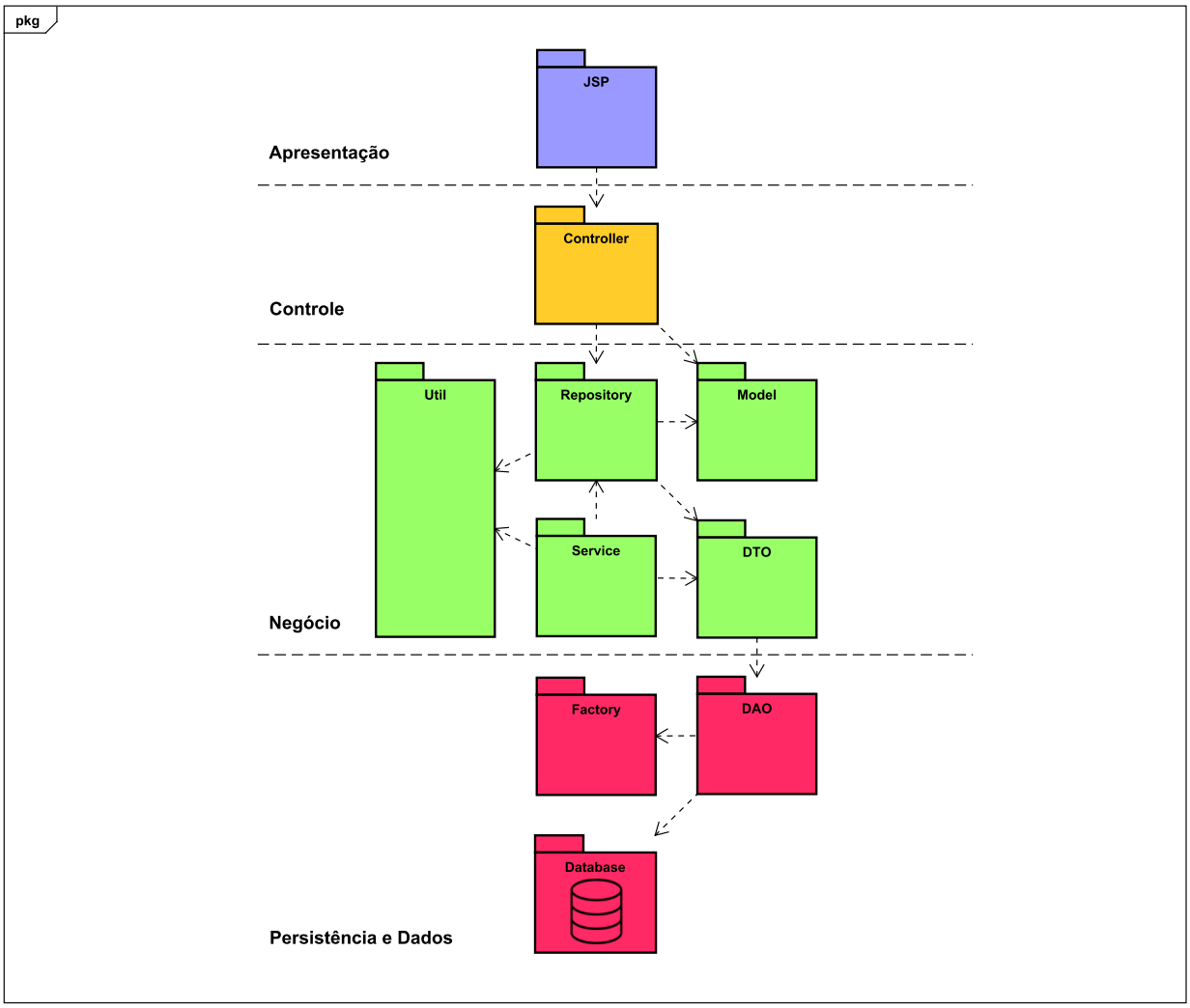
- **Confiabilidade**: Os gráficos devem apresentar de maneira clara e não tendenciosa os dados coletados.

6.4 Gestão de eventos

O sistema deverá lidar com a coleta de dados assíncrona, envio de mensagens automáticas e acesso de usuários via interface web e API.

- **Paralelismo**: Atividades de coleta de dados e envio de mensagens deverão ser realizadas de maneira paralela.

7. Camadas da Arquitetura



A solução será desenvolvida utilizando arquitetura em camadas. Os componentes do sistema serão alocados em 4 camadas: Apresentação, Controle, Negócio e Persistência de Dados. Essa arquitetura tem a característica que uma camada depende apenas da camada imediatamente abaixo.

A camada de apresentação é responsável por disponibilizar aos usuários as interfaces do sistema. A camada de controle é responsável por tratar as requisições da camada de apresentação e alimentá-la com dados.

A camada de negócio tem o objetivo de implementar as regras de negócio. Essa camada possui algumas características muito importantes pro design da solução:

- Essa camada utiliza dois objetos de tranferência de dados: **Model e DTO**. O Model é utilizado para comunicação com a camada de controle, enquanto o DTO é usado para comunicação com a camada de persistência e dados e entre objetos da própria camada de negócio.
- Os componentes da classe **Service** são executados de maneira assíncrona e tem o objetivo de realizar ações automáticas e paralelas ao fluxo principal de execução.

A camada de persistência e dados é responsável pelo armazenamento e recuperação de dados.

7.1 Estilos arquiteturais

7.1.1 Factory

Esse padrão é usado para criar instâncias *Entity Manager*, usado para persistência de dados da solução. Isso é necessário pois só deve haver um *Entity Manager Factory* para cada *Persistence Unity.* Além disso visamos a utilização desse estilo com o fato de flexibilizar e tornar reutilizável os objetos de uso.

7.1.2 Composition over Inheritance

Esse padrão é utilizado nas classes DTO. Isso é necessário pois uma base de dados relacional não suporta o conceito de herança, apenas o de composição. Esse estilo arquitetural possui benefícios para o reúso de código permitindo assim uma maior usabilidade e flexibilidade, entretanto por não utilizar de herança faz com que métodos fornecidos por componentes individuais possam necessitar serem implementados nos derivados.

7.1.3 Data Transfer Object

Um DTO é uma classe java que tem o objetivo de auxiliar a transferência de dados e desacoplar a lógica de negócio da persistência dos dados, deixando assim objetos simples de obtenção e armazenamentos de dados, deixando a lógica mais robusta e mais segura.

7.1.4 Data Access Object

Um DAO é uma classe java responsável pelo acesso à base de dados. Sua utilização é importante pois separa as regras de negócio e as regras de acesso ao banco de dados, aumentando assim a robustez do sistema e a proteção dos dados.

7.1.5 Repository

Esse padrão foi utilizado para isolar a camada lógica da camada de acesso aos dados. Apenas as classes Repository podem utilizar os Data Access Objects para acesso à base de dados. Outras classes devem utilizar classes Repository.

7.1.6 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.

.

8. Visões da Arquitetura

8.1 Casos de uso

Nome	Procurar por serviço
Ator(es)	Usuário
Descrição	O usuário busca por um serviço da internet no sistema
Referência(s)	
Gatilho	Ao tentar utilizar um serviço da internet, o usuário encontra um problema
Pré-condições	
Pós-condições	
Fluxo principal	**1.** O usuário acessa a área pública do sistema

	2. O usuário realiza a busca por um serviço **3.** O usuário escolhe um serviço **4.** O usuário tem acesso ao histórico do serviço
Fluxo alternativo	**3.** O usuário não encontra o serviço da busca **3.1** O usuário solicita que o serviço seja monitorado

Nome	Realizar cadastro
Ator(es)	Usuário
Descrição	O usuário deseja se cadastrar no sistema
Referência(s)	
Gatilho	O usuário deseja ter acesso a funcionalidades exclusivas para usuários internos
Pré-condições	
Pós-condições	O usuário pode acessar o sistema como usuário interno
Fluxo principal	**1.** O usuário acessa a área pública do sistema **2.** O usuário preenche as seguintes informações para realizar o cadastro no sistema: **-** E-mail **-** Senha **3.** O sistema valida a entrada do usuário **4.** O sistema conclui o cadastro
Fluxo alternativo	**3.** A entrada do usuário é inválida **3.1** Uma mensagem de erro é exibida **3.2** O usuário é redirecionado para a tela de cadastro

Nome	Acessar API
Ator(es)	Usuário Interno
Descrição	O usuário interno realiza uma requisição para a API e obtém dados de serviços monitorados pelo sistema
Referência(s)	
Gatilho	O usuário interno deseja realizar uma integração do sistema “Is down ?” com um sistema próprio
Pré-condições	O usuário interno deverá possuir permissão de acesso a API
Pós-condições	O acesso é registrado, dados de serviço são enviados
Fluxo principal	**1.** O usuário interno realiza uma requisição para a API **2.** O usuário interno obtém dados de serviços monitorados pelo sistema
Fluxo alternativo	**1.** O usuário interno excedeu o limite de requisições permitidas

	1.1 Uma mensagem de erro é retornada

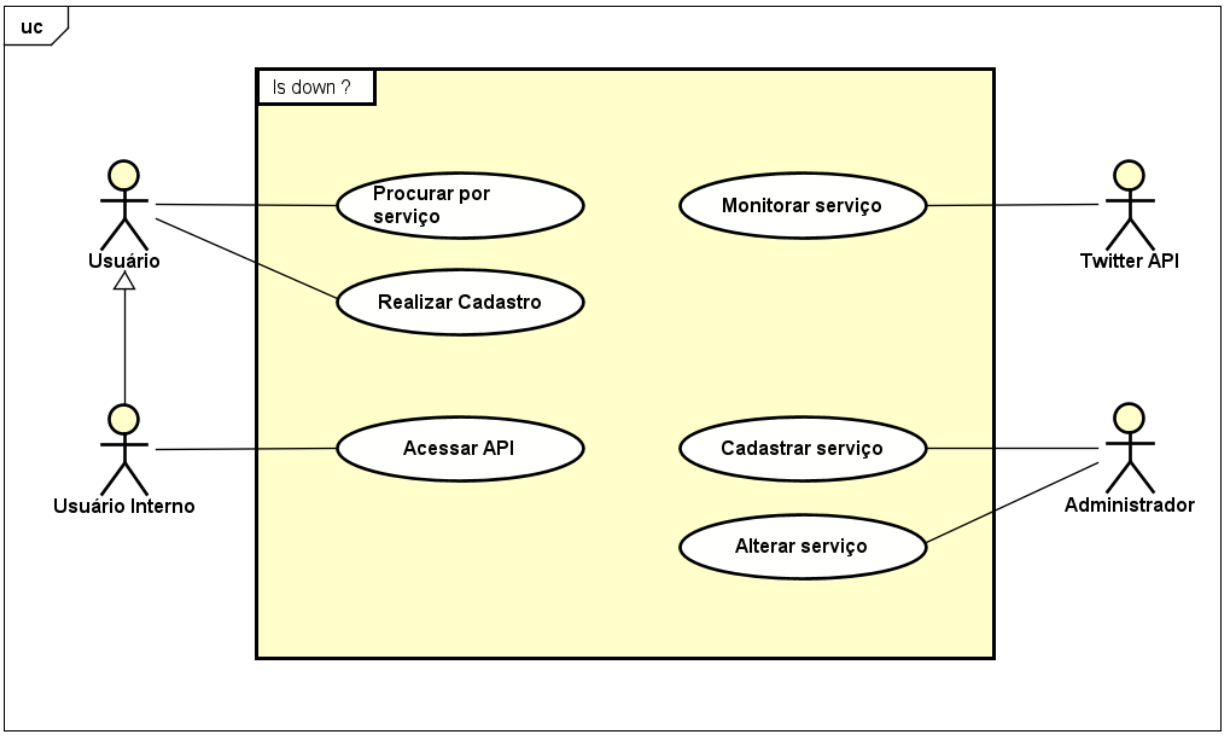
Nome	Cadastrar serviço
Ator(es)	Administrador
Descrição	O administrador adiciona um novo serviço para ser monitorado pelo sistema
Referência(s)	
Gatilho	O administrador identifica que um serviço deve ser monitorado pelo sistema
Pré-condições	O administrador deve estar logado no sistema
Pós-condições	Um novo serviço é monitorado pelo sistema
Fluxo principal	**1.** O administrador acessa a área privada do sistema por meio de login e senha **2.** O administrador preenche as seguintes informações para realizar o cadastro do novo serviço: **-** Nome do serviço **-** Logo do serviço **-** *Query* utilizada na consulta **3.** O sistema valida a entrada do administrador **4.** O sistema conclui o cadastro
Fluxo alternativo	**3.** A entrada do administrador é inválida **3.1** Uma mensagem de erro é exibida **3.2** O administrador é redirecionado para a tela de cadastro de serviço

Nome	Alterar serviço
Ator(es)	Administrador
Descrição	O administrador realiza uma modificação em um serviço já monitorado pelo sistema
Referência(s)	
Gatilho	O administrador identifica que um serviço monitorado pelo sistema precisa sofrer alguma modificação prevista
Pré-condições	O administrador deve estar logado no sistema
Pós-condições	Um serviço é modificado
Fluxo principal	**1.** O administrador acessa a área privada do sistema por meio de login e senha

	2. O administrador realiza a busca por um serviço **3.** O administrador escolhe um serviço **4.** O administrador modifica o serviço
Fluxo alternativo	**1.** O administrador erra os dados de login **1.1** Uma mensagem de erro é exibida **1.2** O administrador é redirecionado para a tela de login

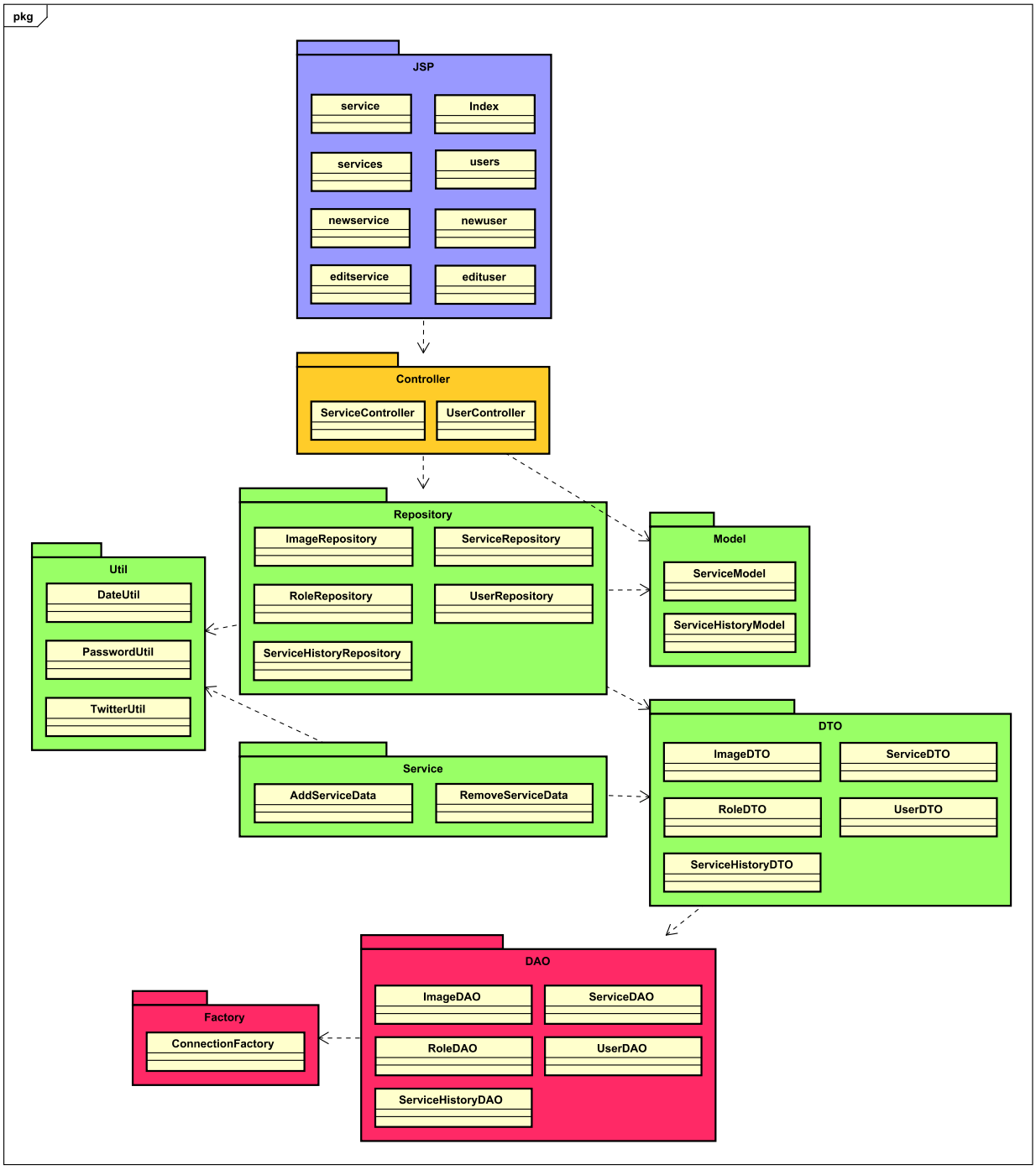
Nome	Monitorar serviço
Ator(es)	*Twitter API*
Descrição	O sistema realiza uma consulta para cada um dos serviços monitorados para obter dados recentes de reclamações dos usuários
Referência(s)	
Gatilho	Ação automática realizada a cada 15 minutos
Pré-condições	
Pós-condições	Todos os serviços são atualizados com novas informações
Fluxo principal	**1.** O sistema realiza uma consulta na *Twitter API* para cada um dos serviços cadastrados **2.** O sistema processa as respostas da *API* **3.** O sistema atualiza os serviços com os dados obtidos
Fluxo alternativo	

8.1.1 Diagrama de casos de uso



8.2 Visão Lógica

8.2.1 Estrutura de pacotes



9. Qualidade

9.1 Extensibilidade

A arquitetura em camadas permite que as camadas de negócio e persistência e dados possam ser reutilizadas no desenvolvimento de uma API, que em um primeiro momento não será desenvolvida.

9.2 Confiabilidade

Devido as metas e restrições da aplicação, é possível sempre ter dados atualizados e assertivos, por causa de pegar dados e informações de mais de uma fonte e ter momentos de manutenção recorrentes a fim de trazer todas as informações em qualquer horário.