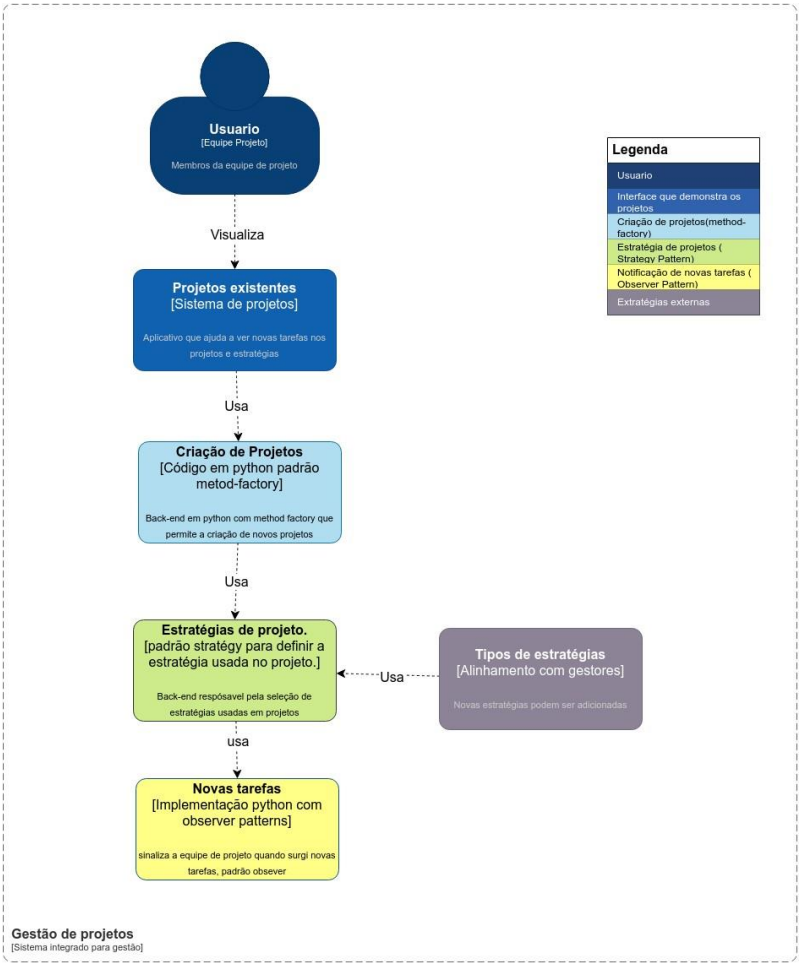
3.1 DELINEAR PROBLEMA

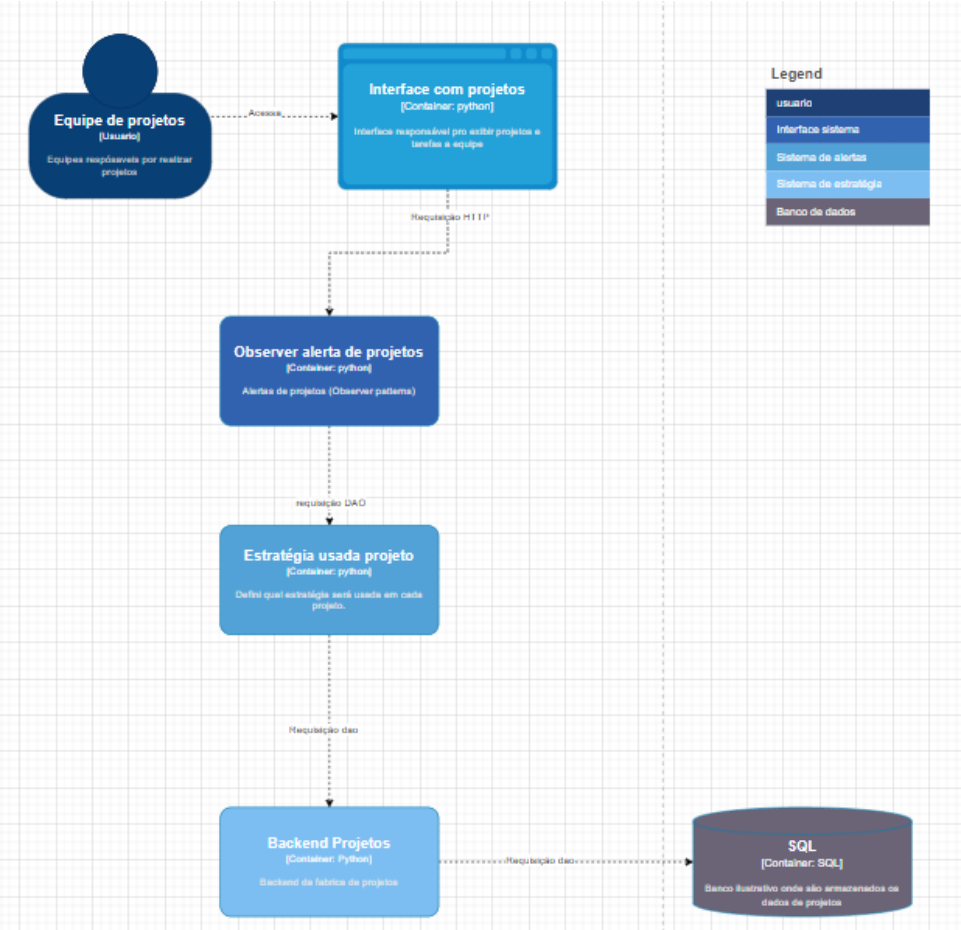
a. Contextualização sobre o sistema:

O sistema em questão é um aplicativo de gerenciamento de tarefas, desenvolvido para auxiliar equipes na organização e execução de projetos. Ele permite a criação de projetos, atribuição de tarefas a membros da equipe, definição de prazos e acompanhamento do progresso. O problema que o sistema resolve é a dificuldade de coordenação e acompanhamento de tarefas em equipes de trabalho.

C4 nivel 1



C4 NIVEL 2



3.2 IMPLEMENTAÇÃO

Observer Pattern: Para notificar automaticamente as equipes sobre as atualizações no projeto, como a conclusão de uma tarefa.

Factory Method Pattern: Para criar instâncias de tarefas específicas com base nas necessidades do projeto.

Strategy Pattern: Para permitir diferentes estratégias de acompanhamento de progresso, como gráficos de Gantt ou listas simples.

Sistem composto pelos padrões patterns method factory, estratégi factory, observ factory

O padrão Factory Method foi empregado para proporcionar uma maneira flexível de criar instâncias de projetos com diferentes estratégias de acompanhamento de progresso..

```python
class ProjetoFactory(ABC):
    @abstractmethod
    def criar_projeto(self):
        pass
```

ProjetoDetalhadoFactory e ProjetoSimplesFactory implementam o método criar_projeto, retornando instâncias específicas de Projeto com estratégias correspondentes.

```python
class EstrategiaAcompanhamento(ABC):
    1 usage (1 dynamic)
    @abstractmethod
    def acompanhar_progresso(self, update):
        pass
```

Padrão Strategy:

O padrão Strategy foi aplicado para permitir diferentes estratégias de acompanhamento de progresso nos projetos.

```python
class ProjetoDetalhadoFactory(ProjetoFactory):
    1 usage
    def criar_projeto(self):
        return Projeto(estrategia_acompanhamento=AcompanhamentoDetalhado())

1 usage
class ProjetoSimplesFactory(ProjetoFactory):
    1 usage
    def criar_projeto(self):
        return Projeto(estrategia_acompanhamento=AcompanhamentoSimples())
```

Implementações Concretas:

AcompanhamentoDetalhado e AcompanhamentoSimples são classes que implementam estratégias específicas de acompanhamento.
Acompanhamentosimples e detalhado foram duas estratégias ficticias usadas para ilustrar como funcionaria a escolha de estratégia usada pela equipe.

Padrão Observer:

- O padrão Observer foi utilizado para permitir que as equipes (observadores) recebam notificações automáticas sobre as atualizações nos projetos.

```python
class Projeto:
    def __init__(self, estrategia_acompanhamento):
        self.observers = []
        self.update = None  # Estado do projeto a ser enviado aos observadores
        self.relatorio = Relatorio()  # Instância do módulo de relatórios
        self.estrategia_acompanhamento = estrategia_acompanhamento

    4 usages
    def add_observer(self, observer):
        self.observers.append(observer)

    def remove_observer(self, observer):
        self.observers.remove(observer)
```

```
1 usage
def notify_observers(self):
    for observer in self.observers:
        observer.update(self.update)
```

```
class Equipe:
    1 usage (1 dynamic)
    def update(self, update):
        print(f"Equipe notificada sobre a atualização: {update}")
```

Implementação:

-
- Equipe é uma classe que implementa a interface Observer, com o método update que é chamado quando o projeto é atualizado.

Modificações e Adições:

Funcionalidade Adicionada: Introdução de um módulo de relatórios para análise de desempenho do projeto.

Modificação: Refatoração do sistema de notificação para incorporar o padrão Observer.

```
class Relatorio:
    1 usage
    def gerar_relatorio(self, projeto):
        # Lógica para gerar relatório
        print("Relatório gerado com sucesso.")
```

3.3 DOCUMENTAR OS DETALHES DAS IMPLEMENTAÇÕES

a. Descrição da Funcionalidade Adicionada:

O módulo de relatórios permite aos usuários visualizar o desempenho do projeto, incluindo a análise de tarefas concluídas, atrasadas e em andamento.

3.3 DOCUMENTAR OS DETALHES DAS IMPLEMENTAÇÕES

Descrição:

- A classe Relatorio contém um método gerar_relatorio que aceita uma instância de Projeto.
- O método gerar_relatorio encapsula a lógica para analisar o desempenho do projeto e gerar um relatório correspondente.
- No momento, a implementação inclui uma simples mensagem de sucesso, mas a lógica real pode ser expandida para incluir métricas específicas, gráficos, ou outros detalhes relevantes do projeto.

- A classe **`Projeto`** possui um método **`gerar_relatorio`** que delega a responsabilidade à instância de **`Relatorio`**.
- Ao chamar **`self.relatorio.gerar_relatorio(self)`**, a classe **`Projeto`** integra o módulo de relatórios e aciona a geração do relatório correspondente ao seu estado atual.
- Este método pode ser chamado em momentos específicos do ciclo de vida do projeto para fornecer insights atualizados sobre seu desempenho.

Trechos de Código Relevantes:

A seguir estão os trechos de código relevantes, enfatizando as modificações introduzidas para o módulo de relatórios.

```python
class Projeto:
    def __init__(self, estrategia_acompanhamento):
        self.observers = []
        self.update = None  # Estado do projeto a ser enviado aos observadores
        self.relatorio = Relatorio()  # Instância do módulo de relatórios
        self.estrategia_acompanhamento = estrategia_acompanhamento

    4 usages
    def add_observer(self, observer):
        self.observers.append(observer)

    def remove_observer(self, observer):
        self.observers.remove(observer)

    1 usage
    def notify_observers(self):
        for observer in self.observers:
            observer.update(self.update)

    2 usages
    def set_project_update(self, update):
        self.update = update
        self.notify_observers()

    2 usages
    def gerar_relatorio(self):
```



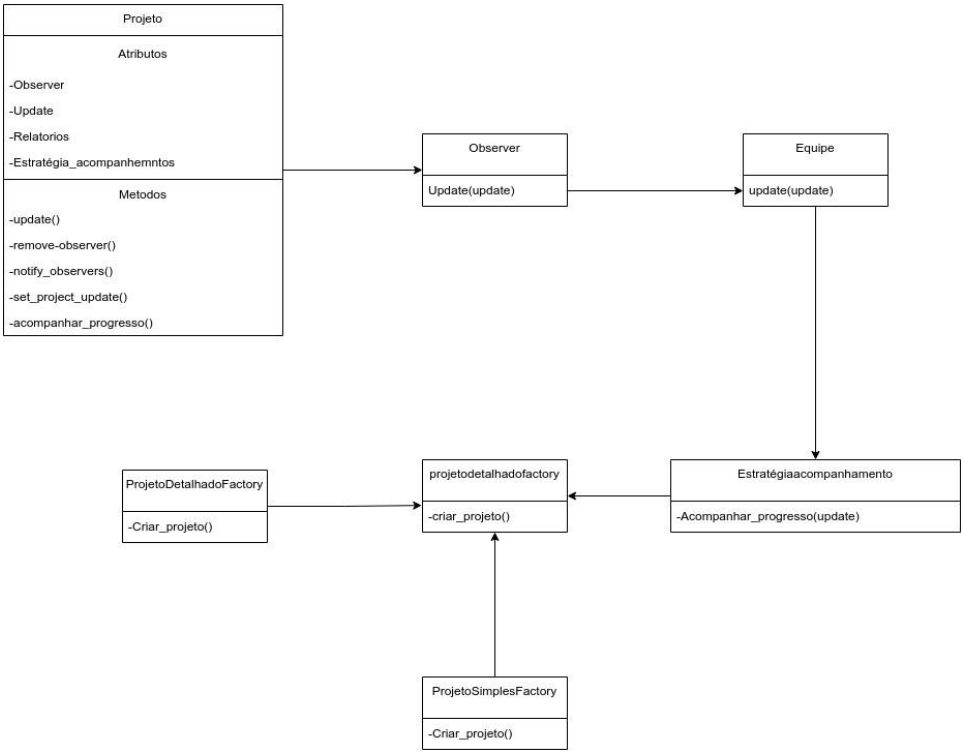
```python
    2 usages
    def gerar_relatorio(self):
        self.relatorio.gerar_relatorio(self)

    3 usages (1 dynamic)
    def acompanhar_progresso(self):
        self.estrategia_acompanhamento.acompanhar_progresso(self.update)
```

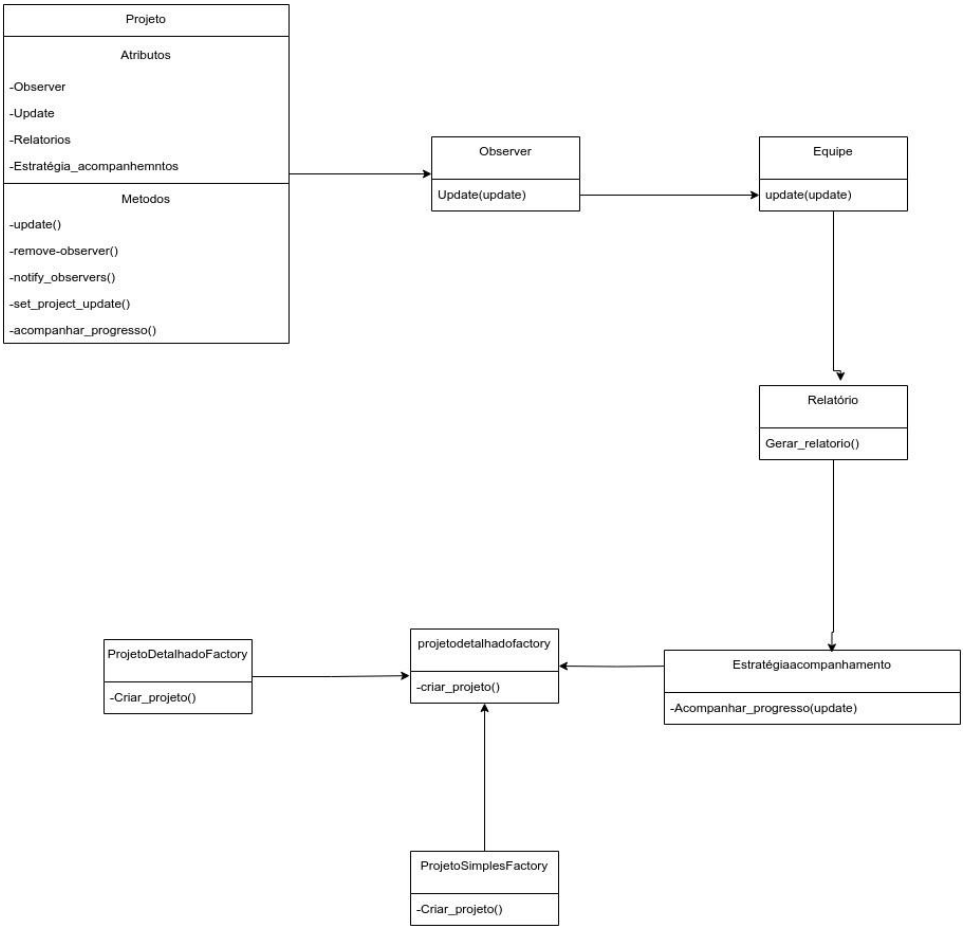
Saida terminal

```
Equipe notificada sobre a atualização: Nova tarefa no projeto detalhado.
Equipe notificada sobre a atualização: Nova tarefa no projeto detalhado.
Acompanhamento detalhado: Nova tarefa no projeto detalhado.
Detalhes específicos do acompanhamento detalhado.
Relatório gerado com sucesso.
```

b. Diagrama de Classe Pré-Modificação:



d. Diagrama de Classe Pós-Modificação:



Ferramentas:

Drawn.io