Relatório de Projeto

Paulo Jorge Mendes Proença

Engenharia Informática

nov | 2023

GUARDA
POLI
TÉCNICO

POLITÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

POKÉDEX

PROJETO EM CONTEXTO DE ESTÁGIO
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA INFORMÁTICA

Paulo Proença
Novembro / 2023

POLITÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

POKÉDEX

PROJETO EM CONTEXTO DE ESTÁGIO
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA INFORMÁTICA

Professor(a) Orientador(a): Maria Clara Silveira

Paulo Proença

Novembro / 2023

Agradecimentos

Gostava de agradecer ao Cristiano Pires e à Joana Ramos da LabsXD por todo o esforço despendido para que o estágio se realizasse, apesar do contexto desfavorável em que correu o estágio. Um enorme bem-haja à minha orientadora, Professora Maria Clara Silveira pela sua disponibilidade e confiança nas minhas capacidades.

Não podia deixar de agradecer a todos os professores que ao longo desta caminhada me passaram não só o seu conhecimento, mas também valores muito importantes.

Não menos importante um agradecimento para a minha família por todo o apoio concedido desde o primeiro dia e também aos meus colegas pela ajuda, coragem e força que me deram ao longo de toda a licenciatura.

O agradecimento mais especial é para a minha namorada por toda a paciência, dedicação e carinho me deu e por nunca me deixar desistir do meu sonho.

Ficha de Identificação

Aluno

Nome: Paulo Jorge Mendes Proença

Nº de Aluno: 1704890

Licenciatura: Engenharia Informática

Estabelecimento de Ensino

Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda (ESTG-IPG)

Morada: Av. Dr. Francisco Sá Carneiro 50, 6300-559 Guarda

Telefone: 271 220 120

Entidade Acolhedora do Estágio

Nome: LabsXD

Morada: Centro de Negócios e Serviços, Praça Amália Rodrigues S/N, 6300-350 Fundão

Telefone: 935578357

Duração do Estágio: 13/06/2023 – 13/08/2023

Supervisor de Estágio

Nome: Cristiano Pires

Função: SFDC & SI Developer

Docente Orientador de Estágio

Nome: Maria Clara Dos Santos Pinto Silveira

Grau Académico: Doutoramento

Resumo

O presente relatório pretende descrever o projeto em contexto de estágio realizado na empresa LabsXD, no âmbito da unidade curricular Projeto de Informática.

O objetivo do projeto consiste no desenvolvimento de uma aplicação utilizando a plataforma Salesforce, para a visualização de Pokémons recorrendo a uma *API* externa. A aplicação deve:

- Efetuar a inserção de dados usando os dados de uma *API* externa;
- Apresentar os Pokémons e suas informações na forma de *Card;*
- Permitir efetuar pesquisas por Nome, Geração e Tipo, em simultâneo;
- Ter em conta os limites de *Callouts* por transação em Apex.

O desenvolvimento da aplicação foi efetuado recorrendo a uma metodologia ágil, assim como ás seguintes tecnologias:

- Apex;
- JavaScript;
- HTML/CSS;
- Plataforma Salesforce.

Palavras-Chave: Salesforce, *CRM*, Apex, *API*

Abstract

This report aims to describe the internship project carried out at the LabsXD company, as part of the Informatics Project course.

The aim of the project is to develop an application using the Salesforce platform for visualising Pokémon using an external *API*. The application should:

- Insert data using data from an external *API*;
- Display Pokémon and their information in *Card* form;
- Allow simultaneous searches by Name, Generation and Type;
- Consider the limits of Callouts per transaction in Apex.

The application was developed using agile methodology and the following technologies:

- Apex;
- JavaScript;
- HTML/CSS;
- Salesforce platform.

Keywords: Salesforce, *CRM*, Apex, *API*

Índice Geral

Índice de Figuras

Índice de Tabelas

Lista de Siglas e Acrónimos

API	Application Programming Interface
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
DML	Data Manipulation Language
GraphQL	Graph Query Language
gRPC	Google Remote Procedure Call
HP	Hit Points
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JS	JavaScript
JSON	JavaScript Object Notation
LWC	Lightning Web Components
PaaS	Platform as a Service
QA	Quality Assurance
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
SLDS	Salesforce Lightning Design System
SOAP	Simple Object Access Protocol
SOQL	Salesforce Object Query Language
UI	User Interface
URL	Uniform Resource Locator
WIP	Work In Progress
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

1. Introdução

O presente relatório descreve o project desenvolvido em contexto de estágio na LabsXD, pelo aluno Paulo Proença, no âmbito da unidade curricular de Projeto de Informática, referente ao 3º ano da licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

O projeto teve uma duração de 2 meses, tendo sido realizado na sua totalidade de forma remota e dividido em duas fases:

- *Learning Path* utilizando a plataforma Trailhead;
- Desenvolvimento da aplicação.

1.1. Enquadramento e Motivação

Num mercado cada vez mais competitivo, o relacionamento com os clientes torna-se muito importante para o sucesso de uma empresa. É aqui que o *Customer Relationship Management (CRM)* adquire um papel importante.

A utilização de um *CRM* promove um aumento significativo na eficiência com que os dados dos clientes são tratados e colocados ao dispor das suas equipas, promovendo assim um melhor relacionamento com o cliente e por consequência aumento de receitas.

Considerado o líder em sistemas de *CRM*, a Salesforce oferece tanto soluções *Software as a Service* (SaaS*)* como *Platform as a Service* (PaaS*)* (Cofes, 2023). É esta última a utilizada para o desenvolvimento da aplicação a que este relatório se refere.

O projeto além de demonstrar o potencial da solução *PaaS* da Salesforce, pretende também mostrar a importância das *Application Programming Interface (API)* e a sua integração em aplicações.

1.2. Caraterização sumária da instituição de acolhimento

A LabsXD Portugal, fundada em 2021 e situada na cidade do Fundão, faz parte de multinacional sediada na Argentina especializada em soluções Salesforce e Industries Cloud e considerada uma referência na América Latina no que diz respeito a soluções ágeis voltadas para a transformação digital dos seus clientes.

A LabsXD conta com uma vasta experiência em áreas como vendas, integração, análise e comércio, e uma presença significativa em vários sectores, incluindo comunicações, saúde, governo ou energia.

No seu portfolio conta com vários projetos de larga escala para clientes de alto nível.

De forma a captar novos talentos para as suas equipas, a LabsXD conta com o XDGen, um curso direcionado a quem pretende integrar o mundo da Salesforce, socorrendo-se da plataforma Trailhead para o efeito.

1.3. Descrição do problema

A aplicação desenvolvida faz parte do processo de avaliação do curso *XDGen – Developers* no qual é pretendido que seja desenvolvido um Pokédex, uma aplicação que serve como uma enciclopédia eletrónica dedicada às personagens do popular jogo de vídeo Pokémons.

Os dados são obtidos recorrendo a uma *API* externa chamada PokeAPI, que contem as informações atualizadas e detalhadas sobre cada Pokémon.

Esses dados são depois transformados e inseridos num *Salesforce Custom Object* criado para efeito. É usada a *framework Lightning Web Components* para criar uma *interface* mais intuitiva e amigável, de forma a facilitar a interação dos utilizadores com a aplicação.

Além de permitir a visualização de dados, um outro recurso importante da aplicação é a capacidade de pesquisa, permitindo aos utilizadores realizar pesquisas por nome, geração ou tipo(s), tornando-se uma ferramenta valiosa para os amantes e entusiastas de Pokémons.

Em resumo, o projeto pretende demonstrar a capacidade de integração de uma *API* externa com a plataforma Salesforce e a capacidade de criar *interfaces* intuitivos e fáceis de usar recorrendo à *framework LWC*. É um excelente exemplo como criar aplicações úteis e interessantes, utilizando tecnologias modernas.

1.4. Objetivos

O objetivo do projeto foi desenvolver uma aplicação que demonstre os conhecimentos adquiridos durante o curso XDGen e que cumpra uma série de requisitos fornecidos pela empresa. Alguns desses requisitos são:

- Integração da *API* PokeAPI que permita a inserção de dados num *Custom Object* a ser criado;
- Desenvolver uma *interface* mais agradável ao utilizador usando a *framework LWC*;
- Desenvolver filtros de pesquisa.

Para cumprir os requisitos funcionais da aplicação, foram realizadas as seguintes etapas:

- Estudo das tecnologias necessárias ao desenvolvimento do projeto;
- Análise do estado da arte;
- Preparação do ambiente de desenvolvimento;
- Desenvolvimento da aplicação;
- Realização de testes;
- Elaboração do relatório do projeto.

1.5. Estrutura do documento

O presente relatório está estruturado em oito capítulos que pretendem descrever as diferentes etapas de ciclo de desenvolvimento de software.

O capítulo inicial apresenta uma breve introdução ao método utilizado durante o estágio bem como uma descrição sobre a empresa e quais foram os objetivos propostos para a realização do projeto. No capítulo dois é apresentado o estado da arte. A metodologia utilizada durante o desenvolvimento do projeto é abordada no capítulo três, seguindo-se o capítulo quatro onde é apresentada a análise de requisitos. O capítulo cinco apresenta as tecnologias utilizadas no desenvolvimento do projeto. Os capítulos seis e sete apresentam respetivamente a implementação do projeto e os testes realizados.

Por último, no oitavo capítulo são apresentadas as conclusões relativas a este projeto.

2. Estado da Arte

O estado da arte é um ponto importante pois permite situar o projeto no contexto de pesquisas realizadas na área, identificar lacunas existentes, as oportunidades de inovação bem como as melhores práticas para a sua execução.

Neste projeto é proposto o desenvolvimento de uma aplicação em Salesforce, realizando a integração da *API* pública PokeAPI para a inserção de dados num objeto Salesforce e desenvolvendo um componente *LWC* que permita a visualização desses dados e realizar pesquisas. O objetivo é demonstrar as capacidades da plataforma Salesforce e do seu *framework LWC* no desenvolvimento de aplicações web recorrendo a fontes publicas e gratuitas para a obtenção de dados. Para isso, é necessário realizar um estudo sobre o estado da arte das tecnologias utilizadas no projeto, ou seja, o estado atual do conhecimento técnico e científico sobre *APIs*, a plataforma Salesforce, o *framework LWC* e a biblioteca *SLDS (Salesforce Lightning Design System)*.

2.1. APIs e a sua Integração em Salesforce

Dado que o mundo digital se encontra cada vez mais exigente é necessário encontrar uma solução que permita corresponder a esse nível de exigência de forma mais célere e eficiente. É aqui que as *APIs* ganham destaque, uma vez que a sua integração permite uma maior agilidade e produtividade, pelo que a sua integração em plataformas *SaaS* e *PaaS*, neste caso a plataforma *Salesforce*, adquirem uma enorme importância.

Uma *API* pode ser vista como um conjunto de normas que possibilita a comunicação entre plataformas utilizando uma série de padrões e protocolos.
A arquitetura de uma *API* é uma arquitetura cliente-servidor, onde o cliente é a aplicação que solicita o *request* e o servidor a aplicação que retorna a *response* com um resultado esperado (AmazonWebServices, 2023).

A *Salesforce* permite a utilização de quatro tipos de *APIs* (Salesforce, 2023):

- *REST - Representational State Transfer;*
- *GraphQL - Graph Query Language;*
- *SOAP - Simple Object Access Protocol;*
- *gRPC - Google Remote Procedure Call.*

2.1.1. REST

Quando se trata do desenvolvimento de *APIs* públicas e simples, as *REST APIs* são a escolha indicada, até pela sua facilidade de aprendizagem e a grande comunidade existente. Além do formato *XML* podem ser utilizados também os formatos *HyperText Markup Language* (*HTML), Plain Text* e *JavaScript Object Notation (JSON)*, sendo este último o mais utilizado pela sua simplicidade. Foi este o protocolo utilizado no projeto (AltexSoft, 2023).

2.1.2. SOAP

As *SOAP APIs*, lançadas em 1999, contam com uma difícil curva de aprendizagem, bem como uma comunidade pequena, e utilizam apenas o formato *Extensible Markup Language (XML)*. São a escolha indicada quando o objetivo é maior segurança como por exemplo (AltexSoft, 2023):

- métodos de pagamento;
- serviços de comunicação e financeiros;
- Soluções *CRM.*

2.1.3. GraphQL

Com o crescimento no desenvolvimento de aplicações móveis e o aumento na complexidade dos sistemas foi necessário dar um passo em frente e assim nasceu o *GraphQL*. Enquanto que nas *REST APIs* poderá ser necessário efetuar vários *calls* para obter todos os dados pretendidos, utilizado o *GraphQL* será efetuado apenas um uma vez que no *request* é passado um esquema, que é nada mais que um conjunto de *queries* e os respetivos tipos que serão devolvidos. Infelizmente esta vantagem pode ser ao mesmo tempo uma desvantagem pois ter demasiado campos aninhados pode levar a uma sobrecarga do sistema, pelo que por vezes a escolha mais acertada passa por utilizar *REST.*

De referir que o formato utilizado é o *JSON* (AltexSoft, 2023).

2.1.4. gRPC

O *gRPC* é nada mais que um *upgrade* do protocolo *RPC*. Desenvolvido pela *Google* em 2015, o *gRPC* implementa o protocolo *RPC* tradicional, mas utilizando *buffers* de protocolo e *HTTP 2,* e que remove a necessidade de mapear os conceitos de *RPC* para o protocolo *HTTP*. Esta otimizações torna o *gRPC* mais rápido e eficiente. Tal como o *RPC* tradicional utiliza a sintaxe *GET* e *POST* para efetuar a comunicação, tornando-o simples de aprender e utilizar, sendo a sua utilização adequada na criação de chats e microsserviços internos. Utiliza os formatos *XML* e *JSON* bem como binário (AltexSoft, 2023) (AWS, 2023).

2.2. Vantagens e desvantagens das APIs

Aqui são apresentadas algumas das vantagens e desvantagens de uma *API* (Idwall, 2018).

2.2.1. Vantagens

- Facilitar a integração de sistemas que sejam incompatíveis;
- Permitir a automatização de operações, aumentando assim a produtividade e agilizando processos;
- Criação de soluções inovadoras recorrendo a fontes externas e gratuitas para a obtenção de dados.

2.2.2. Desvantagens

- Requer custos e tempo elevados no seu desenvolvimento;
- Necessidade de manutenção constante;
- Levar a quedas de sistemas ou perda de dados caso não sejam bem projetadas ou geridas.

3. Metodologia

Os métodos ágeis são métodos incrementais, onde geralmente os itens incrementados são pequenos e normalmente as novas versões são disponibilizadas em duas ou três semanas (Somerville, 2011).

Este capítulo aborda a metodologia utilizada ao longo do processo de desenvolvimento do projeto, o que neste caso foi a metodologia ágil *Kanban* com recurso à ferramenta Trello.

3.1. Metodologia Kanban

A metodologia Kanban é um sistema visual de gestão de trabalho, que se baseia no acompanhamento visual das etapas de um projeto, utilizando cartões para representar tarefas e colunas para definir o seu estado, sendo os mais comuns "*Backlog*", "*To Do", "Doing"* e "*Done*". O objetivo é otimizar o fluxo de atividades e definir prioridades de forma a aumentar a produtividade (HINC, 2023).

Tem a sua origem nos processos de produção *just-in-time (JIT)* idealizados pela Toyota, nos quais eram usados cartões para identificar necessidades materiais na linha de produção (IEBS, 2023).

É necessário definir limites de trabalho em andamento (WIP) em cada coluna, ou seja, definir o número máximo de quantos cartões podem existir ao mesmo tempo em cada coluna de forma a ajudar a controlar o fluxo de trabalho e a evitar o acumular de trabalho.

3.2. Aplicação da metodologia Kanban no desenvolvimento do projeto

Embora a metodologia Kanban não seja rígida na atribuição de papeis, pode-se considerar que a equipa formada para o desenvolvimento do projeto foi composta pelo aluno no papel de d*eveloper* e *tester*, e pelo supervisor de estágio no papel de gestor de projeto e cliente. Foi criado um quadro na plataforma Trello composto por 6 colunas que representavam os vários estados do desenvolvimento e onde foram criados cartões numerados e ordenados, como se pode ver na Figura 1.

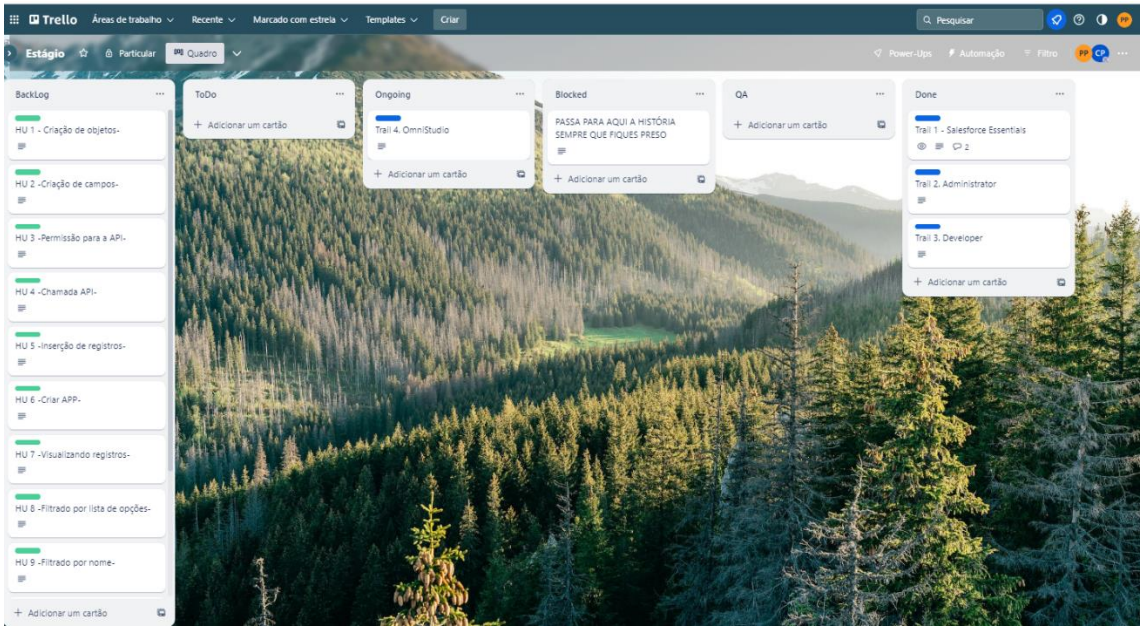


Figura 1 - Fluxo de trabalho - Trello

Cada coluna representa uma fase especifica no processo de desenvolvimento, e onde se encontram as tarefas a presentes no projeto. Cada tarefa representa uma *user story*, como se pode verificar na Figura 2, que neste caso consiste na inserção de dados no objeto Salesforce.

Apresenta-se de seguida uma breve descrição sobre cada uma das colunas usadas no projeto:

- *Backlog*: aqui situa-se a lista de tarefas a realizar, normalmente identificadas por prioridade;
- *ToDo*: quando as tarefas presentes no *Backlog* estão prontas para serem executadas, elas são colocadas na coluna *ToDo*;
- *Ongoing*: nesta coluna encontram-se as tarefas que estão as ser realizadas, ou seja, quando o *developer* inicia uma tarefa presente na coluna *ToDo*, essa tarefa é deslocada para a coluna *Ongoing*;
- *Blocked*: sempre que ao longo do projeto existisse alguma tarefa na qual o *developer* estivesse bloqueado, ela seria colocada na coluna *Blocked* de forma a que o supervisor tivesse conhecimento disso e prestasse o devido auxílio;
- *QA(Quality Assurance)*: nesta coluna serão colocadas a tarefas terminadas pelo *developer*, mas que ainda necessitam ser verificas e validadas pelo supervisor.
- *Done*: esta coluna contem as tarefas realizadas e que já foram verificadas e validadas pelo supervisor. É da responsabilidade do mesmo mover as tarefas da coluna *QA* para a coluna *Done*.

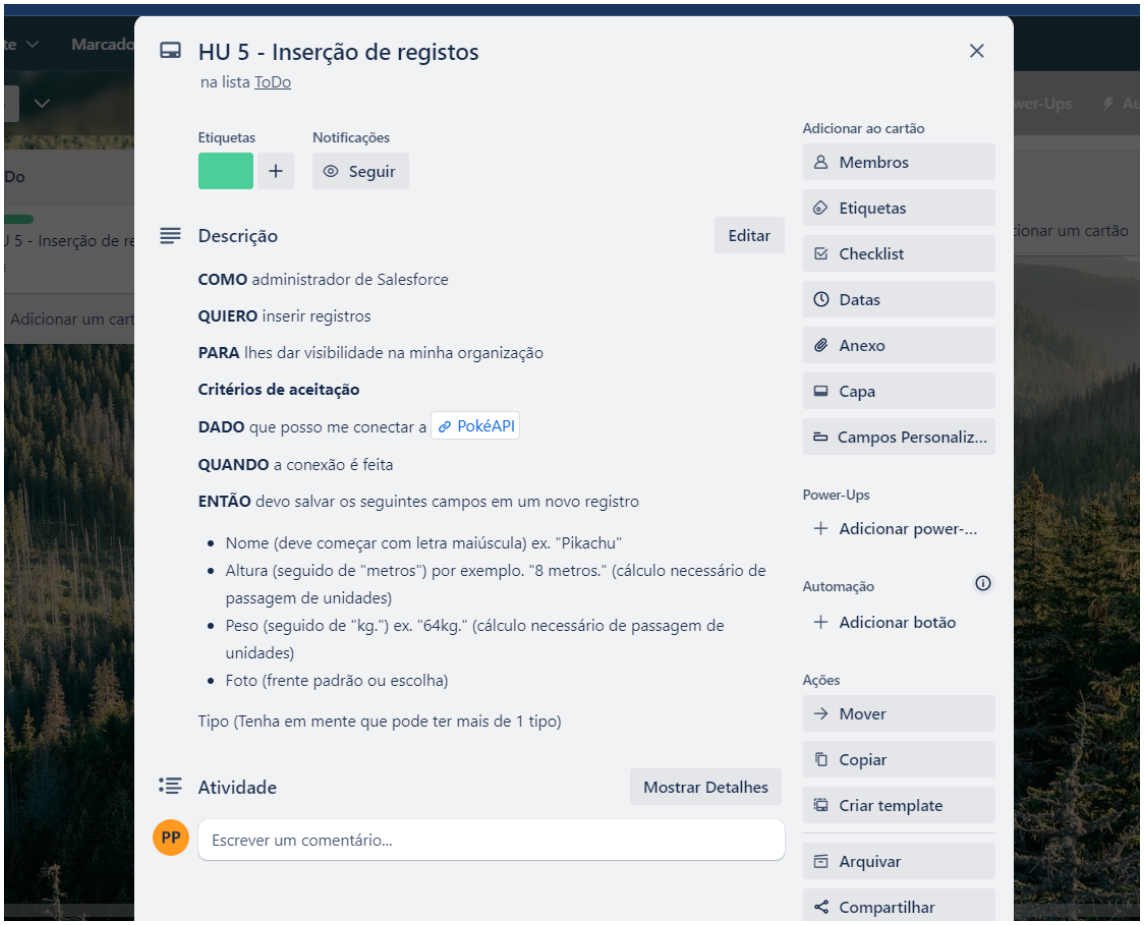


Figura 2 - Cartão Kanban

Foi da responsabilidade do *developer* a gestão do fluxo de trabalho até ao estado "*QA*", tendo apenas sido definido o prazo final para a conclusão total do projeto, não havendo lugar a entregas intermédias nem reuniões para discutir a evolução do projeto, sendo estas algo bastante pontual e apenas quando algum dos cartões se encontrava no estado "*Blocked*" para que o problema que impedia o desenvolvimento do projeto pudesse ser resolvido. Uma vez movido um cartão para "*QA*", coube ao gestor de projeto efetuar a revisão do trabalho realizado e mover o mesmo para "*Done*" caso este cumprisse os critérios de aceitação, ou devolvendo o cartão para "*ToDo*" com um comentário associado, referindo o erro ou sugestão de melhoramento.

3.3. Trailhead

De forma a garantir um maior conhecimento sobre as tecnologias, linguagens e ferramentas a utilizar no desenvolvimento do projeto foi realizado um curso denominado XDGen, utilizando a plataforma de aprendizagem *online* Trailhead. Esta plataforma contém inúmeros módulos sobre tudo o que é necessário saber acerca de Salesforce. Conta também com uma comunidade bastante prestativa e pronta a colaborar com todos os seus utilizadores. Foram estabelecidos 4 cartões no quadro da plataforma Trello de forma a que a evolução do aluno fosse sendo acompanhada.

O curso foi composto por quatro *trailmixes*:

- XD Gen Essentials: é um *trailmix* onde é feita uma introdução à plataforma Salesforce e abordados temas como *data modeling*;
- XD Gen Administrator: aqui são tratados temas ligados à administração da plataforma, segurança, automação de processos e *data quality*, bem como uma introdução à linguagem Apex e SOQL;
- XD Gen Developer: *trailmix* que aborda *API*s e a sua integração, desenvolvimento de componentes LWC, implementação de classes assíncronas, VisualForce Pages e *debugging*;
- XD Gen OmniStudio: aqui é feita a introdução ao OmniStudio, uma plataforma que oferece um conjunto de serviços e componentes, que são utilizados no desenvolvimento de aplicativos do *Industry Cloud*. São apresentados os *Data Raptors*, *Integration Procedures, Omnicards* e *Omniscripts.*

4. Análise de requisitos

A análise de requisitos é uma etapa crucial no desenvolvimento de software, pois identifica e especifica as funcionalidades propostas e as expectativas dos *stackholders*. É frequentemente considerada uma das primeiras ações a realizar de forma a garantir o desenvolvimento de *software* de qualidade.

Para o desenvolvimento do projeto foram fornecidos cartões com *user stories* baseados no enunciado presente em anexo (Anexo A 1).

4.1. Atores

Consideram-se atores utilizadores ou qualquer meio externo que interaja com o sistema, sendo que neste projeto existem dois atores:

- Administrador;
- PokeAPI

Tabela 1 - Tabela de Ator

Ator	Descrição
Administrador Salesforce	O administrador de Salesforce é o utilizador responsável por interagir com a aplicação PokeAPP desenvolvida para visualizar e gerir os registros dos Pokémon. Ele possui permissões administrativas no Salesforce e está encarregue de criar, visualizar, filtrar e gerir os dados dos Pokémon na organização. O administrador utiliza a aplicação para manter registros detalhados dos Pokémon e criar uma *interface* intuitiva para os utilizadores finais.
PokeAPI	É a API publica encarregue de fornecer os dados para a PokeAPP.

4.2. Diagrama de casos de uso

Um diagrama de casos de uso é uma ferramenta de representação que permite representar graficamente os requisitos funcionais de um sistema, apresentando as interações entre os atores e os casos de uso.

Após uma análise foi decidido entre o aluno e o supervisor que seria interessante visualizar também os dados referentes aos *stats* (*HP, Attack, Speed, …)* dos pokemons

quando fosse acedida a página do registo, tendo sido alterado o diagrama de casos de uso inicial (Anexo A 2) pelo apresentado na Figura 3:

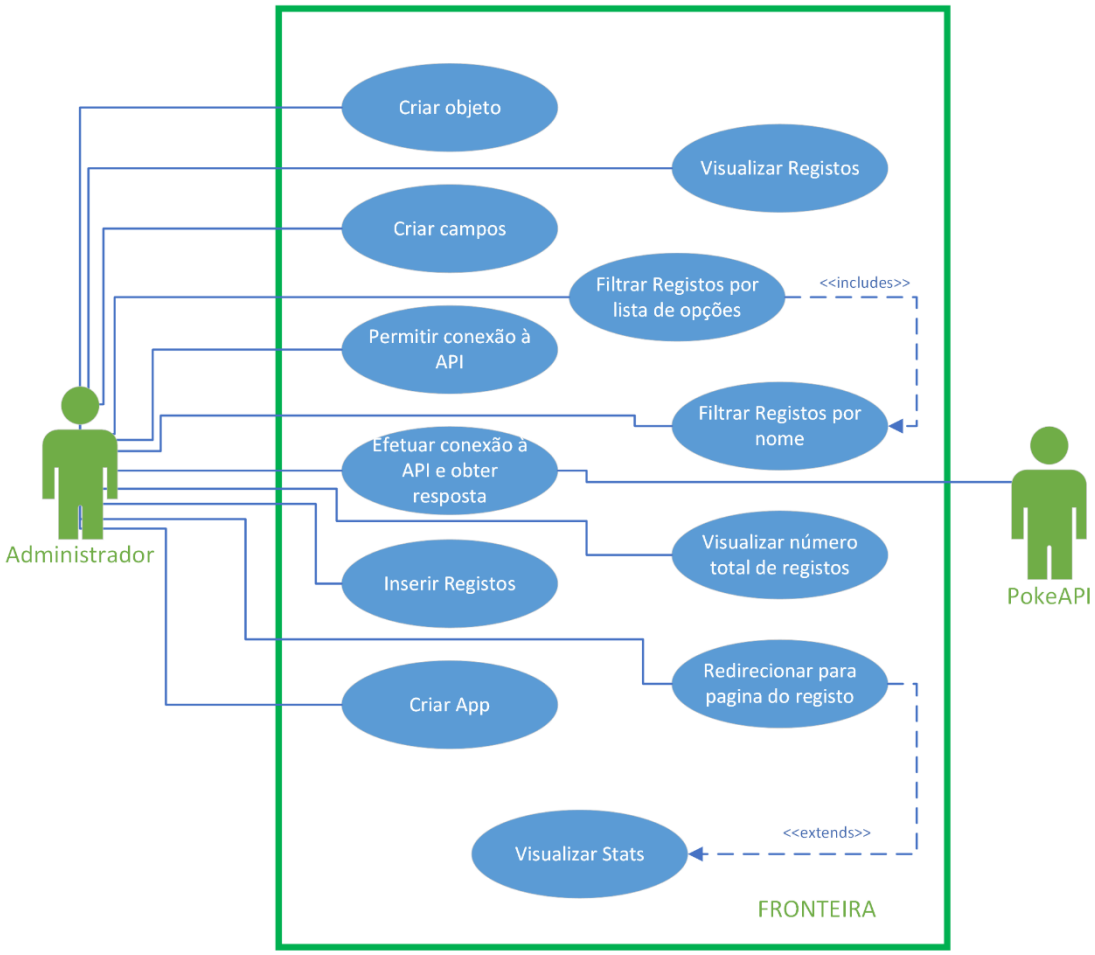


Figura 3 - Diagrama de casos de uso

4.3. Descrição de casos de uso

Neste subcapítulo encontram-se descritos alguns dos casos de uso referenciados no diagrama apresentado no subcapítulo anterior, encontrando-se os restantes em anexo (Anexo A 3).

Estão descritos em forma de tabela com a seguinte estrutura:

- Nome: Nome do caso de uso;
- Descrição: Breve descrição sobre a funcionalidade do caso de uso;
- Pré-condição: Condição inicial necessária para realização do caso de uso;
- Caminho Principal: Descrição das etapas a serem realizadas para que o caso de uso seja realizado com sucesso;

- Caminho Alternativos: Descrição das etapas quando ocorre uma falha no caminho principal;
- Adornos ou Suplementos: Requisitos não funcionais e testes a serem realizados;
- Pós-condições: Condição obtida após a execução do caso de uso.

4.3.1. Criar Campos

A tabela 2 descreve os passos necessários para que o Administrador possa Cria Campos num determinado objeto Salesforce.

Tabela 2 - Descrição de caso de uso Criar Campos

Nome	Criar Campos.
Descrição	Este caso de usos tem como objetivo descrever o processo de Criar Campos num objeto Salesforce.
Pré-Condição	O administrador está autenticado no Salesforce, tem permissão para editar o objeto.
Caminho Principal	1. O administrador seleciona "*Object Manager*". 2. O sistema devolve uma lista com todos os objetos presentes no ambiente Salesforce visíveis ao administrador; 3. O administrador localiza e seleciona o objeto Pokémon; 4. O sistema devolve a página de *Details* do objeto; 5. O Administrador seleciona no menu da esquerda "Fields & Relationships" e seleciona *"New";* 6. O sistema retorna uma lista de tipos de dados; 7. O administrador seleciona o tipo de dados desejado e efetua a configuração do campo, introduzindo o seu nome e outras características que sejam relevantes e clica em *"Save"*.
Caminho Alternativo	3. a) O Administrador não localiza o objeto pois ele não existe ou não se encontra visível ao administrador, pelo que o sistema não devolve nenhum resultado; 7. a) O administrador clica em *"Cancel"*.
Adornos ou Suplementos	-
Pós-Condições	Campo adicionado ao Objeto.

4.3.2. Inserir Registos

A seguinte tabela descreve os passos necessários para que o Administrador possa inserir registos no objeto Salesforce.

Tabela 3 - Descrição de caso de uso Inserir Registos

Nome	Inserir Registos.
Descrição	Este caso de usos tem como objetivo descrever o processo de Inserir Registos num objeto Salesforce.
Pré-Condição	Garantida a permissão para que a plataforma se conecte à PokeAPI.
Caminho Principal	1. O administrador executa a *Apex Batch Class* que contém o código necessário para efetuar as *calls* e respetiva inserção; 2. O sistema conecta à *API* e efetua os pedidos; 3. A *API* retorna as respostas aos pedidos efetuados; 4. O sistema efetua a desserialização das respostas obtidas e efetua a inserção dos dados no objeto.
Caminho Alternativo	2. a) O sistema não se conecta à *API;* 3. a) A *API* é incapaz de retornar os pedidos.
Adornos ou Suplementos	Testar a utilização de *endpoints* incorretos; Testar a execução da *Apex Batch Class* usando um valor demasiado alto para o tamanho do *batch.*
Pós-Condições	Dados inseridos no Objeto.

5. Tecnologias

Neste capítulo serão apresentadas as tecnologias utilizadas no desenvolvimento do projeto.

5.1. Plataforma Salesforce

A Salesforce é uma plataforma que permite o desenvolvimento de aplicações de forma rápida e simples, recorrendo a ferramentas *low code* através de uma abordagem *point and click*. Através desta mesma abordagem podem ser criados processos de automação e definir regras de negócio que tornem a solução *CRM* da Salesforce mais eficiente. Além de permitir a integração de *APIs* externas como fonte de dados ou de conectores pré-configurados disponíveis na AppExchange, a plataforma garante, através de uma arquitetura baseada em *Cloud, o Hyperforce,* segurança, escalabilidade e desempenho das aplicações (Salesforce P. , 2023) .Mas nem só de *low code* vive a plataforma Salesforce uma vez que podemos recorrer ao desenvolvimento de código de forma a suprimir alguma necessidade especifica assim como desenvolver uma *interface* personalizada e mais *user-friendly*, sendo para tal fornecidos aos *Developers* vários recursos tais como o *Salesforce DX*, *Lightning Experience ou LWC.*

Sendo uma solução *PaaS*, é executada num ambiente *multitenant*, isto é, um ambiente de partilha de recursos onde todos os clientes detêm a mesma capacidade de computação e armazenamento, tal como podemos ver na Figura 4.

Figura 4 - Ambiente Multitenant

5.2. Apex

Apex é uma linguagem de programação exclusiva do Salesforce, fortemente tipada e orientada a objetos com uma sintaxe similar à linguagem Java. Oferece suporte integrado para Data Manipulation Language (*DML*), *Salesforce Object Query Language (SOQL)*, *looping* para processamento de dados em massa, criação de testes unitários, entre outros. (Apex, 2023)

5.3. LWC

LWC é um *framework* de *UI* que permite a criação de páginas customizadas e funções na plataforma Salesforce. Faz uso da linguagem *JS*, *HTML* e *CSS*, e contém uma biblioteca de design denominada *SLDS*.

Cada componente é um bloco que pode ser reutilizado pelos *developers* para os mais variados usos.

5.3.1. JavaScript

O *JavaScript*, tipicamente designado por *JS*, é uma linguagem de alto nível, dinâmica, baseada em protótipo e multiparadigma. Desde o seu lançamento em 1996 que foram adicionadas novas funções e aplicações. Como se pode ver pela Figura 5 existem vários *frameworks* e bibliotecas que utilizam o *JS* para criar tanto aplicações web como aplicações móveis.

Figura 5 - Frameworks e Bibliotecas JS

5.3.2. HTML

HTML é uma linguagem de marcação usada para criar páginas web. Ela define a estrutura e o conteúdo de um site, permitindo que os navegadores exibam texto, imagens e outros elementos, através da interpretação de *tags* (HTML, s.d.).

5.3.3. CSS

Cascading Style Sheets, vulgarmente conhecido por *CSS*, é uma linguagem estilo usada para descrever a apresentação de um documento escrito em *HTML* ou *XML*. O *CSS* descreve como os elementos devem ser renderizados no ecrã (CSS, s.d.).

5.4. PokeAPI

PokeAPI é uma *API* pública baseada em *RESTful* que contém dados sobre todos os Pokémons conhecidos. Ela é capaz de retornar dados nos formatos *JSON*, *YAML Ain't Markup Language (YAML)* e *XML*. Permite também a utilização de *GraphQL* (Hallett, s.d.).

5.5. Visual Studio Code

Uma vez que os componentes *LWC* não podem ser executados na *developer console* presente na plataforma Salesforce, foi necessário recorrer ao *IDE* tendo sido o Visual Studio Code o escolhido para o efeito.

Visual Studio Code é um editor de código-fonte leve, mas poderoso que se encontra disponível para Windows, macOS e Linux. Ele vem com suporte embutido para JavaScript, TypeScript e Node.js e tem um rico ecossistema de extensões para outras linguagens e tempos de execução (como C++, C#, Java, Python, PHP, Go, .NET). (Microsoft, 2023).

6. Implementação

Como foi referido anteriormente, este projeto tem como objetivo efetuar a integração de uma *API* externa com a plataforma Salesforce para a obtenção de dados, inserir esses mesmo dados e criar uma *interface* mais amigável para a visualização dos dados obtidos pelo que este desenvolvimento da aplicação foi separada essencialmente em quatro partes:

- Criação do objeto Salesforce;
- Definir permissão para a integração da *API* externa;
- Back-end onde foi efetuada a integração da *API* externa para a aquisição e inserção de dados, bem como a execução de *queries*, utilizando para isso Apex e SOQL;
- Front end onde será criada a *interface* do utilizador recorrendo ao LWC.

Todo o código exceto o referente aos testes encontram-se nos Anexos.

6.1. Objeto Pokémon

De forma a guardar os dados obtidos externamente a partir da integração com a PokeAPI, foi necessário criar um objeto Salesforce denominado Pokémon, assim como os campos adequados aos dados que serão recebidos. É apresentado em seguida, utilizando o *Schema Builder* da plataforma Salesforce, o objeto criado.

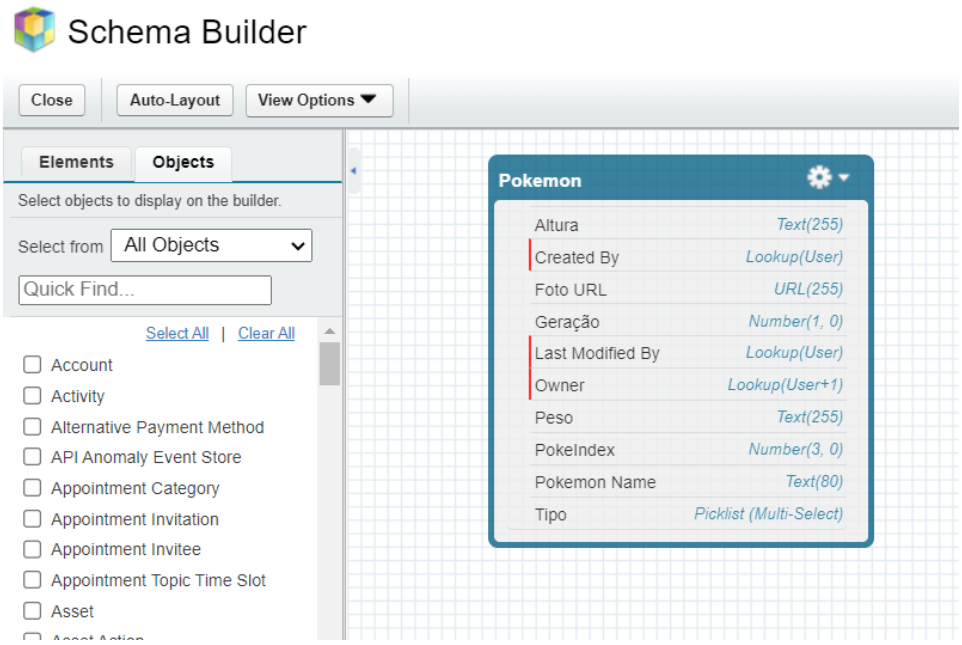


Figura 6 - Objeto Pokemon visualizado no Schema Builder

6.2. Permissões

Uma vez que a Salesforce leva muito a sério a segurança, existe a necessidade de garantir permissão para que o código possa aceder a uma *API* externa. Para tal é necessário adicionar os *endpoints* na plataforma. Para tal são necessários seguir os seguintes passos:

- Localizar no menu *Remote Site Settings* - Figura 7;

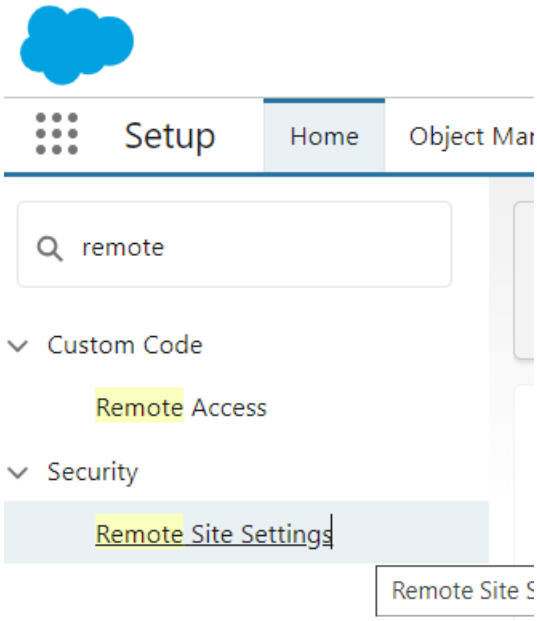


Figura 7 - Aceder à configuração de site remoto

- Selecionar *New Remote Site* e introduzir o URL do *end point* assim como o nome pelo qual será identificado. Basta introduzir apenas o endpoint inicial uma vez que as restantes subpastas serão também autorizadas - Figura 8.

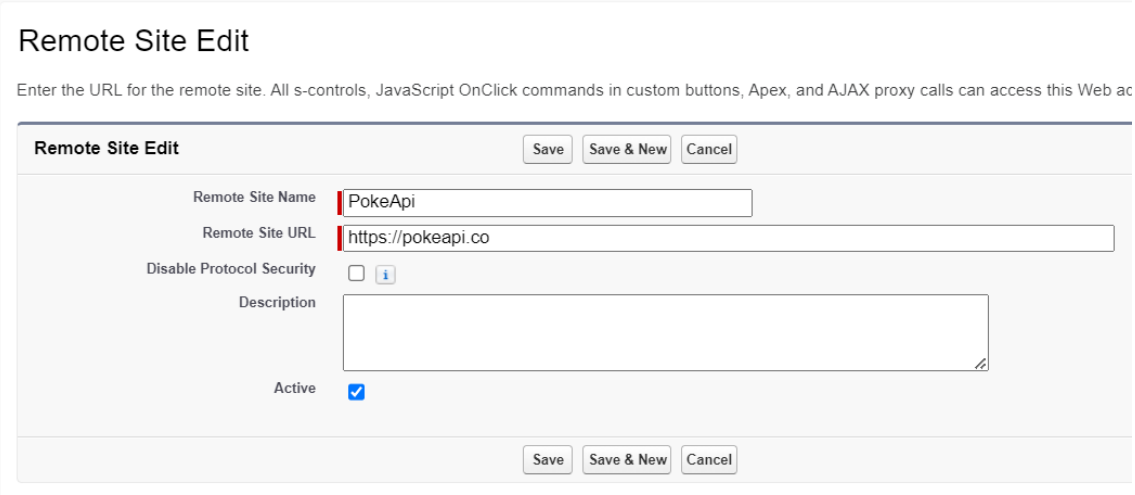


Figura 8 - Inserir endpoint a ser autorizado

6.3. Back end

Este subcapítulo tem como objetivo demonstrar as soluções implementadas para a parte referente ao *back end*, nomeadamente a aquisição de dados e a comunicação com a Poke*API*.

6.3.1. Calls e inserção de dados

Uma vez que a Salesforce impõe algumas restrições, habitualmente designadas por *governor limits,* foi necessário encontrar uma forma de contornar essas mesmas restrições.

No caso do projeto desenvolvido, essas limitações eram referentes ao número de *calls* permitidas por transação e ao tempo de utilização *de Central Processing Unit(CPU)*, uma vez que seriam necessárias pelo menos 1810 *calls*, pelo que a utilização de métodos assíncronos seria a mais aconselhada. Após uma análise cuidada, foi verificado que a solução passava pela utilização de processamento em *batchs* usando uma *Apex Batch Class*.

Uma *Apex Batch Class* é uma classe que permite o processamento de um grande volume de registos de uma forma eficiente e controlada. Ela implementa a *interface* '*Database.Batchable'* que contém três métodos:

- *'start'*: método responsável por iniciar o processo do processamento em *batch*, é executado apenas uma vez no início e tem como principal objetivo preparar e retornar um conjunto de registos a serem processados, quer por método de uma consulta *SOQL* quer pela criação de uma lista ou iterável;
- *'execute':* o núcleo do processamento em *batch*, é executado várias vezes para processar os registos retornados pelo método *'start'*. Cada execução recebe um lote de registos e efetua as ações necessárias sobre eles;
- *'finish'*: é executado após a conclusão bem-sucedida do processamento em lote. Ele é usado para efetuar quaisquer ações de limpeza ou pós-processamento necessárias.

No entanto a solução inicial apresentada ao aluno não era mais eficiente uma vez que pretendia utilizar uma *Apex Batch Class* para efetuar uma iteração de cada vez e efetuar uma *call* e a sua inserção em cada uma dessas iterações.

Após um estudo mais profundo e o auxílio da comunidade Salesforce foi possível melhorar essa mesma solução de forma a ter um melhor aproveitamento do processamento por *batch.*

Inicialmente foi utilizado método *'start'* para efetuar uma *call* à *API* e foi retornar um objeto *JSON* (Anexo A 4) com os nomes de todos os Pokémons e seu respetivo *endpoint* criar uma lista contendo os nomes de todos os Pokémons.

No método *'execute'* foi utilizada lista retornada no método *'start'* para iterar sobre todos os *endpoints* e realizar as *calls* necessárias para a obtenção dos dados. As respostas são então desserializadas em 'Map<Key,Value>', o qual será navegado de forma a extrair os dados necessários utilizando para isso *keys*. Ao mesmo tempo é criada uma instância do objeto Pokémon que por sua vez é inserida numa lista.
Por fim essa lista será inserida no objeto Pokémon previamente criado na plataforma.
Uma vez que estamos a utilizar uma *Apex Batch Class* a inserção será efetuada de forma parcial com cada *batch*, isto é, supondo que serão utilizados *batchs* de 50 registos, a cada 50 registos processados será inserida a lista com esses mesmos registos.

O método *'finish'* contém apenas com uma mensagem de depuração para que o administrador tenha conhecimento que a execução da classe *batch* chegou ao fim (Anexo A 5).

6.3.2. SOQL

Para poder visualizar os dados foram criados componentes *LWC*, mas foi necessário carregar esses dados através de um controlador contendo uma *querie SOQL*. Recorrendo às melhores práticas no desenvolvimento de software foram desenvolvidos três controladores de forma a manter o código mais organizado, criando uma estrutura em camadas que segue o princípio da "separação de preocupações" (DevMedia, s.d.) (Anexo A 6).

6.3.3. Stats Call

Como mencionado anteriormente, foi incluída uma *feature* que permite visualizar os *stats* de cada pokémon. Para isso foi desenvolvida uma classe Apex com um método que recebe uma *string* como parâmetro, mais concretamente o nome do pokémon, e executa uma *call* à *API* para retornar o objeto *JSON* com os detalhes referentes a pokémon. Serão extraídos desse objeto os dados referentes aos *stats,* que serão depois guardados numa coleção, mais concretamente um *map*, para serem enviados para o componente LWC pokeStats (Anexo A 7).

A Figura 9 apresenta os *stats* que serão visualizados no componente *LWC*.

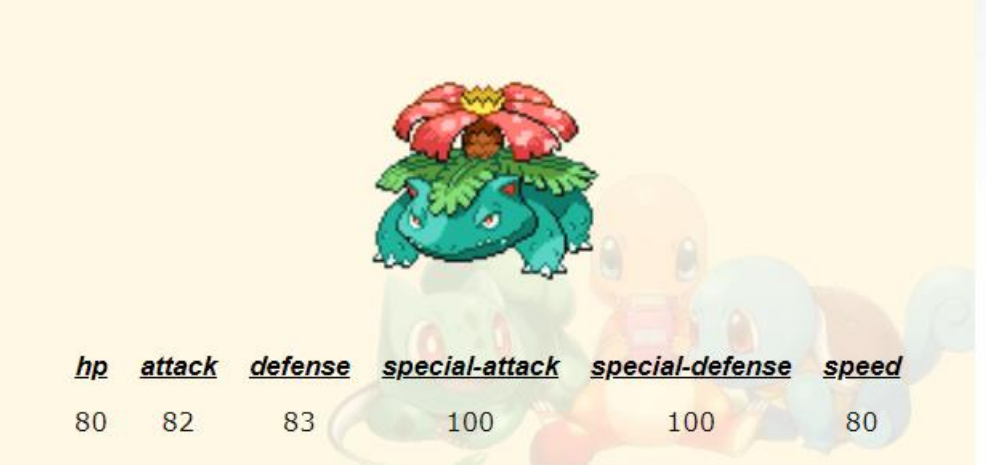


Figura 9 - Stats a serem visualizados no Componente

6.3.4. Trigger

Para assegurar que o código de inserção de dados no objeto Salesforce não seja executado mais de uma vez, seja por engano do administrador, tentativa de inserção manual ou possível duplicação de dados na API, foi implementado um Trigger. Esse Trigger foi projetado para evitar a duplicação de registros no objeto Pokémon (Anexo A 8).

6.4. Front End

Neste subcapítulo irá ser abordada a parte referente ao desenvolvimento da *interface* utilizando os componentes *LWC* e o seu *framework.*

Cada componente é composto obrigatoriamente por um ficheiro *JS* e um ficheiro *HTML*, onde o ficheiro *JS* trata da lógica e do comportamento do componente, enquanto o ficheiro *HTML* é o responsável pela estrutura e design do componente.

6.4.1. Componente pokedex

Este é o componente *LWC* principal do projeto, uma vez que é o componente onde serão visualizados os dados adquiridos. Foi usado o princípio de encapsulamento de componentes e a utilização do decorador '*@api*' que permite transformar propriedades privadas em propriedades públicas, de forma a serem acessíveis pelo componente pai.

É no ficheiro *JS* que vão ser carregados e manipulados os dados do objeto de forma a que possam ser visualizados.

Foram implementadas também algumas funções do *framework LWC*, tais como a '*getPicklistValues'*, que simplifica o acesso a dados de uma *picklist* de um objeto, ou a '*getObjectInfo'*, que permite obter informações sobre um objeto, os seus campos, *layouts* ou metadados.

Ambas usam o decorador *'@wire'* de forma a facilitar a procura e manipulação de dados de forma reativa, ou seja, os dados são alterados automaticamente quando existe alterações nos dados relacionados.

Para o projeto a sua utilização serviu para adquirir a informação do objeto Pókemon e usar essa informação para retornar os valores do *field* Tipo__c através da função '*getPicklistValues'*, e que serão guardados num *map* e serão posteriormente utilizados no preenchimento da *'Lightning Combobox'* para o filtro por Tipo(s), sendo esse preenchimento feito de forma ascendente graças à utilização dos métodos '*sort'* e

‘*localeCompare’*. Uma vez que existe a necessidade de selecionar um ou dois tipos e considerando que a Salesforce não dispõe de um componente *select* com formato *dropdown* que permita selecionar vários valores existe a necessidade de criar um componente que satisfaça essa falha. Dado que na comunidade Salesforce se encontram disponíveis componentes que resolvam esse problema, foi utilizado um desses mesmo componentes devidamente adaptado ao contexto do projeto (Schwarz, s.d.).

Para as opções de filtro por geração foi desenvolvido um método que a partir de um *set* preenchido durante a execução do método que recebe e trata os dados do *back end*, cria um *array* de objetos com dados referentes às gerações.

Para a implementação do método de pesquisa consideraram-se dois pontos:

- Primeiro quando o filtro se encontra vazio, isto é, quando não foi selecionado qualquer valor nas *‘Combobox’* ou inserido qualquer termo a pesquisar, será devolvida a lista de Pokémons na sua totalidade;
- Quando é feita a seleção de algum ou ambos os filtros, ou é inserido algum termos de pesquisa, essa seleção ou inserção será guardada em variáveis que serão utilizadas pela função *‘filter’* aplicada à lista de Pokémons.

A implementação do contador de registos foi efetuada recorrendo ao método ‘*Object.keys’* e à propriedade ‘*length*’.

Foi também utilizada a funcionalidade *‘NavigationMixin’* de forma a facilitar o redireccionamento da página.

Quanto ao *HTML*, foi utilizado o componente nativo ‘*lightning-layout’* para definir o layout como *grid*, sendo esta criada de forma dinâmica, iterando ao longo da lista de objetos recebidos através da função de filtragem. Cada espaço da grelha vai ser preenchido por outro componente *LWC* criado para o efeito e que recebe como parâmetro o objeto Pokémon referente a cada iteração realizada no componente *‘lightning-layout’*, assim como um evento que será utilizado para a implementação do redireccionamento.

O design do componente envolve ainda um ficheiro *CSS* de forma a customizar os *containers* que recebem as mensagens de erro e a *navbar* onde se encontram as opções de pesquisa e o contador de registos (Anexo A 9).

6.4.2. Componente pokemonTile

O componente pokemonTile tem como objetivo criar o design das *cards* que irão mostrar os dados de cada registo. Vai estar encapsulado no componente pokedex, a partir do qual recebe um objeto por parâmetro, acedendo e mostrando as propriedades do objeto. É efetuada também uma iteração na propriedade Tipo__c, uma vez que cada pokémon pode ter 1 ou 2 tipos e que contém um *array* de objetos onde a cada tipo está associada uma *template string* que representa uma classe CSS (Anexo A 10).

O ficheiro JS implementa:

- um método *getter* para que retorna uma *template string* que representa uma classe de CSS;
- um método que cria o evento que retorna o *id* do registo selecionado, para ser utilizado no método que efetua o redireccionamento.

6.4.3. Componente multiSelectCombobox

Como foi já referido, existe a necessidade de colmatar a falta de um componente *multiselect* com formato *dropdown*. Dada a existência de componentes customizados na comunidade Salesforce, essa necessidade foi solucionada reutilizando e adaptando um desses componentes.

Tal como o componente pokemonTile, este componente foi encapsulado no componente pokedex.

Ele é um *input* que será expandido numa lista recebe as opções a serem visualizadas, bem como um método que irá tratar as eventuais alterações associadas a um *change event*. Contém também *pills* para saber quais foram as opções selecionadas.

Quando é selecionado o *input* é expandida a lista e efetuada uma cópia das opções passadas para o componente para a uma lista.

Quando é efetuada uma seleção vai ser aplicado um filtro a essa lista de forma a encontrar a opção a que corresponde o item que disparou o evento, atualizando o seu estado de seleção com o valor obtido através do evento, assim como executando o método que atualiza os *pills* e disparando um evento que retorna os valores para o componente pai, que os utiliza na filtragem de opções (Anexo A 11).

6.4.4. Componente mutiSelectComboboxItem

Este componente está encapsulado no componente multiSelectCombobox e é composto por uma lista dinâmica de opções, e que vai receber as opções selecionadas e que serão enviadas para o componente pai. O ficheiro *JS* implementa dois métodos (Anexo A 12):

- um método *getter* que retorna uma *template string* que usa um operador ternário para verificar se o item está ou não selecionado;
- um método que cria e dispara um evento cujo objetivo é atualizar as propriedades do *Details* do evento.

6.4.5. Componente pokeStats

Este componente implementa um caso de uso que não constava nas *user stories* iniciais, mas que após uma troca de ideias com o supervisor do projeto foi decidido incluir no projeto.

Ele estende o caso de uso relacionado com o redireccionamento da página quando o utilizador clica na imagem presente na *card* com os dados do pokémon.

Após ser clicada a imagem, é disparado um evento no componente pokemonTile que retorna o valor do *id* do registo associado a que essa imagem pertence. Esse valor é depois recebido no componente pokedex e utilizado para efetuar o redireccionamento recorrendo à funcionalidade *'NavigationMixin'.* Uma vez na página de registo é utilizada função *'getRecord'* para retornar os valores dos *fields* Name e Foto_URL__c do respetivo registo.

O valor do *field* Name é depois utilizado para efetuar uma *call* à *API* recorrendo a um método de uma classe Apex criada para o efeito, para retornar os *stats* desse pokémon. Esse método devolve os dados num formato *map* e dado que para os visualizar no *HTML* foi necessário transformar esse *map* numa lista de *maps (Anexo A 13).*

7. Testes

A Salesforce impõe que pelo menos 75% de todo o código Apex esteja coberto por testes. Nesse sentido foram desenvolvidos testes para todas as classes Apex.

Para realizar os testes das classes que efetuam a integração à *PokeAPI*, foram criadas *mockups* de forma a garantir independência de recursos externos, bem como maior rapidez e eficiência dos testes.

Os testes foram realizados utilizando a *Developer Console* da plataforma Salesforce.

7.1. PokemonBatchTest

Aqui é apresentado o código para a realização do teste à Apex Batch Class que faz a integração com a *PokeAPI* e insere os dados, assim como o resultado do teste realizado.

7.1.1. Mockup Class

A Figura 10 apresenta a classe PokemonBatchMock. Esta implementa a interface '*HttpCalloutMock'* que é usada para simular respostas *HTTP* em testes de unidade.

O método *'respond'* é chamado sempre que um teste faz uma chamada *HTTP*. Este método recebe o *'HttpRequest'* como parâmetro e retorna um *'HttpResponse'*.

O código verifica o endpoint da solicitação HTTP e, com base nisso, define a resposta:

- Se o *endpoint* contém 'pokemon?limit=905', a resposta será uma lista de Pokémons (Bulbasaur, Charmander, Squirtle);
- Se o *endpoint* contém 'pokemon/', a resposta será os detalhes de um pokemon específico (id, altura, peso, tipos, espécies e sprites);
- Se o *endpoint* contém 'pokemon-species/', a resposta será os detalhes da espécie do pokemon (geração).

```
@isTest
public with sharing class PokemonBatchMock implements HttpCalloutMock {
  public HttpResponse respond(HttpRequest request) {
    HttpResponse res = new HttpResponse();
    res.setHeader('Content-Type', 'application/json');
    if (request.getEndpoint().contains('pokemon?limit=905')) {
      res.setBody(
        '{"results": [{"name": "bulbasaur"},' +
```

```
        '{"name": "charmander"},{"name": "squirtle"}]}'
      );
      res.setStatusCode(200);
    } else if (request.getEndpoint().contains('pokemon/')) {
      res.setBody(
        '{"id": 1, "height": 7, "weight": 69,' +
          '"types": [{"type": {"name": "grass"}},{"type": {"name":
"poison"}}],' +
          '"species": {"url": "https://pokeAPI.co/API/v2/pokemon-
species/1/"}, ' +
          '"sprites": {"front_default":' +
          '
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/1.p
ng"}}'
      );
      res.setStatusCode(200);
    } else if (request.getEndpoint().contains('pokemon-species/')) {
      res.setBody(
        '{"generation": {"url": "https://pokeAPI.co/API/v2/generation/1/"}}'
      );
      res.setStatusCode(200);
    }
    return res;
  }
}
```

Figura 10 - Excerto de código da classe PokemonBatchMock

7.1.2. Test Class

Na Figura 11 é apresentado o excerto de código referente à classe de teste PokmenonBatchTest. Ela tem como objetivo testar o funcionamento da classe Apex PopulatePokemonBatch. Inicialmente são criadas instâncias das classes PokemonBatchMock que serve para simular as respostas *HTTP*, e PopulatePokemonBatch que é a classe a ser testada.

Depois são configurados os dados a serem utilizados e a instância do *mock.*

É então colocado entre os métodos *'Test.startTest'* e *'Test.stopTest'* o código que permite testar a inserção de dados na PokeAPP através da classe PopulatePokemonBatch.

O método *'start'* da classe *batch* é iniciado através de *'Iterable<String> scope = batch.start(null)'* , sendo chamado o método *execute* e passados os dados do teste para o método *'batch.execute(null, testData)'* .

Após a sua execução, é realizada uma consulta á base de dados usando *SOQL*, retornando todos os registos para uma lista, sendo depois efetuada uma verificação sobre o seu tamanho dessa mesma lista.

```
@isTest
public with sharing class PokmenonBatchTest {
  @isTest
    static void testPokemonbatch()
    {
        PokemonBatchMock mock = new PokemonBatchMock();
        PopulatePokemonBatch batch = new PopulatePokemonBatch();
        Test.setMock(HttpCalloutMock.class, mock);
        List<String> testData = new List<String>{
        'bulbasaur',
        'ivysaur',
        'venusaur',
        'charmander',
        'charmeleon',
        'charizard',
        'squirtle',
        'wartortle',
        'blastoise',
        'caterpie'
        };

        Test.startTest();
        Iterable<String> scope = batch.start(null);
        batch.execute(null, testData);
        Test.stopTest();
        List<Pokemon__c> pokemonList = [
        SELECT Name, Peso__c, Altura__c, Tipo__c, Geracao__c, Foto_URL__c
        FROM Pokemon__c];
        System.assertEquals(10, pokemonList.size());
    }
}
```

Figura 11 - Excerto de código da classe PokmenonBatchTest

7.1.3. Resultados

A implementação e realização do teste unitário é importante para garantir que a classe *PopulatePokemonBatch* funciona como esperado e que a inserção de dados é feita corretamente. Como mostra a Figura 12, o teste foi executado com sucesso, não contendo erros e abrangendo a quase totalidade do código da classe.

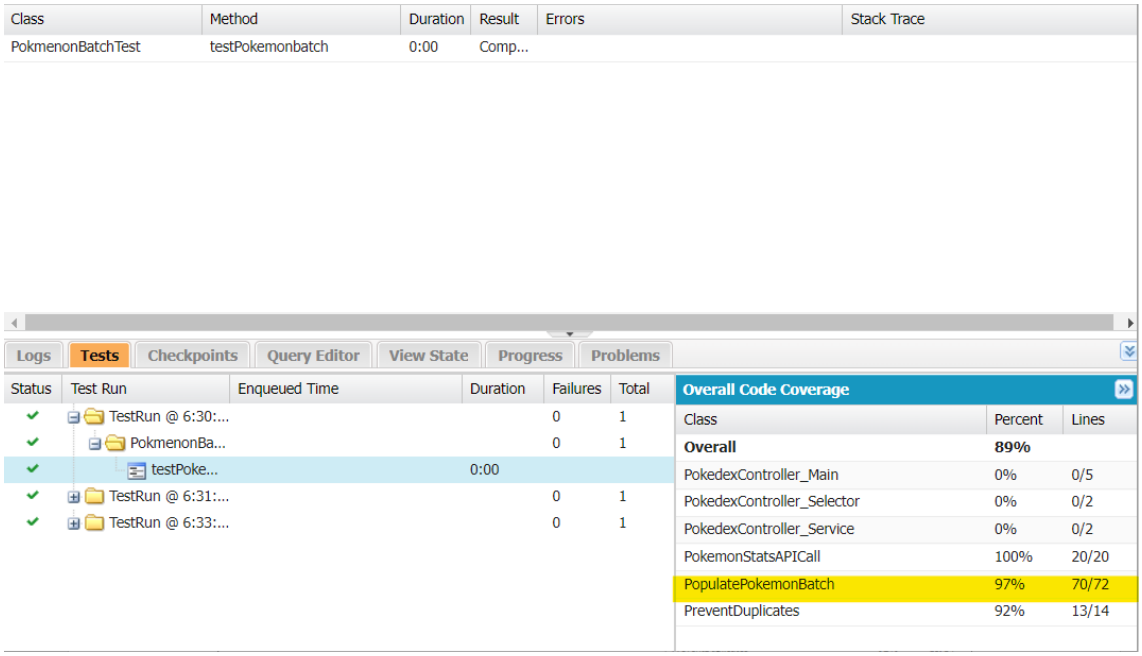


Figura 12 - Resultado do Teste PokemonBatchTest

7.2. PokemonStatsCallTest

Apresenta-se o código do teste e resultados obtidos, referentes à classe Apex que realiza a integração da *PokeAPI* para obter os dados referentes aos *stats* dos Pokémons.

7.2.1. Mockup Class

É apresentado na Figura 13 o código do teste em Apex que simula respostas *HTTP* para chamadas externas. A classe *PokemonStatsCallMock* implementa a *interface 'HttpCalloutMock'* que é usada para simular respostas *HTTP* em testes de unidade.

O método *'respond'* é chamado sempre que um teste faz uma *call HTTP*. Este método recebe o *'HttpRequest'* como parâmetro e retorna um *'HttpResponse'*.

O código define o corpo da resposta, o código de status e o cabeçalho.

```
@isTest
public with sharing class PokemonStatsCallMock implements HttpCalloutMock {
    public HttpResponse respond(HttpRequest request){
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type', 'application/json');
        res.setBody('{"stats":[{"stat":{"name":"speed"},'+
        '"base_stat":70},{"stat":{"name":"attack"},"base_stat":90}]}');
        res.setStatusCode(200);
        return res;
    }
}
```

Figura 13 - Excerto de código da classe PokemonStatsCallMock

7.2.2. Test Class

A classe de teste apresentada na Figura 14Figura 15, tem como objetivo verificar e validar o funcionamento do método *'getStats'* presente na classe *PokemonStatsAPICall*.

É criada e configurada a instância do *mock* a utilizar, seguindo-se a chamada ao método a ser testado.

Por fim é verificado a consistência dos resultados obtidos.

```
@isTest
public with sharing class PokemonStatsCallTest {
  @isTest
  static void testStatsCall() {
    PokemonStatsCallMock mock = new PokemonStatsCallMock();
    Test.setMock(HttpCalloutMock.class, mock);
    Map<String, Integer> statsMap = PokemonStatsAPICall.getStats('pikachu');
```

```
    System.assertEquals(2, statsMap.size());
    System.assertEquals(70, statsMap.get('speed'));
    System.assertEquals(90, statsMap.get('attack'));
  }
}
```

Figura 14 - Excerto de código da classe PokemonStatsCallTest

7.2.3. Resultados

A implementação e realização do teste unitário é importante para garantir que a classe PokemonStats*API*Call funciona como esperado, efetuando a integração com a *API* e recebendo os dados adequados. A Figura 15 apresenta o resultado obtido, onde se pode verificar que o teste foi executado com sucesso. Toda a classe foi abrangida pelo teste, obtendo 100% na cobertura do teste unitário.

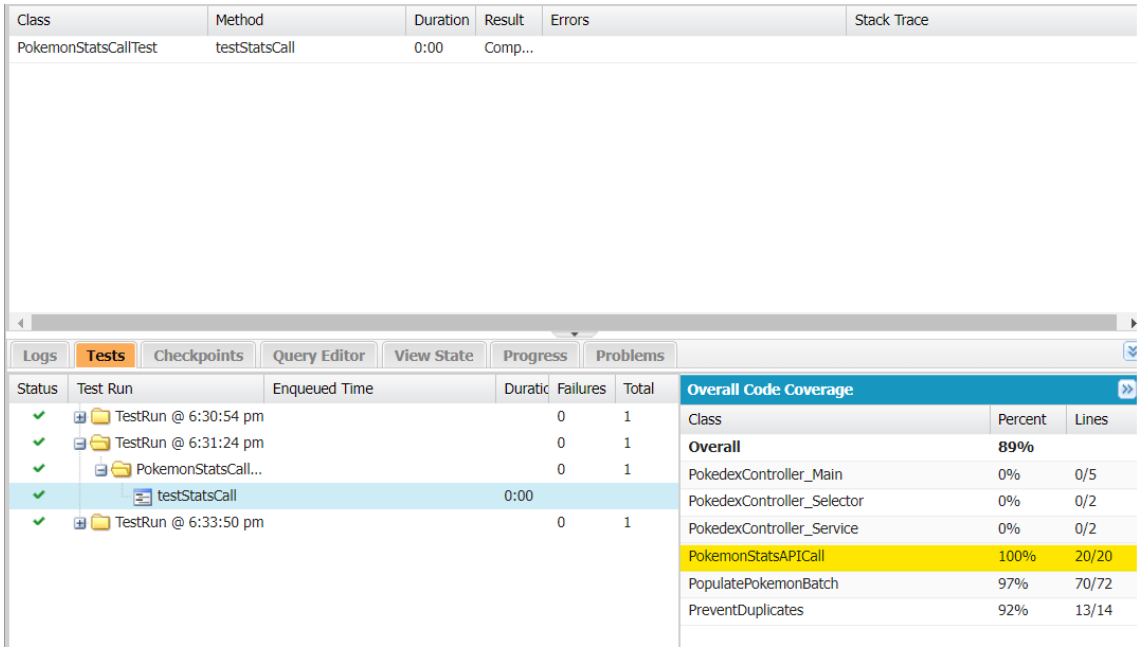


Figura 15 - Resultado do Teste PokemonStatsCallTest

7.3. TestInsertTrigger

Neste subcapítulo são apresentados o código e os resultados do teste referentes *Trigger* desenvolvido para evitar a inserção de dados duplicados.

7.3.1. Test Class

Por forma a testar se o *Trigger* desenvolvido de forma a evitar duplicação de dados funciona como seria de esperar, foi criado o teste cujo código é apresentado na Figura 16.

Inicialmente são criadas duas instâncias do objeto Pokemon__c onde é preenchido o único campo que não pode ter valores duplicados, neste caso o *Name.* Foi utilizado um nome em *uppercase* e outro em *lowercase* de forma a mostrar também que a introdução de valores nesse mesmo campo não é *case sensitive*. É também efetuada a inserção de uma das instâncias, sendo efetuada a inserção da outra dentro dos métodos *'startTest'* e *'stopTest'*, de forma a garantir que o teste não falhe devido a limites de recurso.

São depois verificadas as afirmações de forma validar os dados obtido no teste.

```
@istest
private class TestInsertTrigger {
  @isTest
  static void PokemonInsertTriggerTest() {
    Pokemon__c pokemon1 = new Pokemon__c(Name = 'Pikachu');
    insert pokemon1;
    Pokemon__c pokemon2 = new Pokemon__c(Name = 'pikachu');
    Test.startTest();
    Database.SaveResult insertResult = Database.insert(pokemon2, false);
    Test.stopTest();
    //* Check results */
    System.assert(!insertResult.isSuccess());
    System.assert(insertResult.getErrors().size() > 0);
    system.assertEquals(
      'Duplicate record found. Please check the data.',
      insertResult.getErrors()[0].getMessage()
    );
  }
}
```

Figura 16 - Excerto de código da classe TestInsertTrigger

7.3.2. Resultados

A implementação e realização do teste unitário é importante para garantir que o *Trigger PreventDuplicates* de facto previne a duplicação de dados, mesmo que seja através da inserção manual por parte do administrador.

É possível verificar através da Figura 17, que o teste foi realizado com sucesso, obtendo uma *coverage* de 92%.

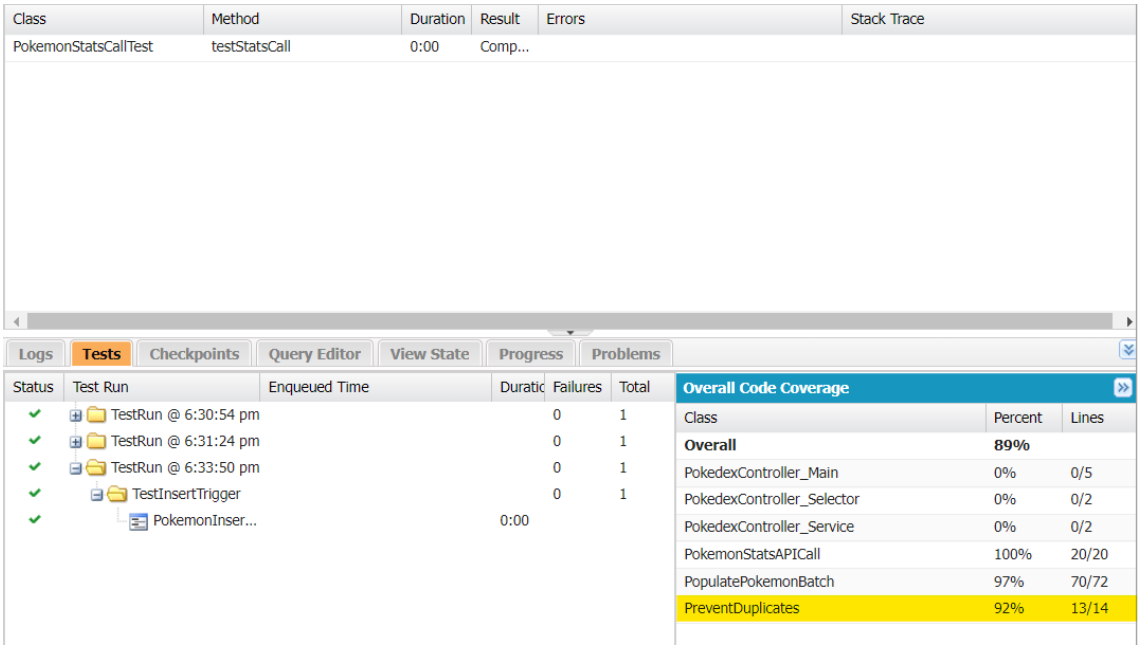


Figura 17 - Resultado do Teste TestInsertTrigger

Como se pode verificar foi obtido um *overall code coverage* de 89%, bem acima dos 75% impostos pela Salesforce.

8. Conclusão

Este projeto teve como objetivo o desenvolvimento de uma aplicação na plataforma Salesforce integrando uma *API* externa para a aquisição de dados. Inicialmente foi utilizada a plataforma Trailhead de forma a serem adquiridos os conhecimentos básicos necessários para o desenvolvimento do projeto.

Foi um projeto interessante uma vez que permitiu abordar temas como a integração de *APIs*, a utilização de *JS*, algo que deveria ser abordado ao longo da Licenciatura, e a utilização da plataforma Salesforce e as suas potenciais vantagens para os seus clientes.

Ao longo do desenvolvimento foi interessante perceber que o projeto poderia ser mais eficiente se na integração da *API* tivesse sido utilizado *GraphQL* em vez de *REST*, por forma a contornar os limites impostos pela Salesforce, embora fosse menos flexível na manipulação dos dados, razão pela que a escolha final foi a utilização de *REST.*

Ainda assim, a solução inicialmente indicada pelo supervisor para contornar esses limites não era a mais eficiente, pelo que após algum estudo mais aprofundado do aluno sobre a questão, foi possível melhorar significativamente a performance da aplicação no que à integração e inserção de dados diz respeito, uma vez que do processamento de mais de mil *batchs*, para o processamento de menos de vinte *batchs*, uma vez que passou a inserir cinquenta registos de cada vez em vez de apenas um. A forma encontrada para essa melhoria foi efetuar uma *call* à *API* no que retorna uma lista com o nome de todos os Pokémons, lista essa que é passada para o método *'execute'* e que será iterada de forma fazer as *calls* de acordo com o nome do pokémon.

Sendo que os requisitos propostos forma implementados com sucesso, existem pequenas questões que poderiam ter sido implementadas ou melhoradas tal como a criação de uma *VisualForce Page* com um botão para executar o código necessário para a integração com *API* e subsequente inserção dos dados, em vez da execução manual do método.

Bibliografia

AltexSoft. (2023). Obtido de Comparing API Architectural Styles: SOAP vs REST vs GraphQL vs RPC: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/

AmazonWebServices. (2023). *O que é uma API?* Obtido de https://aws.amazon.com/pt/what-is/api/

Apex, S. (2023). *What is Apex?* Obtido de https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.htm

AWS. (2023). *Qual é a diferença entre gRPC e REST?* Obtido de https://aws.amazon.com/pt/compare/the-difference-between-grpc-and-rest/

Cofes. (2023). *Is Salesforce SaaS, PaaS or IaaS? The Basic Differences (2023)*. Obtido de https://cofes.com/salesforce-saas-paas-iaas/

CSS, M. . (s.d.). *CSS: Cascading Style Sheets*. Obtido de https://developer.mozilla.org/en-US/docs/Web/CSS

DevMedia. (s.d.). *Amadurecendo com Separation Of Concerns*. Obtido de https://www.devmedia.com.br/amadurecendo-com-separation-of-concerns/18699

Hallett, P. (s.d.). *PokeAPI*. Obtido de PokeAPI: https://pokeapi.co/

HINC. (2023). *O que é a metodologia Kanban?* Obtido de https://hinc.com.br/blog/metodologia-kanban/

HTML, M. . (s.d.). *HTML: HyperText Markup Language*. Obtido de https://developer.mozilla.org/en-US/docs/Web/HTML

Idwall. (2018). *Qual a diferença entre API e web service?* Obtido de https://blog.idwall.co/qual-a-diferenca-entre-api-e-web-service/#penci-Vantagens-da-API

IEBS. (2023). *O que é a metodologia Kanban e como utilizá-la?* Obtido de https://www.iebschool.com/pt-br/blog/empreendedores-e-gestao-empresarial/agile-e-scrum/o-que-e-a-metodologia-kanban-e-como-utiliza-la/

Microsoft. (2023). *VS Code Docs*. Obtido de https://code.visualstudio.com/docs

Salesforce. (2023). *Which API Do I Use?* Obtido de https://help.salesforce.com/s/articleView?id=sf.integrate_what_is_api.htm&type=5

Salesforce, P. (2023). *Hyperforce*. Obtido de https://www.salesforce.com/products/platform/hyperforce/

Schwarz, S. (s.d.). *How to Create the LWC Multi-Select Combobox that Salesforce is Still Missing*. Obtido de https://javascript.plainenglish.io/how-to-create-the-lwc-multi-select-combobox-that-salesforce-is-still-missing-c7bf3a2850dd

Somerville, I. (2011). SOFTWARE ENGINEERING. Em I. Somerville, *SOFTWARE ENGINEERING,(p.58)* (9 ed.). Pearson Education, Inc.

Anexos

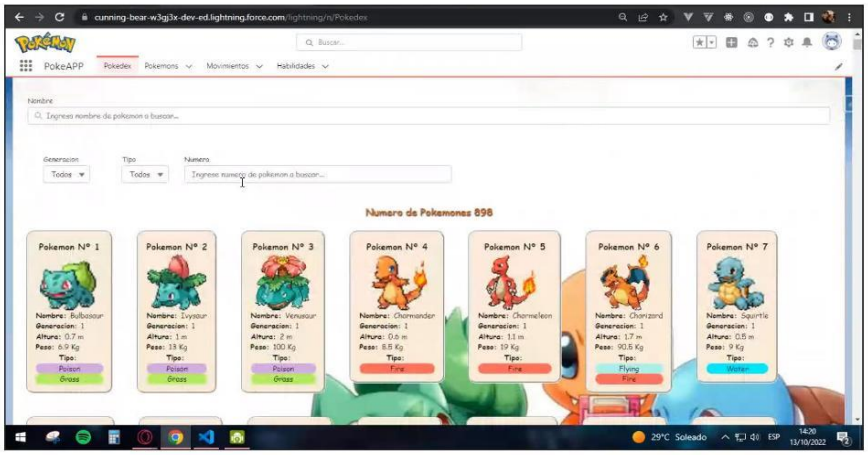
A 1. Enunciados

Olá Explorer! Como parte do processo de avaliação do curso **XDGen** - **Developers** apresentamos as seguintes histórias de usuários que correspondem ao exercício final, este exercício consiste em fazer uma aplicação para visualizar o **pokémon** dentro de um **LWC**, aplicando filtros e criando uma visão amigável para o usuário.

Para extrair os dados dos **pokémon**, eles devem utilizar a seguinte **API** pública: https://pokeapi.co/

A idéia principal é seguir cada uma das histórias de usuários, completá-las e que no final haja uma interface visual que contenha os filtros e a lista de **pokémon**, o **CSS** é livre, o participante pode colocar o quanto considerar necessário.

Exemplo de um exercício pokeApp de referência: (isto é apenas uma referência, não tem que ser igual).



HU 1 -Criação de objetos-

COMO administrador de Salesforce
QUERO criar o objeto Pokémon
PARA MANTER REGISTROS

HU 2 -Criação de campos-

COMO administrador de Salesforce
QUERO criar campos no objeto Pokémon
PARA ter mais detalhes sobre os registros

Critérios de aceitação
DESDE que eu queira ter mais detalhes sobre um registro
QUANDO eu faço um insert desse registro
ENTÃO devo ter os seguintes campos

- Geração (Number, tamanho 1);
- Altura (Texto, tamanho 255);
- Foto URL (URL, tamanho 255);
- PokeIndex (Number, tamanho 3);
- Peso(Texto, tamanho 255);
- Tipo (Multi-Select Picklist);

Valores para Tipo:
Normal,
Fighting,
Flying,
Poison,
Ground,
Rock,
Bug,
Ghost,
Steel,
Fire,
Water,
Grass,
Electric,
Psychic,
Ice,
Dragon,
Dark,
Fairy.

HU 3 -Permissão para a API-

COMO administrador de Salesforce
QUERO conectar-me a https://pokeapi.co/
PARA OBTER REGISTROS

Critérios de aceitação
DADO QUE quero conectar-me a https://pokeapi.co/
QUANDO a conexão é feita
ENTÃO o Salesforce deve ter as permissões necessárias

HU 4 -Chamada API-

COMO administrador do Salesforce
QUERO conectarme a https://pokeapi.co/
PARA INSERIR registros

Critérios de aceitação
DADO que me conectei a https://pokeapi.co/ com as permissões relevantes
QUANDO a conexão é feita
ENTÃO o Salesforce deve receber uma resposta do mesmo

HU 5 -Inserção de registros-

COMO administrador de Salesforce
QUIERO inserir registros
PARA lhes dar visibilidade na minha organização

Critérios de aceitação
DADO que posso me conectar a https://pokeapi.co/
QUANDO a conexão é feita
ENTÃO devo salvar os seguintes campos em um novo registro

- Nome (deve começar com letra maiúscula) ex. "Pikachu"
- Altura (seguido de "metros") por exemplo. "8 metros." (cálculo necessário de passagem de unidades)
- Peso (seguido de "kg.") ex. "64kg." (cálculo necessário de passagem de unidades)
- Foto (frente padrão ou escolha)
- Tipo (Tenha em mente que pode ter mais de 1 tipo)

HU 6 -Criar APP-

COMO administrador de Salesforce
QUERO criar a aplicação PokeAPP e Tab Pokedex
PARA ver os registros

Critérios de aceitação
Incluir a aba Pokedex no aplicativo PokeAPP

HU 7 -Visualizando registros-

COMO administrador de Salesforce
QUERO ver os registros
PARA montar a minha Pokédex

Critérios de aceitação
DADO que tenho os 898 registros inseridos corretamente na minha Org
QUANDO entro na aba Pokedex
ENTÃO devo exibi-los em linhas de 6, seguindo a ordem do índice de cada registro
dica investigue o for:each em HTML
No momento não é necessário aplicar estilos

HU 8 -Filtrado por lista de opções-

COMO administrador de Salesforce
QUERO filtrar registros
PARA refinar minha pesquisa

Critérios de aceitação
DADO que visualizo todos os Pokémon
QUANDO eu quiser restringir minha pesquisa
ENTÃO devo ter duas listas de visualização para refinar minha pesquisa
- Lightning-Combobox para filtrar os resultados por tipo
- Lightning-Combobox para filtrar os resultados por geração
* Devo ter "All" como primeira opção em ambas as listas de opções
* Posso filtrar por ambas as listas simultaneamente. Por exemplo, poderei procurar todos os Pokémon do Tipo Normal da geração 5

HU 9 -Filtrado por nome-

COMO administrador de Salesforce
QUERO filtrar registros
PARA refinar minha pesquisa
Critérios de aceitação
DADO que eu visualizo todos os Pokémon
QUANDO eu quiser restringir minha pesquisa
ENTÃO devo ter um campo de entrada para refinar minha pesquisa
-lightning-input para filtrar os resultados
* Esta filtragem pode ser feita simultaneamente com as listas de opções criadas anteriormente
Ex: eu digito a letra "a", a busca deve retornar todos os Pokémon que CONTÉM a letra "a", não necessariamente que comecem com a letra "a".

HU 10 -Contador de registros-

COMO administrador de Salesforce
QUERO mostrar o número de registros
PARA saber quantos Pokémon pertencem a uma geração, a um tipo, etc.

Critérios de aceitação
*Devo visualizar um contador que ao aplicar um filtro, conta o número de resultados
-hint- Investigue a função Object.keys em JavaScript

HU 11 -Estilos-

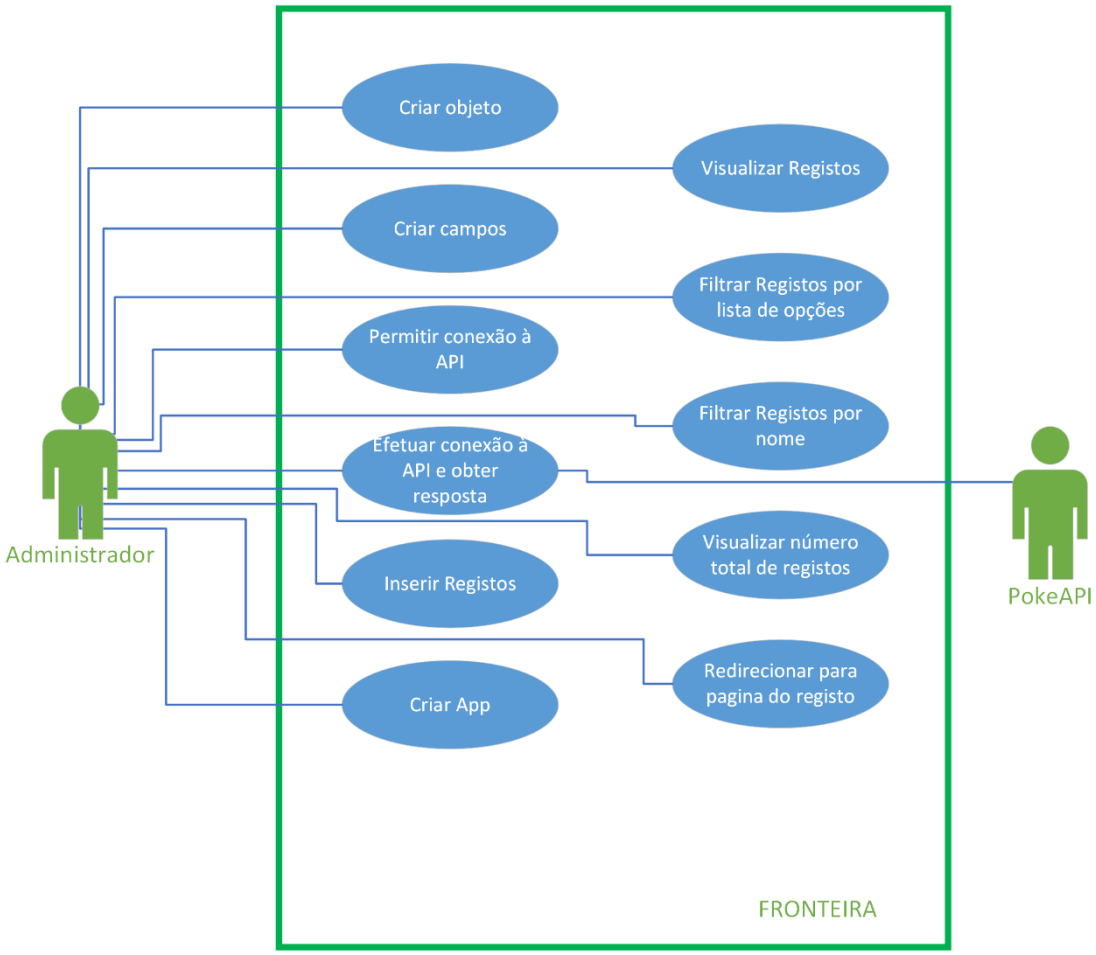
Critérios de aceitação
Terei que aplicar estilos na obra previamente construída para uma visualização mais bonita.
*Os estilos que serão usados ficam a critério pessoal, deixe a sua imaginação fluir livremente.

HU 12 -Redirecionamento para página de registro- (opcional)

COMO administrador de Salesforce
QUERO clicar na imagem do meu Pokémon
PARA me redirecionar para o registro do meu Pokémon

Critérios de aceitação
Depois de clicar na imagem do meu Pokémon, devo ser redirecionado para a sua página de registro
-dica- Investigue a função NavigationMixin em JavaScript

A 2. Diagrama de casos de uso inicial



A 3. Descrições de Casos de Uso

Tabela 4 - Descrição de caso de uso Criar Objeto

Nome	Criar Objeto
Descrição	Este caso de uso permite que um administrador crie um novo objeto no Salesforce.
Pré-Condição	O administrador está autenticado no Salesforce e possui permissões para criar o objeto desejado.
Caminho Principal	1. O administrador seleciona "*Object Manager*" e clica em "*New Object*"; 2. O sistema redireciona para a página de configuração do objeto; 3. O administrador preenche os campos necessários e seleciona "*Save*".
Caminho Alternativo	3. a) O administrador clica em *"Cancel";* 3. a) O administrador insere campos em branco ou não válidos.
Adornos ou Suplementos	-
Pós-Condições	Objeto criado com sucesso.

Tabela 5 - Descrição de caso de uso Permitir Ligação à API

Nome	Permitir Ligação à *API*
Descrição	Este caso de usos tem como objetivo descrever o processo que permite garantir o acesso à *API.*
Pré-Condição	O administrador está autenticado no Salesforce.
Caminho Principal	1. O administrador seleciona *"Home"* e localiza no submenu *"Security"* e seleciona *"Remote Site Settings";* 2. O sistema abre a página *"Remote Site Settings";* 3. O administrador seleciona *"New Remote Site";* 4. O sistema abre a página *"Remote Site Edit";* 5. O administrador introduz a *URL* do *endpoint* e o nome para identificar a permissão e clica em *"Save".*
Caminho Alternativo	5. a) O administrador clica em *"Cancel";* 5. b) O endereço introduzido não é o correto.
Adornos ou Suplementos	Efetuar um teste de ligação à *API.*
Pós-Condições	Permissão garantida com sucesso.

Tabela 6 - Descrição de caso de uso Efetuar conexão à API e obter resposta

Nome	Efetuar conexão à *API* e obter resposta.
Descrição	Este caso de usos descreve os passos a realizar para efetuar a conexão à *API.*
Pré-Condição	Permissão à *API* garantida.
Caminho Principal	1. O administrador executa o método; 2. O sistema executa o código que efetua os pedidos à *API:* 3. A *API* retorna as respostas ao sistema.
Caminho Alternativo	1. a) O administrador executa o método Errado; 2. a) O método não foi implementado corretamente; 3. a) A *API* é incapaz de responder.
Adornos ou Suplementos	Efetuar um teste de ligação à *API*; Testar o método.
Pós-Condições	Obtidos dados da *API* com sucesso.

Tabela 7 - Descrição de caso de uso Inserir Registos

Nome	Inserir Registos.
Descrição	Este caso de usos descreve os passos a realizar efetuar a inserção de dados.
Pré-Condição	Foi efetuada a ligação à *API* e obtidos os dados.
Caminho Principal	1. O sistema manipula os dados recebidos; 2. O sistema efetua a inserção dos dados no objeto.
Caminho Alternativo	1. a) O código que manipula os dados está incorreto; 2. b) Os dados não são compatíveis com os campos do objeto.
Adornos ou Suplementos	Testar o método; Verificar o tipo de dados do objeto.
Pós-Condições	Dados inseridos com sucesso no objeto.

Tabela 8 - Descrição de caso de uso Criar App

Nome	Criar App.
Descrição	Este caso de usos descreve os passos a realizar para a criação de uma App para a visualização de Pokémons.
Pré-Condição	O administrador está autenticado no Salesforce, tem permissão para criar App e o objeto já foi criado.
Caminho Principal	1. O administrador seleciona o *"App Manager"* no menu *"Home";* 2. O sistema abre a página *"Lightning Experience App Manager";* 3. O administrador seleciona *"New Lightning App";* 4. O sistema abre a página *"App Details & Branding"* da App; 5. O Administrador preenche os campos necessários e seleciona *"Next"* repetindo os passos nos *screens* seguintes, selecionando *"Save & Finish"* no após preencher os dados do *screen* final; 6. O administrador seleciona *"Tabs"* do submenu *"User Interface";* 7. O sistema abre a página *"Tabs";* 8. O administrador seleciona *"New"* na secção *"Custom Object Tabs"* e preenche os dados o objeto e clica *"Next"* até a página final onde após preencher os campos, seleciona *"Save"* .
Caminho Alternativo	5. a) O *"App Name"* introduzida pelo administrador já existe; 8. a) O administrador seleciona *"Cancel";* 8. b) Não existe nenhum *custom object*
Adornos ou Suplementos	-
Pós-Condições	App criada com sucesso.

Tabela 9 - Descrição de caso de uso Visualizar Registos

Nome	Visualizar Registos.
Descrição	Este caso de usos descreve os passos a realizar para a visualização de registos.
Pré-Condição	Criada a app e componente *LWC.*
Caminho Principal	1. O administrador acede à app; 2. O sistema abre a app na *tab* Pokedex onde se encontra o componente *LWC* que permite a visualização dos registos.
Caminho Alternativo	-
Adornos ou Suplementos	-
Pós-Condições	Registos visualizados com sucesso;

Tabela 10 - Descrição de caso de uso Filtra registos por listas de opções

Nome	Filtrar registos por listas de opções.
Descrição	Este caso de uso permite que o administrador filtre os resultados com base em várias opções disponíveis.
Pré-Condição	O administrador está autenticado no Salesforce e está na página onde a lista de registros é exibida.
Caminho Principal	1. O administrador seleciona a lista *"Types"* e/ou a lista *"Generations";* 2. O sistema abre a(s) lista(s) de opções; 3. O administrador seleciona uma ou mais opções; 4. O sistema atualiza os resultados para exibir apenas os itens que correspondem às opções selecionadas.
Caminho Alternativo	-
Adornos ou Suplementos	-
Pós-Condições	Os resultados exibidos são filtrados com base nas opções selecionadas pelo usuário.

Tabela 11 - Descrição de caso de uso Filtrar registos por nome

Nome	Filtrar registos por nome.
Descrição	Este caso de uso permite que o administrador filtre os resultados com base num *input*.
Pré-Condição	O administrador está autenticado no Salesforce e está na página onde a lista de registros é exibida.
Caminho Principal	1. O administrador introduz o termo a pesquisar; 2. O sistema valida e processa o termo de pesquisa inserido; 3. O sistema atualiza os resultados para exibir apenas os itens que correspondem ao termo inserido.
Caminho Alternativo	3. a) Se o termo de pesquisa inserido não corresponder a nenhum item, o sistema exibe uma mensagem indicando que não há resultados correspondentes.
Adornos ou Suplementos	-
Pós-Condições	Os resultados exibidos são filtrados com base no termo inserido.

Tabela 12 - Descrição de caso de uso Visualizar número total de registos

Nome	Visualizar número total de registos.
Descrição	Este caso de uso permite que o administrador visualize o número total de registos mostrados.
Pré-Condição	O administrador está autenticado no Salesforce e está na página onde a lista de registros é exibida.
Caminho Principal	1. O sistema mostra o número de total de registos mostrados, atualizando o valor após o administrador aplicar um filtro de pesquisa;
Caminho Alternativo	-
Adornos ou Suplementos	-
Pós-Condições	O administrador visualiza com precisão o número total de registros incluído após ter sido efetuada uma pesquisa.

Tabela 13 - Descrição de caso de uso Redirecionar para a página de registo

Nome	Redirecionar para a página de registo.
Descrição	Este caso de uso permite que o administrador seja redirecionado para a página do registo.
Pré-Condição	O administrador está autenticado no Salesforce e está na página onde a lista de registros é exibida.
Caminho Principal	1. O administrador clica na imagem presente na *card* do registo; 2. O sistema redireciona o administrador para a página de detalhes do registo.
Caminho Alternativo	-
Adornos ou Suplementos	O sistema pode permitir que o administrador edite o registro diretamente da página de detalhes, caso tenha permissões adequadas.
Pós-Condições	O administrador é redirecionado para a página de detalhes do registo.

Tabela 14 - Descrição de caso de uso Visualizar Stats

Nome	Visualizar stats.
Descrição	Este caso de uso estende o caso de uso "Redirecionar para a página de registo" e permite que o administrador visualize os *stats* do pokémon, efetuando uma *call* à *API*.
Pré-Condição	O administrador clicou no registo e foi redirecionado para a página de detalhes.
Caminho Principal	1. O sistema utiliza o nome do registo para efetuar uma *call* à *API* para obter os dados referentes aos *stats;* 2. A *API* retorna os dados pedidos. 3. O sistema exibe no componente *LWC* os dados obtidos.
Caminho Alternativo	2. a) A *API* não responde ou devolve dados errados.
Adornos ou Suplementos	-
Pós-Condições	O administrador consegue visualizar os *stats.*

A 4. Objeto JSON retornado no método 'start'


```
{"count":1292,"next":"https://pokeapi.co/api/v2/pokemon?offset=905&limit=387","previous":null,"results":[{"name":"bulbasaur","url":"h
{"name":"ivysaur","url":"https://pokeapi.co/api/v2/pokemon/2/"},{"name":"venusaur","url":"https://pokeapi.co/api/v2/pokemon/3/"},{"na
{"name":"charmeleon","url":"https://pokeapi.co/api/v2/pokemon/5/"},{"name":"charizard","url":"https://pokeapi.co/api/v2/pokemon/6/"},
{"name":"wartortle","url":"https://pokeapi.co/api/v2/pokemon/8/"},{"name":"blastoise","url":"https://pokeapi.co/api/v2/pokemon/9/"},
{"name":"metapod","url":"https://pokeapi.co/api/v2/pokemon/11/"},{"name":"butterfree","url":"https://pokeapi.co/api/v2/pokemon/12/"},
{"name":"kakuna","url":"https://pokeapi.co/api/v2/pokemon/14/"},{"name":"beedrill","url":"https://pokeapi.co/api/v2/pokemon/15/"},{"n
{"name":"pidgeotto","url":"https://pokeapi.co/api/v2/pokemon/17/"},{"name":"pidgeot","url":"https://pokeapi.co/api/v2/pokemon/18/"},
{"name":"raticate","url":"https://pokeapi.co/api/v2/pokemon/20/"},{"name":"spearow","url":"https://pokeapi.co/api/v2/pokemon/21/"},{"
{"name":"ekans","url":"https://pokeapi.co/api/v2/pokemon/23/"},{"name":"arbok","url":"https://pokeapi.co/api/v2/pokemon/24/"},{"name"
{"name":"raichu","url":"https://pokeapi.co/api/v2/pokemon/26/"},{"name":"sandshrew","url":"https://pokeapi.co/api/v2/pokemon/27/"},{"
{"name":"nidoran-f","url":"https://pokeapi.co/api/v2/pokemon/29/"},{"name":"nidorina","url":"https://pokeapi.co/api/v2/pokemon/30/"},
{"name":"nidoran-m","url":"https://pokeapi.co/api/v2/pokemon/32/"},{"name":"nidorino","url":"https://pokeapi.co/api/v2/pokemon/33/"},
{"name":"clefairy","url":"https://pokeapi.co/api/v2/pokemon/35/"},{"name":"clefable","url":"https://pokeapi.co/api/v2/pokemon/36/"},
{"name":"ninetales","url":"https://pokeapi.co/api/v2/pokemon/38/"},{"name":"jigglypuff","url":"https://pokeapi.co/api/v2/pokemon/39/"
{"name":"zubat","url":"https://pokeapi.co/api/v2/pokemon/41/"},{"name":"golbat","url":"https://pokeapi.co/api/v2/pokemon/42/"},{"name
{"name":"gloom","url":"https://pokeapi.co/api/v2/pokemon/44/"},{"name":"vileplume","url":"https://pokeapi.co/api/v2/pokemon/45/"},{"n
{"name":"parasect","url":"https://pokeapi.co/api/v2/pokemon/47/"},{"name":"venonat","url":"https://pokeapi.co/api/v2/pokemon/48/"},{"
{"name":"diglett","url":"https://pokeapi.co/api/v2/pokemon/50/"},{"name":"dugtrio","url":"https://pokeapi.co/api/v2/pokemon/51/"},{"n
{"name":"persian","url":"https://pokeapi.co/api/v2/pokemon/53/"},{"name":"psyduck","url":"https://pokeapi.co/api/v2/pokemon/54/"},{"n
{"name":"mankey","url":"https://pokeapi.co/api/v2/pokemon/56/"},{"name":"primeape","url":"https://pokeapi.co/api/v2/pokemon/57/"},{"n
{"name":"arcanine","url":"https://pokeapi.co/api/v2/pokemon/59/"},{"name":"poliwag","url":"https://pokeapi.co/api/v2/pokemon/60/"},{"
{"name":"poliwrath","url":"https://pokeapi.co/api/v2/pokemon/62/"},{"name":"abra","url":"https://pokeapi.co/api/v2/pokemon/63/"},{"na
{"name":"alakazam","url":"https://pokeapi.co/api/v2/pokemon/65/"},{"name":"machop","url":"https://pokeapi.co/api/v2/pokemon/66/"},{"n
{"name":"machamp","url":"https://pokeapi.co/api/v2/pokemon/68/"},{"name":"bellsprout","url":"https://pokeapi.co/api/v2/pokemon/69/"},
{"name":"victreebel","url":"https://pokeapi.co/api/v2/pokemon/71/"},{"name":"tentacool","url":"https://pokeapi.co/api/v2/pokemon/72/"
{"name":"geodude","url":"https://pokeapi.co/api/v2/pokemon/74/"},{"name":"graveler","url":"https://pokeapi.co/api/v2/pokemon/75/"},{"
{"name":"ponyta","url":"https://pokeapi.co/api/v2/pokemon/77/"},{"name":"rapidash","url":"https://pokeapi.co/api/v2/pokemon/78/"},{"n
{"name":"slowbro","url":"https://pokeapi.co/api/v2/pokemon/80/"},{"name":"magnemite","url":"https://pokeapi.co/api/v2/pokemon/81/"},
{"name":"farfetchd","url":"https://pokeapi.co/api/v2/pokemon/83/"},{"name":"doduo","url":"https://pokeapi.co/api/v2/pokemon/84/"},{"n
```

A 5. Calls e inserção

```
public class PopulatePokemonBatch implements Database.Batchable<String>,
Database.AllowsCallouts {
  public Iterable<String> start(Database.BatchableContext bc) {
    List<String> pokeList = new List<String>();
    //Make API call
    Http http = new Http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://pokeAPI.co/API/v2/pokemon?limit=905');
    request.setMethod('GET');
    HttpResponse response = http.send(request);
    // Check if Connection was successful
    if (response.getStatusCode() == 200) {
      //deserialize response
      Map<String, Object> responseData = (Map<String, Object>)
JSON.deserializeUntyped(
        response.getBody()
      );
      List<Object> results = (List<Object>) responseData.get('results');
      //get url of pokemon's endpoints
      for (Object result : results) {
        Map<String, Object> pokemonData = (Map<String, Object>) result;
        String pokeName = (String) pokemonData.get('name');
        pokeList.add(pokeName);
      }
    }
    return pokeList;
  }

  public void execute(Database.BatchableContext bc, List<String> scope) {
    try {
      List<Pokemon__c> pokemonList = new List<Pokemon__c>();
      for (String record : scope) {
        String pokeName = record;
        HttpRequest detailsRequest = new HttpRequest();
        detailsRequest.setEndpoint(
          'https://pokeAPI.co/API/v2/pokemon/' + pokeName
        );
        detailsRequest.setMethod('GET');
```

```
        HttpResponse detailsResponse = new Http().send(detailsRequest);
        // Check if Connection to the new endpoint was successful
        if (detailsResponse.getStatusCode() == 200) {
          //deserialize response
          Map<String, Object> detailData = (Map<String, Object>)
JSON.deserializeUntyped(
            detailsResponse.getBody()
          );
          // get fields from detail data
          //***** get the id
          Integer id = (Integer) detailData.get('id');
          //***** get and convert height and weight to the correct units
          Double height = (Double) detailData.get('height') / 10;
          Double weight = (Double) detailData.get('weight') / 10;
          // String name = (String) detailData.get('name');
          //***** get and handle multiple types
          List<Object> types = (List<Object>) detailData.get('types');
          List<String> typeList = new List<String>();
          for (Object type : types) {
            Map<String, Object> typeData = (Map<String, Object>) type;
            Map<String, Object> typeName = (Map<String, Object>)
typeData.get(
              'type'
            );
            String nameType = (String) typeName.get('name');
            typeList.add(nameType);
          }
          //***** get species url to adquire the generation
          Map<String, Object> species = (Map<String, Object>) detailData.get(
            'species'
          );
          String speciesUrl = (String) species.get('url');
          // make a call to the species endpoint
          HttpRequest speciesRequest = new HttpRequest();
          speciesRequest.setEndpoint(speciesUrl);
          speciesRequest.setMethod('GET');
          HttpResponse speciesResponse = new Http().send(speciesRequest);
          // Check if Connection to the new endpoint was successful
          if (speciesResponse.getStatusCode() == 200) {
            //deserialize response
```

```
            Map<String, Object> speciesDetails = (Map<String, Object>)
JSON.deserializeUntyped(
              speciesResponse.getBody()
            );
            //***** get the generation url and extract the the generation
number from the end
            Map<String, Object> generationDetails = (Map<String, Object>)
speciesDetails.get(
              'generation'
            );
            String generationUrl = (String) generationDetails.get('url');
            String generation = generationUrl.substring(
              generationUrl.length() - 2,
              generationUrl.length() - 1
            );
            //***** get the sprites
            Map<String, Object> sprites = (Map<String, Object>)
detailData.get(
              'sprites'
            );
            String spriteUrl = (String) sprites.get('front_default');
            //***** create a new pokemon object and insert the data extrated
fom the API
            Pokemon__c pokemon = new Pokemon__c(
              Name = pokeName.cAPItalize(),
              PokeIndex__c = id,
              Peso__c = String.valueOf(weight + ' kg'),
              Altura__c = String.valueOf(height + ' mts'),
              Tipo__c = String.join(typeList, ';'),
              Geracao__c = Integer.valueOf(generation),
              Foto_URL__c = spriteUrl
            );
            pokemonList.add(pokemon);
          }
        }
      }
      if (!pokemonList.isEmpty()) {
        insert pokemonList;
      }
    } catch (Exception e) {
      System.debug('Error: ' + e.getMessage());
```

```
    }
  }

  public void finish(Database.BatchableContext bc) {
    system.debug('batch finished');
  }
}
```

A 6. SOQL

```
public with sharing class PokedexController_Selector {

    public static List<Pokemon__c> getPokemons()
    {
        return [SELECT Id,Name,Altura__c,Foto_URL__c,Geracao__c,
                Peso__c,PokeIndex__c,Tipo__c FROM Pokemon__c
                ORDER BY PokeIndex__c ASC];
    }

}
```

```
public with sharing class PokedexController_Service {
  public static List<Pokemon__c> PokedexGetService() {
    return PokedexController_Selector.getPokemons();
  }
}
```

```
public with sharing class PokedexController_Main {
  @AuraEnabled(cacheable=true)
  public static List<Pokemon__c> PokedexGetMain() {
    try {
      return PokedexController_Service.PokedexGetService();
    } catch (Exception e) {
      throw new AuraHandledException(e.getMessage());
    }
  }
}
```

A 7. Stats Call

```
public with sharing class PokemonStatsAPICall {
  @AuraEnabled(cacheable=true)
  public static Map<String, Integer> getStats(String pokemon) {
    Map<String, Integer> statsMap = new Map<String, Integer>();
    Http http = new http();
    HttpRequest request = new HttpRequest();
    request.setEndpoint('https://pokeAPI.co/API/v2/pokemon/' + pokemon);
    request.setMethod('GET');
    HttpResponse response = http.send(request);
    if (response.getStatusCode() == 200) {
      Map<String, Object> responseData = (Map<String, Object>)
JSON.deserializeUntyped(
        response.getBody()
      );
      List<Object> stats = (List<Object>) responseData.get('stats');
      for (Object stat : stats) {
        Map<String, Object> statData = (Map<String, Object>) stat;
        Map<String, Object> statName = (Map<String, Object>) statData.get(
          'stat'
        );
        String name = (String) statName.get('name');
        Integer statValue = (Integer) statData.get('base_stat');
        statsMap.put(name, statValue);
      }
    }
    return statsMap;
  }
}
```

A 8. Trigger

```
trigger PreventDuplicates on Pokemon__c(before insert, before update) {
  Set<String> names = new Set<String>();
  for (Pokemon__c pokemon : Trigger.new) {
    names.add(pokemon.Name.toUpperCase());
  }

  List<Pokemon__c> existingPokemons = [
    SELECT Name
    FROM Pokemon__c
    WHERE Name IN :names
  ];
  Map<String, Pokemon__c> existingMap = new Map<String, Pokemon__c>();
  for (Pokemon__c existing : existingPokemons) {
    existingMap.put(existing.Name.toUpperCase(), existing);
  }

  for (Pokemon__c pokemon : Trigger.new) {
    if (
      existingMap.containsKey(pokemon.Name.toUpperCase()) &&
      (pokemon.Id == null ||
      pokemon.Id != existingMap.get(pokemon.Name.toUpperCase()).Id)
    ) {
      pokemon.Name.addError('Duplicate record found. Please check the
data.');
    }
  }

}
```

A 9. Pokedex LWC

JS File

```
/* eslint-disable no-unreachable */
/* eslint-disable @lwc/lwc/no-async-operation */
/* eslint-disable radix */
import { LightningElement, wire } from "lwc";
import getPokemons from
"@salesforce/apex/PokedexController_Main.PokedexGetMain";
import { getObjectInfo } from "lightning/uiObjectInfoAPI";
import { getPicklistValues } from "lightning/uiObjectInfoAPI";
import { NavigationMixin } from "lightning/navigation";
import SAD_PIKACHU from "@salesforce/resourceUrl/SadPikachu";
import ERROR_IMG from "@salesforce/resourceUrl/ErrorPsyduck";
export default class Pokedex extends NavigationMixin(LightningElement) {
  spinner = true;
  errorImg = ERROR_IMG;
  sad_pikachu = SAD_PIKACHU;
  selectedGeneration = "all";
  typeOptions = [];
  selectedTypes = [];
  pokemons;
  error;
  searchTerm = "";
  numberOfPokemons = 0;
  generationOptions = [];
  genSet = new Set();

  connectedCallback() {
    this.loadPokemoms();
    this.setGenerationOptions();
  }
  loadPokemoms() {
    // this.spinner = false; //* to force the error for testing */
    // return (this.error = new Error("Network error test"));
    getPokemons()
      .then((result) => {
        this.pokemons = result.map((pokemon) => {
          const mappedTipo = pokemon.Tipo__c
            ? pokemon.Tipo__c.split(";").map((tipo) => ({
                name: tipo,
                colorType: `poke ${tipo}`
              }))
            : [];
```

```
          return {
            ...pokemon,
            tipo__c: mappedTipo,
            Geracao__c: pokemon.Geracao__c || 0
          };
        });
        this.pokemons.forEach((pokemon) => {
          this.genSet.add(pokemon.Geracao__c);
        });
        this.setGenerationOptions();
        this.numberOfPokemons = result.length;
        this.spinner = false;
      })
      .catch((error) => {
        this.spinner = false;
        this.error = error;
      });
  }
  //* To set the options to be displayed on the generation picklist.
  //* the sort is for the case there is an insert
  //* the list will be allways ordered asc.
  setGenerationOptions() {
    this.generationOptions = [{ label: "All", value: "all" }];
    const genSorted = Array.from(this.genSet).sort((a, b) => a - b);
    genSorted.forEach((generation) => {
      this.generationOptions.push({
        label: `Generation ${generation}`,
        value: `${generation}`
      });
    });
  }
  //* get objectInfo to get the picklist values of the types
  //* to set on the typeOptions and on the typeColor
  @wire(getObjectInfo, { objectAPIName: "Pokemon__c" })
  objectInfo;
  @wire(getPicklistValues, {
    recordTypeId: "$objectInfo.data.defaultRecordTypeId",
    fieldAPIName: "Pokemon__c.Tipo__c"
  })
  loadTypeOptions({ data, error }) {
    if (data) {
      this.typeOptions = data.values
        .map((picklistValue) => ({
          label: picklistValue.label,
          value: picklistValue.value
        }))
        .sort((a, b) => a.label.localeCompare(b.label));
```

```
    } else if (error) {
      console.error("Error retrieving picklist values:", error);
    }
  }
  //* filter the pokemons by specified criteria
  get pokemonsFilter() {
    if (
      this.selectedGeneration === "all" &&
      this.selectedTypes.length === 0 &&
      this.searchTerm === ""
    ) {
      return this.pokemons;
    }
    return this.pokemons.filter((pokemon) => {
      const generationMatch =
        this.selectedGeneration === "all" ||
        pokemon.Geracao__c === parseInt(this.selectedGeneration);
      const searchTermMatch =
        this.searchTerm === "" ||
        pokemon.Name.toLowerCase().includes(this.searchTerm.toLowerCase());
      const typeMatch =
        this.selectedTypes.length === 0 ||
        this.selectedTypes.every((type) => pokemon.Tipo__c.includes(type));
      return generationMatch && searchTermMatch && typeMatch;
    });
  }

  //*  update the number o pokemons there are shown when applied the filter
  updateNumberOfPokemons() {
    this.numberOfPokemons = Object.keys(this.pokemonsFilter).length;
  }

  handleSearchChange(event) {
    window.clearTimeout(this.delayTimeout);
    const searchTerm = event.target.value;
    this.delayTimeout = setTimeout(() => {
      this.searchTerm = searchTerm;
      this.updateNumberOfPokemons();
    }, 300);
  }
  handleGenerationChange(event) {
    this.selectedGeneration = event.target.value;
    this.updateNumberOfPokemons();
  }
  handleTypeChange(event) {
    this.selectedTypes = event.detail.map((item) => item.value);
    this.updateNumberOfPokemons();
```

```
  }
  //* handle the navigation when click on image
  handleNavigationChange(event) {
    const pokemonId = event.detail;
    this[NavigationMixin.Navigate]({
      type: "standard__recordPage",
      attributes: {
        recordId: pokemonId,
        objectAPIName: "Pokemon__c",
        actionName: "view"
      }
    });
  }
  //* check if there are pokemons to displayed so it can show an error
message
  get existsPokemons() {
    const exists = this.pokemonsFilter;
    return exists && exists.length > 0;
  }
}
```

HTML File

```
<template>
  <div>
    <div class="navbar-container">
      <div class="navbar-content">
        <div
          class="slds-size_full slds-grid slds-gutters_medium slds-
grid_vertical-align-start slds-m-bottom_x-small"
        >
          <div class="slds-col slds-size_2-of-12 slds-var-m-
horizontal_medium">
            <lightning-combobox
              label="Generation"
              value={selectedGeneration}
              placeholder="Select a Generation"
              options={generationOptions}
              onchange={handleGenerationChange}
            >
            </lightning-combobox>
          </div>

          <div class="slds-col slds-size_2-of-12 slds-var-m-
horizontal_medium">
            <c-multi-select-combobox
              label="Types"
              name="options"
              options={typeOptions}
              onchange={handleTypeChange}
              show-pills
            ></c-multi-select-combobox>
          </div>

          <div class="slds-col slds-size_4-of-12 slds-var-m-
horizontal_medium">
            <lightning-input
              type="Search"
              class="slds-var-m-bottom_small"
              label="Search"
              value={searcherm}
              onchange={handleSearchChange}
            >
            </lightning-input>
          </div>
        </div>
        <div>
```

```
          <p class="poke-font">Number of pokemons: {numberOfPokemons}</p>
        </div>
      </div>
    </div>
    <div class="slds-card__body slds-card__body_inner">
      <div if:true={spinner}>
        <lightning-spinner
          alternative-text="Loading"
          size="medium"
        ></lightning-spinner>
      </div>
      <template if:true={pokemonsFilter}>
        <lightning-layout multiple-rows="true" pull-to-boundary="small">
          <template for:each={pokemonsFilter} for:item="pokemon">
            <lightning-layout-item
              key={pokemon.Id}
              size="2"
              class="slds-var-p-around_large slds-var-m-top_xx-large slds-
large-size_2-of-12 slds-medium-size_4-of-12 slds-small-size_6-of-12"
            >
              <c-pokemon-tile
                pokemon={pokemon}
                onpokemonview={handleNavigationChange}
              ></c-pokemon-tile>
            </lightning-layout-item>
          </template>
        </lightning-layout>
        <!--* not pokemons found -->
        <template if:false={existsPokemons}>
          <div class="not_found_div">
            <div>
              <img src={sad_pikachu} class="error-img" alt="" />
              <p>Sorry, we couldn't find any Pókemon!!</p>
            </div>
          </div>
        </template>
      </template>
      <!--* Error caught-->
      <template if:true={error}>
        <div class="not_found_div">
          <img src={errorImg} class="error-img slds-var-m-bottom_small" />
          <p class="error-message error-text">{error.message}</p>
        </div>
      </template>
    </div>
  </div>
</template>
```

CSS File

```css
.navbar-container {
  position: sticky;
  top: 5.63rem;
  background-color: ivory; /* Adjust background color as needed */
  z-index: 1; /* Set z-index to ensure it stays on top of other elements
*/
  border-radius: 0 0 0.25rem 0.25rem;
  margin-top: -1.3rem;
  box-shadow: 8px 8px 15px;
}
.navbar-content {
  display: grid;
  place-items: center;
  padding: 10px;
  grid-template-rows: auto auto;
}
.not_found_div {
  display: grid;
  place-items: center;
  text-align: center;
  margin-top: 100px;
  font-size: 4vw;
}
.error-img {
  width: 25%;
  aspect-ratio: 4/3;
  object-fit: contain;
}
@font-face {
  font-family: "Pokemon Solid Font";
  src: url("/resource/PokemonFont/Pokemon-Solid.ttf") format("TrueType");
}
.poke-font {
  font-family: "Pokemon Solid Font";
  font-size: 20pt;
  color: darkred;
  text-shadow: 2px 2px black;
}
```

A 10. PokemonTile LWC

JS File

```
import { LightningElement, API } from "lwc";

export default class PokemonTile extends LightningElement {
  @API pokemon;

  get getColorByGeneration() {
    const generation = this.pokemon && this.pokemon.Geracao__c;
    return `pokemon-gen-color gen${generation}`;
  }

  handleOpenRecordOnClick() {
    const selectEvent = new CustomEvent("pokemonview", {
      detail: this.pokemon.Id
    });
    this.dispatchEvent(selectEvent);
  }
}
```

HTML File

```
<template>
  <div class={getColorByGeneration}>
    <span class="slds-var-m-around_xx-small"
      ><p class="slds-var-m-bottom_xx-small slds-text-align_center">
        <strong>Pokémon Nº </strong>{pokemon.PokeIndex__c}
      </p></span
    >
    <div class="slds-grid slds-align_absolute-center">
      <div>
        <img src={pokemon.Foto_URL__c} onclick={handleOpenRecordOnClick}
/>
      </div>
    </div>
    <div class="slds-p-left_small">
      <p class="slds-var-m-bottom_xx-small">
        <strong>Name: </strong>{pokemon.Name}
      </p>
      <p class="slds-var-m-bottom_xx-small">
        <strong>Generation: </strong>{pokemon.Geracao__c}
      </p>
      <p class="slds-var-m-bottom_xx-small">
        <strong>Height: </strong>{pokemon.Altura__c}
      </p>
      <p class="slds-var-m-bottom_xx-small">
        <strong>Weight: </strong>{pokemon.Peso__c}
      </p>
      <p class="slds-var-m-bottom_xx-small slds-text-align_center">
        <strong>Type: </strong>
      </p>
      <template if:true={pokemon.Tipo__c}>
        <template for:each={pokemon.tipo__c} for:item="tipo">
          <div key={tipo.name} class="types_parent_div">
            <div class={tipo.colorType}>
              <p class="type_text">{tipo.name}</p>
            </div>
          </div>
        </template>
      </template>
    </div>
  </div>
</template>
```

CSS File

```css
p {
  font-family: "Comic Sans MS", "Comic Sans", cursive;
  font-size: small;
}

img {
  height: fit-content;
  width: fit-content;
}

.types_parent_div {
  display: grid;
  place-items: center;
}
/** classes for types colors **/
.poke {
  --type-color: #a1a177;
  background-color: var(--type-color);
  border-radius: 0.75rem;
  margin-bottom: 5px;
  width: 50%;
  text-align: center;
}

.poke.Fire {
  --type-color: #ee8130;
}

.poke.Water {
  --type-color: #6390f0;
}

.poke.Electric {
  --type-color: #f7d02c;
}

.poke.Normal {
  --type-color: #a1a177;
}

.poke.Grass {
  --type-color: #7ac74c;
}

.poke.Ice {
  --type-color: #96d9d6;
}
```

```
.poke.Fighting {
  --type-color: #c22e28;
}

.poke.Poison {
  --type-color: #a33ea1;
}

.poke.Ground {
  --type-color: #e2bf65;
}

.poke.Flying {
  --type-color: #a98ff3;
}

.poke.Psychic {
  --type-color: #f95587;
}

.poke.Bug {
  --type-color: #a6b91a;
}

.poke.Rock {
  --type-color: #b6a136;
}

.poke.Ghost {
  --type-color: #735797;
}

.poke.Dragon {
  --type-color: #6f35fc;
}

.poke.Dark {
  background-color: #705746;
}

.poke.Steel {
  --type-color: #b7b7ce;
}

.poke.Fairy {
  --type-color: #d685ad;
```

```
}

/** Classes for cards colors **/

.pokemon-gen-color {
  --gen-color: white;
  background: var(--gen-color);
  padding: 0.125rem;
  display: block;
  height: 340px;
  width: 200px;
  border-radius: 0.35rem;
  box-shadow: 5px 5px;
  transition: transform 0.5s ease;
  transition: box-shadow 0.01s ease;
}
.pokemon-gen-color.gen1 {
  --gen-color: linear-gradient(
    225deg,
    rgba(236, 222, 64, 0.8) 1%,
    rgba(113, 93, 27, 1) 57%,
    rgba(203, 203, 192, 0.8) 100%
  );
}

.pokemon-gen-color.gen2 {
  --gen-color: linear-gradient(
    225deg,
    rgba(92, 92, 2, 1) 1%,
    rgba(189, 126, 41, 0.8827906162464986) 57%,
    rgba(124, 71, 0, 1) 100%
  );
}
.pokemon-gen-color.gen3 {
  --gen-color: linear-gradient(
    225deg,
    rgba(203, 203, 192, 1) 1%,
    rgba(27, 78, 113, 1) 57%,
    rgba(64, 221, 236, 1) 100%
  );
}
.pokemon-gen-color.gen4 {
  --gen-color: linear-gradient(
    225deg,
    rgba(255, 255, 165, 1) 1%,
    rgba(117, 0, 0, 0.8827906162464986) 57%,
    rgba(242, 102, 102, 0.4009978991596639) 100%
```

```
  );
}
.pokemon-gen-color.gen5 {
  --gen-color: linear-gradient(
    225deg,
    rgba(204, 204, 204, 1) 0%,
    rgba(0, 107, 39, 1) 57%,
    rgba(83, 102, 0, 0.4009978991596639) 100%
  );
}
.pokemon-gen-color.gen6 {
  --gen-color: linear-gradient(
    225deg,
    rgba(204, 204, 204, 1) 0%,
    rgba(73, 0, 110, 0.8995973389355743) 57%,
    rgba(240, 0, 247, 1) 100%
  );
}
.pokemon-gen-color.gen7 {
  --gen-color: linear-gradient(
    225deg,
    rgba(1, 63, 91, 0.7455357142857143) 0%,
    rgba(219, 236, 235, 1) 57%,
    rgba(1, 63, 91, 1) 100%
  );
}
.pokemon-gen-color.gen8 {
  --gen-color: linear-gradient(
    225deg,
    rgba(0, 0, 0, 0.7455357142857143) 0%,
    rgba(219, 236, 235, 1) 57%,
    rgba(0, 0, 0, 1) 100%
  );
}

.pokemon-gen-color:hover {
  transform: scale(1.05);
  box-shadow: 8px 8px 3px;
}
img:hover {
  cursor: pointer;
}

.type_text {
  font-weight: bold;
  opacity: 1;
}
```

A 11. MultiSelectCombobox LWC

JS File

```
import { LightningElement, API, track } from "lwc";

export default class MultiSelectCombobox extends LightningElement {
  @API disabled = false;
  @API label = "";
  @API name;
  @API options = [];
  @API placeholder = "Select only 1 or 2 Types";
  @API readOnly = false;
  @API required = false;
  @API singleSelect = false;
  @API showPills = false;

  @track currentOptions = [];
  selectedItems = [];
  selectedOptions = [];
  isInitialized = false;
  isLoaded = false;
  isVisible = false;
  isDisabled = false;

  connectedCallback() {
    this.isDisabled = this.disabled || this.readOnly;
    this.hasPillsEnabled = this.showPills && !this.singleSelect;
  }

  renderedCallback() {
    if (!this.isInitialized) {
      this.template
        .querySelector(".multi-select-combobox__input")
        .addEventListener("click", (event) => {
          this.handleClick(event.target);
          event.stopPropagation();
        });
      this.template.addEventListener("click", (event) => {
        event.stopPropagation();
      });
      document.addEventListener("click", () => {
        this.close();
      });
      this.isInitialized = true;
```

```
      this.setSelection();
    }
  }

  handlechange(event) {
    this.change(event);
  }

  handleRemove(event) {
    this.selectedOptions.splice(event.detail.index, 1);
    this.change(event);
  }

  handleClick() {
    if (this.isLoaded === false) {
      this.currentOptions = JSON.parse(JSON.stringify(this.options));
      this.isLoaded = true;
    }

    if (this.template.querySelector(".slds-is-open")) {
      this.close();
    } else {
      this.template
        .querySelectorAll(".multi-select-combobox__dropdown")
        .forEach((node) => {
          node.classList.add("slds-is-open");
        });
    }
  }

  change(event) {
    if (this.singleSelect) {
      this.currentOptions.forEach((item) => (item.selected = false));
    }

    this.currentOptions
      .filter((item) => item.value === event.detail.item.value)
      .forEach((item) => (item.selected = event.detail.selected));
    this.setSelection();
    const selection = this.getSelectedItems();
    this.dispatchEvent(
      new CustomEvent("change", {
        detail: this.singleSelect ? selection[0] : selection
      })
    );

    if (this.singleSelect) {
```

```
      this.close();
    }
  }

  close() {
    this.template
      .querySelectorAll(".multi-select-combobox__dropdown")
      .forEach((node) => {
        node.classList.remove("slds-is-open");
      });
    this.dispatchEvent(new CustomEvent("close"));
  }
  //* this is for handling the pills and the input visualization value
  setSelection() {
    const selectedItems = this.getSelectedItems();
    let selection = "";
    if (selectedItems.length < 1) {
      selection = this.placeholder;
      this.selectedOptions = [];
    } else {
      selection = selectedItems.map((selected) => selected.label).join(", ");
      this.selectedOptions = this.getSelectedItems();
    }

    this.selectedItems = selection;
    this.isVisible = this.selectedOptions && this.selectedOptions.length > 0;
  }

  getSelectedItems() {
    return this.currentOptions.filter((item) => item.selected);
  }
}
```

HTML File

```
<template>
  <div class="slds-form-element">
    <label if:true={label} class="slds-form-element__label">
      <abbr if:true={required} title="required" class="slds-
required">*</abbr>
      {label}
    </label>
    <div class="slds-form-element__control">
      <div class="slds-combobox_container">
        <div
          class="slds-combobox slds-dropdown-trigger slds-dropdown-
trigger_click slds-picklist multi-select-combobox__dropdown"
        >
          <div
            class="slds-combobox__form-element slds-input-has-icon slds-
input-has-icon_right"
            role="none"
          >
            <input
              type="text"
              class="slds-input multi-select-combobox__input"
              role="textbox"
              aria-controls="multi-pick-list-dropdown-items"
              value={selectedItems}
              required={required}
              disabled={isDisabled}
              readonly
            />
            <span
              class="slds-icon_container slds-icon-utility-down slds-
input__icon slds-input__icon_right multi-select-combobox__icon"
              title="Click to Open the Dropdown"
            >
              <lightning-icon
                icon-name="utility:down"
                alternative-text="Click Here"
                size="xx-small"
                class="slds-icon slds-var-p-right_medium slds-icon--selected
slds-icon--x-small slds-icon-text-default"
              ></lightning-icon>
            </span>
          </div>

          <div
```

```
            class="slds-dropdown slds-dropdown_length-7 slds-dropdown_fluid
multi-select-combobox__listbox"
            role="listbox"
          >
            <ul class="slds-listbox slds-listbox_vertical"
role="presentation">
              <template for:each={currentOptions} for:item="item">
                <c-multi-select-combobox-item
                  key={item.value}
                  item={item}
                  onchange={handlechange}
                >
                </c-multi-select-combobox-item>
              </template>
            </ul>
          </div>
        </div>
      </div>
    </div>
    <div if:true={hasPillsEnabled}>
      <lightning-pill-container
        if:true={isVisible}
        items={selectedOptions}
        variant="bare"
        onitemremove={handleRemove}
      ></lightning-pill-container>
    </div>
  </div>
</template>
```

CSS File

```
.multi-select-combobox__dropdown {
  max-height: 500px;
  overflow-y: auto;
}

.multi-select-combobox__input {
  background-color: #ffffff;
  border-radius: 0.35rem;
  width: 100%;
  transition: border 0.1s linear, background-color 0.1s linear;
  display: inline-block;
  padding: 0 1rem 0 0.75rem;
  line-height: 1.875rem;
  min-height: calc(1.875rem + (1px * 2));
}

.multi-select-combobox__listbox {
  width: 100%;
}
```

A 12. MultiSelectComboboxItem

JS File

```
import { LightningElement, API } from "lwc";

export default class MultiSelectComboboxItem extends LightningElement {
  @API item;
  get itemClass() {
    return `slds-listbox_item ${this.item.selected ? "slds-is-selected" :
""}`;
  }

  handleClick() {
    const evt = new CustomEvent("change", {
      detail: { item: this.item, selected: !this.item.selected }
    });
    this.dispatchEvent(evt);
  }
}
```

HTML File

```
<template>
  <li
    role="presentation"
    key={item.key}
    class={itemClass}
    dat-id={item.key}
    data-name={item.value}
    onclick={handleClick}
  >
    <div
      class="slds-media slds-listbox__option slds-listbox__option_plain slds-
media_small"
      role="option"
    >
      <span>
        <lightning-icon
          icon-name="utility:check"
          alternative-text="Selected"
          size="x-small"
          class="slds-icon slds-icon--selected slds-icon--x-small slds-icon-
text-default slds-m-right--x-small"
        ></lightning-icon>
      </span>
      <span class="slds-media__body">
        <span class="slds-truncate" title={item.value}>{item.label}</span>
      </span>
    </div>
  </li>
</template>
```

A 13. PokeStats LWC

JS File

```
import { LightningElement, API, wire, track } from "lwc";
import { getRecord } from "lightning/uiRecordAPI";
import getStats from "@salesforce/apex/PokemonStatsAPICall.getStats";
import STATS_BG from "@salesforce/resourceUrl/statsBG";
export default class PokemonStats extends LightningElement {
  @API recordId;
  @track pokemon;
  @track pokemonName;
  error;
  @track imgUrl;
  @track pokeMap = [];
  stats_BG = STATS_BG;
  @wire(getRecord, {
    recordId: "$recordId",
    fields: ["Pokemon__c.Name", "Pokemon__c.Foto_URL__c"]
  })
  wiredPokemon({ data, error }) {
    if (data) {
      this.pokemonName = data.fields.Name.value.toLowerCase();
      this.imgUrl = data.fields.Foto_URL__c.value;
      this.error = undefined;
    } else if (error) {
      this.data = undefined;
      this.error = error;
    }
  }

  @wire(getStats, { pokemon: "$pokemonName" })
  wiredStats(result) {
    if (result.data) {
      let val = result.data;

      for (let key in val) {
        this.pokeMap.push({ value: val[key], key: key });
      }
      console.log("TIPO-->", typeof pokeMap);
    }
  }

  get exists() {
    return this.pokeMap.length > 0;
```

```
  }

  get getStatsBG() {
    return `background-image:url("${this.stats_BG}")`;
  }
}
```

HTML File

```
<template>
  <template if:true={exists}>
    <div class="slds-container_medium stats_bg" style={getStatsBG}>
      <div class="slds-var-p-vertical_large">
        <div
          class="slds-grid slds-align_absolute-center slds-var-m-
bottom_small"
        >
          <img src={imgUrl} class="stat_img" alt="" />
        </div>
        <div class="slds-grid slds-align_absolute-center">
          <template for:each={pokeMap} for:item="mapKey">
            <div
              key={mapKey.key}
              class="slds-var-m-horizontal_x-small slds-text-
align_center"
            >
              <p class="stat_text slds-var-m-bottom_x-
small">{mapKey.key}</p>
              <p class="stat_values">{mapKey.value}</p>
            </div>
          </template>
        </div>
      </div>
    </div>
  </template>
  <template if:false={exists}>
    <div
      class="slds-grid slds-align_absolute-center slds-grid_vertical-
align-center"
    >
      <div
        class="no-data-found slds-grid slds-align_absolute-center slds-
grid_vertical-align-center"
      >
        <p>No Data Found</p>
      </div>
    </div>
  </template>
</template>
```

CSS File

```css
.stat_img {
  width: 50%;
  aspect-ratio: 16/9;
  object-fit: contain;
}

.stat_text {
  font-family: sans-serif;
  font-weight: bolder;
  font-size: 10pt;
  color: black;
  text-decoration: underline;
  font-style: italic;
}

.stat_values {
  font-family: Verdana, Geneva, Tahoma, sans-serif;
}

.stats_bg {
  background-size: cover;
}
.no-data-found {
  width: 250px;
  height: 250px;
  font-family: "Trebuchet MS", "Lucida Sans Unicode", "Lucida Grande",
    "Lucida Sans", Arial, sans-serif;
  font-size: x-large;
}
```