**Automatização da Extração de Comentários e Informações Gerais do Booking.com**

**Autor:** Diogo Amorim

**Índice**

1. **Introdução**  
   1.1. Web Scraping – Conceitos Gerais  
   1.2. Selenium – Automação de Navegadores Web  
   1.3. BeautifulSoup – Extração e Parsing de HTML
2. **Estrutura do Projeto "booking\_scraper"**  
   2.1. Ficheiro hoteis.csv – Lista de Hotéis e URLs  
   2.2. Script scraper\_hotel\_info.py – Extração de Informações Gerais  
   2.3. Script scraper\_reviews\_pagination\_fixed\_v4.py – Extração de Comentários  
   2.4. Dados Extraídos e Estrutura dos Outputs (Excel)
3. **Desempenho e Tempo de Execução**
4. **Desafios e Soluções Implementadas**  
   4.1. Cookies e Consentimento  
   4.2. Carregamento Dinâmico e Paginação  
   4.3. Retomar Execução e Tolerância a Interrupções
5. **Conclusão e Reflexão Crítica**

**1. Introdução**

Este relatório técnico descreve, em detalhe, o projeto de scraping do website Booking.com desenvolvido por Diogo Amorim, intitulado **“Automatização da Extração de Comentários e Informações Gerais do Booking.com”**. O projeto insere-se no âmbito do curso de Engenharia Informática e tem como objetivo extrair automaticamente **comentários de clientes (reviews)** e **informações gerais de hotéis** a partir da plataforma Booking.com, utilizando técnicas de *Web Scraping*. No presente documento, é apresentada a fundamentação teórica relevante (ferramentas e técnicas utilizadas) e uma descrição pormenorizada da estrutura e funcionamento do projeto, incluindo **ficheiros, scripts, outputs gerados, desempenho obtido, dificuldades encontradas** e as **soluções implementadas**. Por fim, é feita uma reflexão crítica sobre o processo de raspagem de grandes volumes de dados e discutem-se possíveis melhorias futuras.

**1.1. Web Scraping – Conceitos Gerais**

**Web Scraping**, ou raspagem de dados web, refere-se ao processo automatizado de extração de informações de sites na Internet. Em termos simples, um programa de web scraping simula a navegação humana: acede a páginas web, **recupera o seu conteúdo HTML e extrai dados de interesse**, convertendo-os em informação estruturada para posterior análise​[pt.wikipedia.org](https://pt.wikipedia.org/wiki/Web_scraping#:~:text=Web%20scraping%20,processamento%20de%20texto%2C%20compreens%C3%A3o%20sem%C3%A2ntica). Conforme a definição da Wikipédia, “*Web scraping (em português, ‘raspagem da web’) é uma forma de mineração que permite a extração de dados de sites da web, convertendo-os em informação estruturada para posterior análise*”​[pt.wikipedia.org](https://pt.wikipedia.org/wiki/Web_scraping#:~:text=Web%20scraping%20,processamento%20de%20texto%2C%20compreens%C3%A3o%20sem%C3%A2ntica). Em vez de copiar e colar manualmente conteúdos (o que seria moroso e pouco prático), um *scraper* (raspador) automatiza esta tarefa, percorrendo páginas e recolhendo dados de forma eficiente.

As aplicações de web scraping são inúmeras: comparação de preços e produtos, monitorização de notícias ou redes sociais, coleta de dados para pesquisas de mercado, agregação de ofertas de emprego, e assim por diante​[pt.wikipedia.org](https://pt.wikipedia.org/wiki/Web_scraping#:~:text=intelig%C3%AAncia%20artificial%20%20e%20,alguns%20casos%2C%20roubo%20de%20informa%C3%A7%C3%A3o). No caso deste projeto, o objetivo específico foi recolher **dados públicos do Booking.com**, nomeadamente a lista de **comentários de hóspedes** e informações gerais sobre determinadas propriedades (hotéis).

É importante mencionar que embora o web scraping seja uma ferramenta poderosa, a sua utilização deve respeitar os **termos de uso** dos sites e as questões de **ética e legalidade**. Muitos sites, incluindo o Booking.com, colocam restrições ao scraping massivo de dados, podendo bloquear acessos suspeitos ou requerer que se respeitem os recursos do servidor. Por isso, um projeto desta natureza deve ser desenvolvido com cuidado, evitando sobrecarregar o site e garantindo que apenas se extraem dados permitidos (dados públicos, não pessoais sensíveis).

**1.2. Selenium – Automação de Navegadores Web**

Para efetuar a raspagem de dados de um site dinâmico como o Booking.com, muitas vezes ferramentas tradicionais baseadas apenas em requisições HTTP não são suficientes. Isso porque plataformas modernas carregam conteúdo de forma dinâmica (via JavaScript), apresentam *pop-ups* de consentimento de cookies, requerem interações de utilizador para revelar certas informações (por exemplo, clicar em botões “Mostrar mais comentários”), etc. Nesses cenários, é útil recorrer ao **Selenium**, uma ferramenta de automação de navegadores web.

O **Selenium WebDriver** permite controlar um navegador (Chrome, Firefox, etc.) através de scripts, simulando cliques, rolagem de página e outras interações exatamente como um utilizador real faria. Originalmente concebido para testes automatizados de aplicações web​[pt.wikipedia.org](https://pt.wikipedia.org/wiki/Selenium_(software)#:~:text=Selenium%20%C3%A9%20um%20framework%20,96%20lan%C3%A7ado%20sob%20a), o Selenium provou ser igualmente útil para **web scraping** quando os sites apresentam obstáculos à extração simples. De acordo com a literatura, o Selenium é frequentemente utilizado em *softwares* de extração de informações de sites “onde existem muitas complicações para interagir utilizando o protocolo HTTP”​[pt.wikipedia.org](https://pt.wikipedia.org/wiki/Selenium_(software)#:~:text=A%20ind%C3%BAstria%20de%20automa%C3%A7%C3%A3o%20por,como%20por%20exemplo%20Scrapy) – ou seja, quando frameworks de scraping tradicionais (como Scrapy, que operam diretamente em HTML estático) não conseguem contornar funcionalidades dinâmicas ou barreiras como autenticações e consentimentos interativos.

No contexto deste projeto, o Selenium foi uma escolha natural porque o Booking.com implementa conteúdo dinâmico e paginação via botões, tornando a **simulação de um utilizador real** necessária para extrair todos os dados. Com o Selenium, o script consegue: abrir cada página de hotel, aceitar automaticamente os cookies, clicar em botões de “Próxima página” dos comentários, e extrair o código HTML resultante já com todos os comentários carregados.

Para concretizar esta automação, utiliza-se o **Chrome WebDriver** (controlador do Google Chrome). O projeto faz uso da biblioteca webdriver\_manager para gerir a versão adequada do ChromeDriver de forma automática, facilitando a execução em diferentes ambientes sem necessidade de configuração manual do executável do driver. Em suma, o Selenium fornece o "browser automatizado" para navegar no Booking.com, e combinado com outras ferramentas descritas adiante, forma a espinha dorsal do scraper.

**1.3. BeautifulSoup – Extração e Parsing de HTML**

Depois de carregar uma página web através do Selenium, é necessário **extrair os elementos desejados do código HTML**. Embora o Selenium permita também localizar elementos diretamente via código (por seletores CSS, XPath, etc.), neste projeto optou-se por usar o **BeautifulSoup** (biblioteca bs4 do Python) para fazer o *parsing* do HTML obtido e extrair de forma mais conveniente os campos de interesse.

O **BeautifulSoup** é uma biblioteca Python poderosa para navegar, pesquisar e modificar a árvore de análise (*parse tree*) de documentos HTML ou XML​[brightdata.com.br](https://brightdata.com.br/blog/procedimentos/aprenda-a-usar-beautiful-soup-para-raspar-a-web-com-python#:~:text=Beautiful%20Soup%20fornece%20m%C3%A9todos%20simples,terceiros%20como%20lxml%20ou%20hml5lib). Em outras palavras, transforma o HTML bruto num objeto estruturado, onde podemos facilmente buscar tags por nome, classes CSS, identificadores, atributos específicos, texto interno, etc. Segundo a documentação, *“Beautiful Soup fornece métodos simples de navegação, pesquisa e modificação de uma árvore de parse em ficheiros HTML/XML. Transforma um documento HTML complexo numa árvore de objetos Python […]. Esta ferramenta não só ajuda a raspar, mas também a limpar os dados.”*​[brightdata.com.br](https://brightdata.com.br/blog/procedimentos/aprenda-a-usar-beautiful-soup-para-raspar-a-web-com-python#:~:text=Beautiful%20Soup%20fornece%20m%C3%A9todos%20simples,terceiros%20como%20lxml%20ou%20hml5lib).

No projeto “booking\_scraper”, o BeautifulSoup é utilizado principalmente no script de extração de comentários para percorrer o HTML das reviews carregadas e **selecionar os elementos individuais de cada comentário** (nome do utilizador, nacionalidade, pontuação, título do comentário, texto positivo, texto negativo, etc.). A combinação Selenium + BeautifulSoup revela-se bastante eficaz: o Selenium garante que todo o conteúdo dinâmico esteja presente no HTML (renderizado pelo navegador), e o BeautifulSoup facilita a extração estruturada desses dados.

Além dessas ferramentas principais, o projeto também utiliza bibliotecas auxiliares como **Pandas** (para armazenar os dados em estruturas tabulares e exportar para ficheiros Excel), **Time** e **WebDriverWait** (para controlar esperas e temporizações necessárias durante o scraping), e **JSON** (para ler/escrever o estado de progresso da execução, como veremos adiante). A próxima secção descreve detalhadamente a estrutura do projeto e o papel de cada componente.

**2. Estrutura do Projeto "booking\_scraper"**

O projeto está organizado em vários ficheiros e scripts, com responsabilidades bem definidas. Os principais componentes incluem um ficheiro de configuração de entrada (hoteis.csv) e dois scripts Python: um dedicado a extrair **informações gerais de cada hotel** (scraper\_hotel\_info.py) e outro focado em extrair **comentários dos hóspedes** (scraper\_reviews\_pagination\_fixed\_v4.py). Existem também ficheiros de output (resultados) gerados em formato Excel e um ficheiro JSON de progresso. Nesta secção, é descrito cada um desses componentes e como eles interagem no fluxo geral.

**2.1. Ficheiro**hoteis.csv**– Lista de Hotéis e URLs**

O ficheiro hoteis.csv contém a lista das unidades hoteleiras alvo de scraping, bem como os endereços necessários para aceder às respetivas páginas no Booking.com. Este CSV atua como a **fonte de dados de entrada** para os scripts, permitindo definir quais hotéis serão processados sem alterar o código. Cada linha do hoteis.csv corresponde a um hotel e tipicamente inclui as seguintes colunas:

* **nome** – O nome do hotel ou alojamento tal como reconhecido (e.g., *"Hotel ABC Resort"*).
* **link** – A URL da página principal do hotel no Booking.com (normalmente terminando em .html). Por exemplo: https://www.booking.com/hotel/pt/nome-do-hotel.html.
* **link\_reviews** – A URL direta para a secção de comentários/reviews desse hotel. No caso do Booking.com, esta URL pode ser obtida acrescentando #tab-reviews ao link principal, ou noutros casos utilizando parâmetros próprios do site para filtrar a página de comentários. Por exemplo: https://www.booking.com/hotel/pt/nome-do-hotel.html#tab-reviews.

Um exemplo simplificado de duas entradas no hoteis.csv poderia ser:

csv

CopiarEditar

nome,link,link\_reviews Hotel Exemplo 1,https://www.booking.com/hotel/pt/exemplo1.html,https://www.booking.com/hotel/pt/exemplo1.html#tab-reviews Hotel Exemplo 2,https://www.booking.com/hotel/pt/exemplo2.html,https://www.booking.com/hotel/pt/exemplo2.html#tab-reviews

Neste projeto, o hoteis.csv foi preparado previamente pelo autor com os hotéis pretendidos. Em particular, foram alvo da extração **124 hotéis**, que somavam em conjunto mais de 120.000 comentários de clientes. É importante que o utilizador/autor forneça corretamente este ficheiro, pois dele depende o alcance da raspagem. Vale notar que, caso o Booking.com altere a forma de acesso aos comentários (por exemplo, mudando o seletor #tab-reviews), seria necessário atualizar esta coluna para garantir que a página de reviews é aberta corretamente.

**2.2. Script**scraper\_hotel\_info.py**– Extração de Informações Gerais**

O script scraper\_hotel\_info.py é responsável por percorrer a lista de hotéis e extrair de cada um **informações gerais e descritivas** sobre o alojamento. Estas informações incluem, por exemplo, a pontuação média do hotel, uma descrição da propriedade, a morada/localização, destaques (facilidades ou características principais), comodidades populares, categorias avaliadas nas reviews, entre outros detalhes disponíveis na página principal do hotel.

**Funcionalidade e Fluxo de Execução:** Ao executar este script, ele realiza os seguintes passos para cada hotel listado no hoteis.csv:

1. **Inicialização do Navegador:** Utiliza-se o Selenium WebDriver para abrir uma janela do Google Chrome. As opções configuradas incluem iniciar o navegador maximizado (--start-maximized) para garantir que todos os elementos estejam visíveis na tela sem necessidade de scroll adicional. O script aproveita o ChromeDriverManager para obter automaticamente o executável do ChromeDriver compatível.
2. **Leitura do CSV:** O ficheiro hoteis.csv é carregado em memória usando Pandas (função pd.read\_csv), permitindo iterar pelas linhas facilmente. Para cada linha (hotel) extrai-se o nome (nome) e o link (link) da página principal.
3. **Navegação para a página do hotel:** O Selenium dirige o Chrome para o URL do hotel (driver.get(link)), carregando assim a página do Booking.com referente àquele alojamento. O script então aguarda alguns segundos (tipicamente time.sleep(4)) para garantir que a página e seu conteúdo dinâmico (incluindo possíveis carregamentos via JavaScript) sejam totalmente renderizados antes de tentar extrair qualquer elemento.
4. **Extração dos elementos de informação:** Utilizando seletores CSS adequados, o script tenta encontrar e recolher:
   * **Rating (Pontuação média do alojamento):** Localiza o elemento identificado por data-testid="review-score-component" e recolhe o texto, que normalmente inclui a pontuação numérica e possivelmente um resumo textual (ex.: “8,6 – Fabuloso”).
   * **Descrição:** Localiza o elemento data-testid="property-description" que contém a descrição geral do hotel (texto que o hotel fornece, indicando instalações, localização, etc.).
   * **Morada:** Procura um elemento com classe CSS específica (por exemplo, .a53cbfa6de.f17adf7576, correspondente ao seletor interno do Booking.com para o endereço).
   * **Destaques da Propriedade:** Procura elementos com data-testid="property-highlights", que tipicamente listam pontos de destaque do alojamento (ex.: “**Localização excelente**”, “**Pequeno-almoço muito bom**”, “**Estacionamento gratuito**”).
   * **Principais Comodidades:** Procura a secção de comodidades populares através de data-testid="property-most-popular-facilities-wrapper", obtendo por exemplo se há Wi-Fi gratuito, piscina, spa, etc., geralmente listados nesta área.
   * **Informações adicionais do alojamento:** Aqui o script adota uma lógica mais elaborada para capturar informações possivelmente espalhadas pela página (como políticas do alojamento, horários de check-in/check-out, detalhes de estacionamento, etc.). Ele recolhe todos os textos de elementos designados por data-testid="facility-group-container" e data-testid="property-section--content", que correspondem a grupos de informação adicionais na página. Depois, concatena esses blocos de texto, separando-os com linhas vazias para melhorar a legibilidade. O resultado é uma cadeia textual que agrega várias informações do alojamento.
   * **Categorias Avaliadas:** Finalmente, o script recolhe os elementos data-testid="review-subscore", que correspondem às sub-pontuações das reviews (por exemplo, a pontuação que os hóspedes atribuem a categorias como *Funcionários*, *Limpeza*, *Conforto*, *Localização*, *Comodidades*, *Relação qualidade/preço*, *Wi-Fi*, etc.). Essas categorias de avaliação aparecem normalmente numa secção da página principal mostrando médias de classificações em cada critério. O código une todas as categorias e suas pontuações numa única string separada por “; ” e depois faz pequenos ajustes de formatação (substitui delimitadores duplicados, insere quebras de linha entre categorias). O resultado final é algo como:

python-repl

CopiarEditar

Funcionários 9,5

Comodidades 8,5

Limpeza 9,0

Conforto 9,0

... (etc)

1. **Armazenamento dos dados:** Os valores extraídos acima são guardados num dicionário Python relativo a esse hotel, com campos como “Nome”, “Link”, “Rating Alojamento”, “Descrição”, “Morada”, “Destaques da Propriedade”, “Principais Comodidades”, “Informações do Alojamento” e “Categorias Avaliadas”. Este dicionário é adicionado a uma lista dados que acumula as informações de todos os hotéis processados. Caso ocorra alguma exceção durante a extração de um determinado hotel (por exemplo, se um seletor não for encontrado devido a diferenças na página), o script apanha o erro e imprime uma mensagem de aviso indicando o hotel em que falhou, continuando depois com o próximo. Dessa forma, o processamento não é abortado completamente por causa de um único hotel.
2. **Fecho do navegador e output:** Após iterar por todos os hotéis, o driver do Chrome é encerrado (driver.quit()), libertando recursos. Em seguida, a lista de dicionários dados é convertida num DataFrame Pandas e exportada para um ficheiro Excel chamado hotel\_info.xlsx dentro da pasta de output do projeto. Cada linha deste Excel corresponde a um hotel e as colunas são exatamente os campos mencionados (Nome, Link, Rating, Descrição, etc.). Ao final, é indicada no terminal a conclusão com sucesso e a criação do ficheiro “hotel\_info.xlsx”.

**Estrutura do Output (**hotel\_info.xlsx**):** O Excel gerado pelo scraper\_hotel\_info.py contém então uma tabela com uma linha por hotel. Para ilustrar, segue um exemplo de colunas e valores obtidos para um hotel fictício (valores meramente ilustrativos):

* Nome: **Hotel Exemplo 1**
* Link: *https://www.booking.com/hotel/pt/exemplo1.html*
* Rating Alojamento: *8,6 – Fabuloso (com base em 1200 comentários)*
* Descricao: *"Localizado no centro da cidade, a 500 metros da catedral, o Hotel Exemplo 1 oferece quartos modernos com ar condicionado, Wi-Fi gratuito e vista panorâmica."*
* Morada: *"Rua Exemplo, 123, 4000-000 Cidade, País"*
* Destaques da Propriedade: *"Localização excelente"; "Pequeno-almoço muito bom"; "Estacionamento privado"*
* Principais Comodidades: *Wi-Fi gratuito; Estacionamento; Quartos familiares; Restaurante; Bar; etc.*
* Informações do Alojamento: *"Check-in a partir das 15:00\nCheck-out até às 12:00\nEstacionamento: Disponível um parque privado grátis no local...\nPolítica de cancelamento: ..."* (vários blocos de informação concatenados)
* Categorias Avaliadas: *"Funcionários 9,5\nLimpeza 9,0\nConforto 8,5\nLocalização 9,7\nComodidades 8,0\nRelação Qualidade/Preço 8,2"*

Esses dados fornecem, portanto, um panorama completo de cada hotel, complementando a extração dos comentários que será descrita a seguir. Em particular, a presença do **rating médio** e das **categorias avaliadas** permite contextualizar os comentários (por exemplo, perceber em que aspetos o hotel mais se destaca ou tem melhorias a fazer, segundo a opinião agregada dos hóspedes).

**2.3. Script**scraper\_reviews\_pagination\_fixed\_v4.py**– Extração de Comentários**

O script scraper\_reviews\_pagination\_fixed\_v4.py é o componente central do projeto no que toca à recolha dos **comentários (reviews) deixados pelos hóspedes** de cada hotel. Este script lê também a lista de hotéis a partir do hoteis.csv, porém utiliza a coluna link\_reviews para aceder diretamente à secção de comentários de cada hotel. O sufixo “\_pagination\_fixed\_v4” sugere que é uma versão aprimorada, com correções voltadas à paginação correta e robustez do processo. A sua funcionalidade pode ser entendida em várias etapas:

**Inicialização e Configuração:**

* Tal como no script anterior, inicia-se uma instância do Chrome via Selenium WebDriver, com a janela maximizada.
* Carrega-se o hoteis.csv para obter a lista de hotéis (num DataFrame Pandas).
* Define-se o diretório de saída (output/) e o nome do ficheiro Excel temporário (comentarios\_final\_temp.xlsx) onde os comentários recolhidos serão armazenados progressivamente. Também se define o nome do ficheiro de progresso progress.json que guardará o índice do último hotel processado (para retoma em caso de falha ou interrupção).

**Retoma de Progresso (se aplicável):**

* O script verifica se existe um ficheiro progress.json previamente gerado. Em caso afirmativo, lê o conteúdo JSON para determinar a partir de que hotel deve retomar o scraping. Essa informação é armazenada na variável hotel\_inicio. Se nenhum progresso anterior existir, começa naturalmente do primeiro hotel (índice 0).
* De igual modo, se existir um ficheiro comentarios\_final\_temp.xlsx de uma execução anterior incompleta, esse ficheiro é carregado e os comentários já recolhidos são inseridos na lista comentarios\_total no início. Assim evita-se duplicar comentários já guardados e assegura-se que a execução possa continuar de onde parou.

**Loop pelos Hotéis:**

* O script entra então num loop que itera sobre os hotéis a partir do índice hotel\_inicio até ao fim da lista. Para cada hotel, extrai-se o nome (para fins de referência e registro no output) e o URL da página de comentários (link\_reviews).
* É impresso no terminal qual hotel está a ser processado, indicando o índice atual e o total (ex.: “🏨 5/124 – Hotel Exemplo 5”), para acompanhamento do utilizador.

**Carregamento da página de comentários:**

* Utilizando o Selenium, o script abre o URL de comentários do hotel (driver.get(url)). É feita uma espera fixa de ~4 segundos para permitir o carregamento inicial.
* **Aceitação de Cookies:** Imediatamente após, o script tenta encontrar e clicar no botão de aceite de cookies, caso este apareça. O seletor utilizado é o ID onetrust-accept-btn-handler, que corresponde ao botão “Aceitar” do banner de cookies frequentemente presente em páginas do Booking.com. A operação é envolvida em uma espera explícita de até 3 segundos (WebDriverWait) para garantir que o botão esteja clicável. Se o botão existir e for clicado, elimina-se essa barreira inicial; se não aparecer (por exemplo, se já tiver sido aceito anteriormente ou não for necessário), o script prossegue imediatamente, ignorando qualquer exceção resultante dessa tentativa.

**Scraping Paginado dos Comentários:**

* Após a página estar pronta e sem pop-ups, inicia-se a extração dos comentários. Define-se um contador de página (pagina = 1) e entra-se num laço while True que irá navegar sequencialmente pelas páginas de comentários daquele hotel.
* Para cada **página de comentários**:
  1. Imprime-se no terminal uma mensagem indicando a página atual que está a ser processada (ex.: “📄 A processar página 1...”).
  2. Espera-se brevemente (~2 segundos) e obtém-se o código-fonte HTML completo da página atual através do Selenium (driver.page\_source). Esse HTML é então entregue ao BeautifulSoup para criar um objeto de parsing (soup = BeautifulSoup(..., "html.parser")).
  3. **Expandir Comentários Truncados:** Antes de extrair os comentários, o script tenta garantir que textos longos de comentários não fiquem cortados. Por vezes, o Booking.com mostra apenas uma parte do comentário e requer um clique em “Ver todos os comentários” para exibir o texto completo. O código procura um botão cujo texto contenha "Ver todos os coment" e, se encontrado e visível, realiza um .click() nele. Após clicar, aguarda mais 2 segundos para que o conteúdo extra seja carregado e atualizado no DOM. Este passo assegura que, ao analisar o HTML com BeautifulSoup, já se tenha o comentário completo visível.
  4. **Seleção dos blocos de review:** Com o HTML (eventualmente atualizado) da página, o BeautifulSoup é utilizado para selecionar todos os elementos correspondentes a um cartão de review. O seletor usado é [data-testid="review-card"], que identifica cada bloco de comentário individual no código do Booking.com. O resultado é uma lista de elementos (no BeautifulSoup) que representam cada review naquela página.
  5. Se por algum motivo nenhum comentário for encontrado (lista vazia), o script emite um aviso no terminal (“⚠️ Nenhum review encontrado nesta página.”), mas normalmente isso só ocorreria se a página de reviews não tiver comentários (por exemplo, hotel sem reviews) ou se algo falhou no carregamento.
  6. **Extração dos campos de cada review:** O script então itera por cada elemento de review obtido:
     + Para cada comentário, tenta extrair, usando seletores dentro do elemento review:
       - **Nome do autor do comentário:** Selector [data-testid="review-avatar"] div[class^="a3332d"] – geralmente o nome ou apelido do cliente que fez o comentário.
       - **Nacionalidade do autor:** Selector [data-testid="review-avatar"] span[class^="afac"] – geralmente aparece ao lado do nome, indicando país/origem do comentador.
       - **Tipo de quarto reservado:** Selector [data-testid="review-room-name"] – indica qual foi o quarto/unidade que o cliente ocupou (ex.: "Quarto Duplo", "Apartamento de 1 Quarto", etc.).
       - **Número de noites da estadia:** Selector [data-testid="review-num-nights"] – texto do tipo "X noites" (por vezes inclui também informação sobre o mês da estadia).
       - **Data do comentário:** Selector [data-testid="review-date"] – normalmente algo como "Data do comentário: 12 de março de 2024" (que o código depois ajusta, removendo o prefixo e formatando apenas a data).
       - **Pontuação:** Selector [data-testid="review-score"] – o score dado pelo hóspede, geralmente apresentado como “Pontuado com 8” ou “10/10”. O código trata este campo para uniformizar em formato numérico com decimal (por exemplo, "8.0" em vez de "8", e "10" é tratado como "10" literalmente). É feita uma substituição de vírgula por ponto e removido texto desnecessário como "Pontuado com".
       - **Título do comentário:** Selector [data-testid="review-title"] – muitos utilizadores fornecem um título resumido para a sua avaliação (ex.: "Experiência excelente", "Dececionante").
       - **Texto do comentário positivo:** Selector [data-testid="review-positive-text"] – o conteúdo do comentário referente aos pontos positivos da estadia, geralmente precedido por um ícone 👍 no site. Pode ser vazio se o utilizador não destacou nada positivo em particular.
       - **Texto do comentário negativo:** Selector [data-testid="review-negative-text"] – o conteúdo referente aos pontos negativos (👎), ou vazio se não houve menção negativa.
       - **Tipo de viajante:** Selector [data-testid="review-traveler-type"] – indica a categoria do viajante (ex.: "Casal", "Viajante individual", "Família", "Grupo de amigos", etc.).
       - **Data da reserva (estadia):** Selector [data-testid="review-stay-date"] – tipicamente o mês/ano em que ocorreu a estadia ou quando a reserva foi feita, por ex.: "Estadia em Março de 2024" (o código extrai apenas a parte relevante, e.g., "Março, 2024").
     + Para cada um desses campos, o script obtém o texto (.text via BeautifulSoup) e faz *strip* de espaços extras. Onde aplicável, efetua pequenas limpezas (por exemplo, remover o prefixo "Data do comentário: " do campo data, remover caracteres indesejados como o ponto mediano "·" do tipo de quarto, etc.).
     + Todos os valores são então armazenados num dicionário representando aquele comentário, incluindo também o nome do hotel atual no campo “Hotel” para referência. Por exemplo, um dicionário de comentário poderia ser:

python

CopiarEditar

{ "Hotel": "Hotel Exemplo 1", "Nome": "João", "Nacionalidade": "Portugal", "Tipo de Quarto": "Quarto Duplo Deluxe", "Nº de Noites": "3 noites", "Data do Comentário": "12 de março, 2024", "Pontuação": "9.0", "Título": "Excelente estadia", "Comentário Positivo": "O quarto era espaçoso e limpo, e o pequeno-almoço muito variado.", "Comentário Negativo": "Nada a apontar.", "Tipo de Viajante": "Casal", "Data da Reserva": "Março, 2024" }

* + - Esse dicionário é adicionado à lista comentarios\_total que armazena todos os comentários recolhidos. Caso ocorra alguma exceção durante a extração de um comentário específico (por exemplo, falha ao encontrar um sub-elemento), o script apanha o erro, imprime uma mensagem “⚠️ Erro ao extrair review: ...” e continua com os restantes.
  1. **Navegar para a próxima página:** Depois de processar todos os comentários da página atual, o script tenta avançar para a página seguinte. Para tal, utiliza o Selenium para encontrar o botão “Página seguinte” (que geralmente tem um atributo aria-label="Página seguinte"). É usada uma espera explícita curta (3 segundos) para garantir que o botão está clicável. Em seguida:
     + Verifica-se se o botão possui a classe ou estado “disabled” (desativado). Se sim, significa que chegámos à última página de reviews (não há mais comentários além dos já exibidos). Nesse caso, imprime-se “Não há mais páginas (botão desativado)” e faz-se break do laço while, terminando a extração para aquele hotel.
     + Se o botão não estiver desativado, procede-se a clicar nele (botao\_proximo.click()) para carregar a página seguinte. Incrementa-se o contador pagina += 1 e espera-se 2 segundos para dar tempo ao conteúdo da nova página ser carregado. Em seguida, o loop while reitera, extraindo a próxima página de comentários.
     + Se ocorrer um TimeoutException ao procurar/clicar no botão (ou seja, se o botão não for encontrado em 3s) isso também é interpretado como não haver próxima página e o loop é interrompido.
     + Existe ainda o tratamento para um caso especial: ElementClickInterceptedException. Isso pode acontecer se algo estiver a bloquear o botão (por exemplo, um overlay inesperado). Nesse cenário, o script avisa “⚠️ Clique bloqueado — clica manualmente e pressiona ENTER para continuar.” e aguarda input do utilizador. Ou seja, permite intervenção manual (o utilizador pode, na janela do Chrome aberta, fechar algum popup ou clicar no botão) e depois prosseguir ao pressionar ENTER na linha de comando. Esta é uma forma de contornar manualmente eventuais obstáculos não programados, sem perder o progresso.
* Este loop continua até não haver mais páginas de comentários para aquele hotel. Nesse ponto, todos os comentários desse hotel estarão acumulados em comentarios\_total.

**Salvaguarda de Dados e Progresso:**

* Assim que os comentários de um hotel são concluídos (ou seja, ao sair do loop de páginas), o script realiza duas ações importantes antes de passar ao próximo hotel:
  1. **Guardar resultados temporários:** Invoca a função guardar\_excel(temp=True), a qual guarda o conteúdo atual de comentarios\_total num Excel temporário (comentarios\_final\_temp.xlsx). Essa função simplesmente converte a lista de dicionários em DataFrame Pandas e escreve para o ficheiro, informando no terminal que o ficheiro foi guardado. Ao fazer isto após cada hotel, o projeto garante que, mesmo que a execução pare abruptamente no hotel seguinte, já se tem armazenado em disco tudo o que foi extraído até ali.
  2. **Atualizar progresso:** Abre (ou cria) o ficheiro progress.json e escreve um objeto JSON com o índice do próximo hotel a ser processado. Especificamente, grava {"hotel\_index": X} onde X é o índice atual (base 0) acrescido de 1. Por exemplo, se acabou de processar o hotel de índice 4, escreverá índice 5 como próxima posição. Assim, caso o script seja reiniciado, saberá que os hotéis de 0 a 4 já estão prontos e começará no 5.
* Depois disto, o loop segue para o próximo hotel e repete todo o processo (carregar página, extrair comentários em todas as páginas, guardar temp, etc.).

**Interrupção e Continuidade:**

* Um aspecto importante é a capacidade de **retomar** em caso de interrupção manual. Se o utilizador decidir parar a execução enquanto o script corre (por exemplo, pressionando Ctrl+C no terminal), o código captura a exceção KeyboardInterrupt. Nessa captura, antes de finalizar, ele:
  + Informa “Interrompido — a guardar ficheiro temporário...” no terminal.
  + Chama novamente guardar\_excel(temp=True) para não perder comentários possivelmente em curso.
  + Atualiza o progress.json com o índice do hotel atual (note: aqui usa idx sem +1, para que ao retomar recomece do mesmo hotel que estava incompleto ou por iniciar).
  + Fecha o driver e sai do programa.
* Com isso, o utilizador pode posteriormente reexecutar o script e ele continuará do ponto certo, evitando extrair novamente hotéis já concluídos e inclusive aproveitando os comentários já guardados da última execução parcial.

**Finalização:**

* Se o loop completar todos os hotéis sem interrupção, o script procede a salvar um ficheiro Excel final com todos os comentários: guardar\_excel(temp=False). Isso criará um comentarios\_final.xlsx (distinto do temporário) consolidando todos os dados.
* Em seguida, imprime “🟢 Scraping concluído com sucesso.” e espera o utilizador pressionar ENTER para fechar (esta pausa final é meramente estética, para o utilizador ler a mensagem). Finalmente, encerra o navegador Chrome (driver.quit()) e termina a execução.
* Opcionalmente, o progress.json final indicará um índice igual ao número total de hotéis, significando que não há mais nada pendente.

**Estrutura do Output (**comentarios\_final.xlsx**):** O resultado principal deste script é um Excel com todos os comentários coletados. Cada linha representa um comentário individual feito por um hóspede, e as colunas incluem os campos detalhados anteriormente. Recapitulando, as colunas são:

* **Hotel** – Nome do hotel ao qual este comentário pertence.
* **Nome** – Nome (ou primeiro nome) do autor do comentário.
* **Nacionalidade** – País ou região de origem do autor.
* **Tipo de Quarto** – Categoria do quarto/unidade reservada pelo hóspede.
* **Nº de Noites** – Duração da estadia (número de noites).
* **Data do Comentário** – Data em que o comentário foi publicado (formato textual, ex: "14 de outubro, 2024").
* **Pontuação** – Avaliação numérica dada (escala de 1 a 10, possivelmente com decimal .0).
* **Título** – Título resumido do comentário, se dado.
* **Comentário Positivo** – Texto referente aos aspectos positivos mencionados.
* **Comentário Negativo** – Texto referente aos aspetos negativos mencionados.
* **Tipo de Viajante** – Perfil do viajante (Casal, Família, etc.).
* **Data da Reserva** – Mês/ano da estadia ou reserva (ex.: "Outubro, 2024").

Para dar uma noção concreta, segue um exemplo de duas entradas de comentários reais (simplificados) extraídos pelo projeto para um mesmo hotel:

* **Hotel:** ASPA Studios - Urban Conscious Living  
  **Nome:** Reinaldo – **Nacionalidade:** Brasil – **Tipo de Viajante:** Casal – **Data da Reserva:** Outubro, 2024  
  **Estadia:** *Estúdio Superior, 4 noites* – **Comentário em:** 14 de outubro, 2024 – **Pontuação:** 10.0  
  **Título:** *"Excecional"*  
  **Positivo:** *"Limpeza, localização, conforto e pequeno-almoço excelentes."*  
  **Negativo:** *"(nenhum comentário negativo)"*
* **Hotel:** ASPA Studios - Urban Conscious Living  
  **Nome:** Rosilene – **Nacionalidade:** Brasil – **Tipo de Viajante:** Casal – **Data da Reserva:** Julho, 2024  
  **Estadia:** *Estúdio Superior, 5 noites* – **Comentário em:** 14 de agosto, 2024 – **Pontuação:** 8.0  
  **Título:** *"Ótima experiência!"*  
  **Positivo:** *"Café da manhã muito bom, ótima localização. Quarto limpo e bem equipado."*  
  **Negativo:** *"Faltou um ar condicionado para maior conforto à noite."*

Cada um destes representa uma linha na tabela final. Como se pode observar, os comentários positivos e negativos fornecem insights diretos sobre a opinião dos hóspedes, enquanto os demais campos contextualizam quem fez o comentário e em que condições (tipo de quarto, data, etc.). No total, após a execução completa do script para os 124 hotéis, obteve-se **mais de 120.000 comentários**, cada um representado no Excel final com este nível de detalhe.

Cabe mencionar que, devido ao grande volume de comentários, o autor optou por salvar periodicamente os resultados em curso (comentarios\_final\_temp.xlsx) e, se necessário, poderia até dividir o output em vários ficheiros para facilitar manuseio. No diretório do projeto, por exemplo, encontram-se ficheiros PARTE1.xlsx, PARTE2.xlsx, etc., que indicam que em alguns momentos os dados de comentários foram fracionados. Esta fragmentação não era estritamente necessária (120 mil linhas cabem num único ficheiro Excel dentro do limite de ~1 milhão de linhas), mas pode ter sido feita para contornar limitações de memória ou simplesmente para organizar os dados em partes manejáveis durante a coleta. De qualquer forma, o *scraper* em si produz (ou pode produzir) um ficheiro único consolidado de todos os comentários.

**2.4. Dados Extraídos e Estrutura dos Outputs**

Recapitulando os outputs gerados pelo projeto:

* hotel\_info.xlsx**:** contém informações gerais de cada hotel (uma linha por hotel, com colunas detalhando rating médio, descrição, morada, comodidades, etc.). Este ficheiro permite, por exemplo, ter uma visão geral dos **124 hotéis** alvo, incluindo a classificação média no Booking.com e os principais pontos positivos apontados (destaques e categorias avaliadas). É um complemento útil à análise dos comentários, pois podemos relacionar as opiniões individuais com os dados globais do hotel.
* comentarios\_final.xlsx**:** contém todos os comentários extraídos (uma linha por comentário, com identificação do hotel e todos os detalhes da review). Este é o principal resultado para análise textual ou estatística das opiniões dos hóspedes. Com mais de **120.000 entradas**, trata-se de um conjunto de dados considerável, possibilitando diversas análises – por exemplo, avaliar qual hotel tem melhores comentários, quais os aspetos mais elogiados ou criticados pelos clientes, tendências ao longo do tempo, etc.

Em termos de exemplos concretos, já fornecemos algumas entradas exemplificativas acima. Poderíamos adicionalmente imaginar análises a partir destes outputs, mas mantendo o foco no escopo deste relatório técnico, o importante é entender que os dados extraídos estão **estruturados** e prontos para consumo: seja numa ferramenta de análise de dados, seja para geração de relatórios ou dashboards que o autor do projeto ou terceiros possam querer elaborar.

A escolha do formato Excel (.xlsx) para os resultados foi estratégica: por um lado, facilita a abertura e consulta manual dos dados (por exemplo, num Microsoft Excel ou equivalente) e por outro, pode ser facilmente importado em linguagens de programação ou ferramentas de data science (Python/Pandas, R, etc.) para processamento adicional. Alternativamente, poderia-se ter optado por CSV ou uma base de dados, mas o Excel entrega uma solução pronta a usar, ideal para apresentação acadêmica ou para um utilizador final menos técnico visualizar os resultados.

Nos próximos capítulos, discutiremos o **desempenho obtido** durante o scraping e os **desafios específicos encontrados**, bem como as soluções implementadas para os contornar.

**3. Desempenho e Tempo de Execução**

Um aspeto crítico em projetos de scraping de larga escala é o **tempo de execução**. Executar um navegador real via Selenium para carregar centenas de páginas e extrair milhares de elementos é uma tarefa relativamente pesada, tanto em termos de tempo quanto de recursos computacionais (CPU, memória) e de rede. No caso deste projeto, os resultados mostram um desempenho aceitável, tendo em conta o volume de dados extraído.

Durante os testes e execução final, verificou-se que o script de comentários conseguia extrair em média cerca de **10 comentários em 5 segundos**. Em outras palavras, a taxa aproximada de scraping foi de 2 comentários por segundo. Pode parecer pouco em comparação com abordagens puramente HTTP (que poderiam potencialmente alcançar dezenas por segundo), mas é um desempenho compreensível dado que cada conjunto de comentários exige carregar uma página web completa (incluindo todo o layout, imagens, scripts do Booking.com) e que o script introduz voluntariamente pequenas esperas (sleep) para garantir estabilidade e evitar problemas. Além disso, utilizando Selenium, cada “página” de comentários carrega tipicamente 10 a 20 reviews de uma só vez, portanto a produtividade real é agregada por página e não por requisição individual de comentário.

Em números globais, para a extração de **mais de 120.000 comentários distribuídos por 124 hotéis**, o tempo total de execução observado foi de aproximadamente **16 horas contínuas**. Esse tempo inclui todo o processo automatizado, do primeiro ao último hotel, navegando página a página. Este valor significa que, numa única sessão de scraping (por exemplo, iniciando ao fim do dia e terminando na manhã seguinte), foi possível recolher toda a base de dados pretendida.

Alguns fatores que influenciam o tempo de execução e merecem nota:

* **Latência de Rede:** Como o script necessita carregar muitas páginas da web, a velocidade da conexão internet e a latência do servidor do Booking.com influenciam o tempo. A cada novo hotel (e a cada página de comentários dentro do hotel), há um novo carregamento HTTP. Se a conexão for rápida, o limite acaba por ser mais do lado do script (esperas fixas, processamento HTML); se for lenta, o carregamento de páginas pode tornar-se o fator dominante.
* **Delays Intencionais:** O código implementa time.sleep(4) após carregar cada novo hotel e sleep(2) a cada troca de página de reviews. Esses valores estão calibrados para um equilíbrio entre confiabilidade e rapidez. Poder-se-ia tentar reduzir esses tempos para acelerar a execução, mas isso poderia levar a falhas intermitentes (elementos não carregados a tempo) ou até marcar comportamento suspeito pela plataforma. Portanto, as ~16 horas incluem já essas esperas deliberadas.
* **Capacidade do Sistema:** O computador utilizado para executar o scraping deve ter recursos suficientes para manter um navegador aberto por tantas horas. O Chrome tende a consumir memória; felizmente, a navegação foi sequencial (um hotel de cada vez, uma página de cada vez), o que ajuda a limitar o uso de recursos, pois a cada página nova as anteriores são descartadas. Ainda assim, ao final, o Python estava a segurar um array de 120k comentários em memória, o que exige alguns gigabytes de RAM para conforto. Pelo menos 8 GB de RAM seria recomendável para um scraping desta magnitude; se a memória fosse menor, o processo poderia começar a ficar mais lento devido a *swap*.
* **I/O de Disco:** Escrever periodicamente para Excel (especialmente com 120k linhas no final) tem um custo. O uso do Excel temporário a cada hotel mitigou o risco de perder dados, mas também significa que a cada iteração de hotel era feita escrita em disco (Pandas to\_excel), o que também consome alguns segundos. No cômputo geral, isso adiciona tempo (provavelmente uns 5-15 segundos por hotel só para I/O). Com 124 hotéis, o overhead de I/O pode ter sido da ordem de 10 a 30 minutos distribuídos no total, o que não é crítico mas existe.

Resumindo, o desempenho alcançado foi satisfatório para os objetivos do projeto. **16 horas** para coletar **120 mil comentários** equivalem a dizer que, em média, cada hotel (com ~970 comentários em média) demorou cerca de 7-8 minutos de processamento completo. Hotéis com menos reviews terminaram mais rápido; hotéis com milhares de reviews levaram proporcionalmente mais tempo.

É evidente que, se precisássemos repetir este processo frequentemente ou em maior escala (mais hotéis ou comentários), seria útil pensar em otimizações (ver secção de melhorias). Todavia, para uma coleta única ou esporádica, este tempo de execução é perfeitamente manejável. Além disso, a capacidade de retomar o processo em caso de falha significa que mesmo que não se consiga executar as 16 horas de uma vez, pode-se dividir em sessões menores (por exemplo, 4 dias executando 4 horas por dia) sem perda de eficiência global, dado que não se repete trabalho já feito.

**4. Desafios e Soluções Implementadas**

Projetos de web scraping enfrentam frequentemente diversos **desafios práticos**, sobretudo quando lidam com sites complexos ou com grandes volumes de dados. No caso do Booking.com, alguns dos obstáculos identificados incluem a necessidade de lidar com *pop-ups* de cookies, conteúdo carregado dinamicamente mediante interação (paginação, expansão de texto), e a robustez da execução ao longo de muitas horas. Nesta secção discutimos as principais dificuldades encontradas e as soluções que o projeto incorporou para as superar.

**4.1. Cookies e Consentimento**

**Desafio:** Quase todos os sites atualmente exibem um aviso de consentimento de cookies na primeira vez que um utilizador os visita, devido às regulamentações de privacidade (como o RGPD na UE). No contexto de um *scraper*, este aviso é um elemento extra que pode **bloquear a página** ou impedir a interação com outros elementos enquanto não for resolvido. Se o script não lidar com isto, poderia ficar preso sem conseguir extrair dados (por exemplo, o banner de cookies poderia cobrir o botão “Próxima página” ou impedir cliques em geral).

**Solução Implementada:** O script de reviews integrou um mecanismo para **aceitar automaticamente os cookies** assim que carrega a página de comentários de um hotel. Como descrito, ele usa uma espera explícita e procura pelo botão "Aceitar" identificado por onetrust-accept-btn-handler, clicando nele se disponível. A utilização do Selenium aqui é crucial: apenas um navegador real renderiza e expõe esse elemento; uma abordagem via requests não veria esse pop-up por ser algo executado em JavaScript no cliente. Com esta solução, o obstáculo dos cookies é removido nos primeiros segundos e a raspagem pode prosseguir sem interferências.

Em casos onde o cookie banner já não aparece (por exemplo, se já foi aceito anteriormente e a preferência ficou guardada numa instância do navegador), o script simplesmente continua – graças ao bloco try/except que ignora exceções nessa parte. Portanto, é uma solução *idempotente*: tenta sempre aceitar cookies, mas não falha se não houver cookies para aceitar.

**4.2. Carregamento Dinâmico e Paginação**

**Desafio:** O Booking.com não exibe todos os comentários de um hotel numa única página por padrão. Em vez disso, implementa uma **paginação**: normalmente são exibados 10 comentários por página, com botões de navegação ("Página seguinte", "Página anterior") para o utilizador percorrer todas as reviews. Além disso, cada comentário individual pode ser truncado se for muito longo, exibindo um botão “Ver todos os comentários” para expandir o texto completo. Esses mecanismos significam que, para coletar *todos* os comentários, o scraper deve:

* Repetidamente acionar o carregamento de novas páginas de reviews (clicando em “Página seguinte” até esgotar).
* Em cada página, garantir que o texto do comentário está completo (clicando em “Ver todos os comentários” nos casos aplicáveis).

Tudo isso requer interações ativas, não bastando fazer um simples GET numa API, por exemplo. Ademais, existe a questão do **scroll infinito** em alguns sites – no Booking.com especificamente, a versão desktop usa paginação com botão, mas se fosse scroll contínuo, teríamos que simular scrolls sucessivos.

**Solução Implementada:** A estratégia adotada no script scraper\_reviews\_pagination\_fixed\_v4.py lida com a paginação e conteúdo dinâmico da seguinte forma:

* Para a paginação, conforme detalhado, o script **identifica e clica no botão “Página seguinte”** automaticamente. A utilização de WebDriverWait assegura que não clicamos antes do botão estar pronto, e a verificação da classe "disabled" ou falta do botão permite detectar o fim da lista de páginas. Dessa maneira, o scraper percorre sequencialmente todas as páginas sem intervenção manual, a não ser em casos excecionais. Ao clicar programaticamente no botão, desencadeamos o mesmo evento que um utilizador faria, carregando a página seguinte de comentários.
* Para comentários truncados, antes de extrair os campos de cada review, o código **procura e clica em “Ver todos os comentários”** dentro daquela página, se o botão estiver presente. Esta ação expande quaisquer textos cortados ("..." indicando continuação) para o seu tamanho completo. Uma vez expandido, o conteúdo completo do comentário passa a estar presente no driver.page\_source e, assim, o BeautifulSoup consegue capturá-lo integralmente. Se não fizéssemos isso, poderíamos acabar por armazenar comentários incompletos, o que comprometeria a qualidade dos dados.
* Em termos de carregamento dinâmico via JavaScript, usar o Selenium contorna o problema – como o Chrome rende tudo, o page\_source resultante já inclui o HTML pós-JS. Em casos de scroll infinito (se existisse no Booking), provavelmente usaríamos uma abordagem de executar JavaScript scrolls ou repetidos .execute\_script("window.scrollTo(0, document.body.scrollHeight);") até não aparecer mais conteúdo. No nosso caso concreto, isso não foi necessário devido à paginação via botão.

**Experiência e Robustez:** Inicialmente, implementar paginação robusta pode requerer tentativa-e-erro. Por exemplo, se o site for muito rápido ou lento, o timing do clique e do parsing pode desalinhar. O projeto mitigou isso com pequenas esperas fixas e verificações. A mensagem de log “📄 Não há mais páginas (botão desativado).” indica que foi identificada corretamente a última página. Houve também o tratamento de ElementClickInterceptedException – isto é, em algum teste verificou-se que, ocasionalmente, o clique no botão "Página seguinte" podia falhar por estar coberto ou não receber o evento. A solução implementada foi engenhosa: alertar o operador e permitir intervenção manual para então continuar. Embora não 100% automática, essa abordagem garantiu que mesmo cenários imprevistos não travariam definitivamente o progresso; o pior caso seria o utilizador ter que dar um empurrão manual em alguns pontos. Nas execuções finais, não há indicação de que isso tenha sido necessário (caso contrário, dificilmente seria viável colher 120k comentários manualmente), logo a automatização mostrou-se satisfatória.

**4.3. Retomar Execução e Tolerância a Interrupções**

**Desafio:** Considerando que o scraping completo demoraria muitas horas, havia um risco real de que a execução pudesse ser interrompida antes da conclusão. Os motivos para tal interrupção podem ser variados: uma falha no navegador ou no sistema, uma queda de conexão, ou simplesmente a decisão do utilizador de pausar e continuar depois (por não poder ou querer deixar o computador ligado continuamente por 16 horas). Sem precauções, uma interrupção significaria perder todo o progresso não salvo e possivelmente recomeçar do zero – o que seria extremamente ineficiente, sobretudo com 120k comentários já recolhidos.

**Solução Implementada:** O projeto incorporou um mecanismo de **checkpointing** simples mas eficaz:

* Após completar cada hotel, salva o progresso atual num ficheiro Excel temporário (comentarios\_final\_temp.xlsx). Assim, os dados mais recentes estão sempre armazenados em disco. Mesmo se o programa terminar abruptamente, esse ficheiro contém tudo até o último hotel terminado.
* Paralelamente, mantém um ficheiro progress.json com o índice do próximo hotel a processar. Este índice atua como um ponteiro para retomar depois. Na inicialização do script, esse ficheiro é lido, ajustando a posição inicial da iteração.
* Ao capturar uma interrupção manual (KeyboardInterrupt), o script também aciona imediatamente a gravação do Excel temporário e atualização do progresso antes de sair de fato.

Com esse arranjo, se por exemplo a execução for parada ao fim do 50º hotel, o comentarios\_final\_temp.xlsx conterá os comentários de 50 hotéis e o progress.json indicará que o próximo a iniciar é o hotel de índice 50 (assumindo indexação a partir de 0). Quando o script for executado novamente, ele detectará esses artefatos, carregará os 50 hotéis de dados já obtidos para memória e começará diretamente do 51º hotel, **sem repetir** nenhum dos anteriores. Isso permite realizar o scraping em sessões múltiplas sem redundância.

Este tipo de tolerância a falhas é fundamental em projetos longos. No desenvolvimento, o programador testou e ajustou este comportamento – por exemplo, garantindo que não haja duplicação de dados ao concatenar o Excel existente com novos comentários (daí usar Pandas para ler e depois continuar a preencher a mesma lista de dicionários).

Em termos de implementação, o uso de JSON para progresso é simples e interoperável, e o Excel temporário aproveita o mesmo formato final. Poder-se-ia também usar um formato mais leve (CSV) para salvar incrementos, mas optou-se pelo Excel por conveniência.

**Outros Desafios Menores e Soluções:**

* **Seletores e Mudanças de Layout:** Um risco em scraping é o site mudar seu HTML, quebrando os seletores. Durante o desenvolvimento, o autor utilizou principalmente seletores estáveis fornecidos por data-testid, os quais são menos propensos a alteração frequente comparado a classes genéricas. Ainda assim, se o Booking.com alterasse esses atributos, seria necessário atualizar o código. Como mitigação, o autor testou previamente em vários hotéis e manteve logs (pasta logs/ no projeto) para verificar eventuais erros de “elemento não encontrado” e ajustar seletores ou adicionar try/except para ignorar partes não críticas.
* **Desempenho vs. Confiabilidade:** A adição de *delays* fixos foi uma solução para garantir confiabilidade (páginas completamente carregadas) ao custo de performance. Outra abordagem poderia ser usar apenas WebDriverWait com condições explícitas (por exemplo, esperar que um certo elemento de comentário esteja presente em vez de um sleep cego de 2s por página). No script, vemos que para o botão de próxima página usou-se WebDriverWait, mas para o carregamento geral usou-se sleeps. Isto simplifica a implementação e, dado que o tempo total era ainda viável, preferiu-se a segurança de esperar um pouco mais do que o estritamente necessário a arriscar condições de corrida.
* **Memória e Tamanho dos Ficheiros:** Já mencionado no desempenho, mas vale reforçar: 120k comentários no Excel geram um ficheiro consideravelmente grande (dezenas de MB). O Pandas lida bem com essa quantidade, mas a escrita poderia ser pesada. O autor fragmentou outputs em partes (PARTE1.xlsx, etc.) possivelmente para inspeção ou para contornar limites de software (por exemplo, Excel às vezes tem dificuldade em abrir ficheiros >~50 MB dependendo da máquina). Essa divisão não afetou o scraper em si, mas foi uma solução pragmática para manusear o resultado fora do Python.

Em suma, os desafios foram abordados com soluções práticas que privilegiaram a **confiabilidade** e a **completude da extração**. O projeto demonstrou conseguir navegar por obstáculos típicos (cookies, paginação, long running process) garantindo que o objetivo final – coletar todos os comentários e info dos hotéis – fosse alcançado.

**5. Conclusão e Reflexão Crítica**

O projeto "Automatização da Extração de Comentários e Informações Gerais do Booking.com" atingiu com sucesso o seu propósito de coletar uma vasta quantidade de dados a partir da plataforma Booking.com de forma automatizada. Através da integração eficaz de ferramentas como Selenium e BeautifulSoup, foi possível extrair **mais de 120 mil comentários de clientes** e **informações detalhadas de 124 hotéis**, estruturando-os em bases de dados facilmente utilizáveis. Este relatório técnico detalhou o funcionamento interno do scraper, desde os conceitos teóricos até às implementações práticas, passando pelos desafios superados. Nesta conclusão, faz-se uma reflexão sobre a experiência adquirida, as implicações de se raspar grandes volumes de dados e apontam-se direções para melhorias futuras.

**Reflexão sobre o processo de scraping de grande volume:** Realizar web scraping em larga escala traz lições importantes. Primeiro, a **robustez** do código torna-se essencial – pequenos imprevistos que em um scraping de 100 ou 200 itens seriam raros, em 120 mil ocorrências tornam-se quase certeiros. O projeto mostrou isso ao incorporar mecanismos de retoma e ao prever situações como cliques bloqueados. Segundo, a **ética e legalidade** não devem ser negligenciadas: embora os dados extraídos sejam públicos, a forma de obtê-los não deixa de gerar carga no site. É crucial usar intervalos (delays) e não fazer mais requisições simultâneas do que um utilizador comum faria. Felizmente, ao distribuir a coleta em 16 horas, a taxa de acesso ao Booking.com manteve-se modesta, diminuindo a chance de bloqueio ou de impactar negativamente o servidor. Em nenhum momento foi reportado um bloqueio de IP ou apresentação de CAPTCHA, sinal de que o scraper operou dentro de limites aceitáveis.

**Manutenção do projeto:** Um ponto a considerar é a **manutenção futura**. Websites estão em constante evolução – o Booking.com pode mudar o design das páginas de comentários, alterar seletores ou fluxos de interação. Um projeto como este deve, portanto, ser visto como uma fotografia válida no momento atual (2025), mas que requererá ajustes se for reutilizado mais adiante. A escolha de seletores baseados em data-testid e a abordagem geral de Selenium dão alguma resiliência, mas não garantem perpetuidade. Por isso, documentar bem o código (como foi feito, inclusive com um README de utilização) e estar preparado para depurar novas situações são partes integrantes de um projeto de scraping.

**Possíveis Melhorias e Extensões:**

* *Paralelização:* Uma forma de acelerar a extração seria rodar múltiplos navegadores em paralelo, cada um cuidando de um subconjunto de hotéis. Isso poderia potencialmente reduzir o tempo total de 16h para, por exemplo, 4h se usássemos 4 threads/browsers em paralelo. No entanto, essa abordagem exige uma máquina mais potente (ou várias máquinas) e aumentaria a carga no site (4x mais requisições simultâneas), o que poderia levantar flags de bloqueio. Teria de ser implementado com cautela e preferencialmente respeitando limites (ex.: não iniciar todos ao mesmo segundo, etc.).
* *Headless Browser:* Atualmente, o Selenium abre o Chrome visível. Poder-se-ia usar o modo **headless** (sem interface gráfica) para reduzir consumo de recursos e talvez ganhar alguma velocidade. Contudo, em certos casos headless Chrome comporta-se de forma ligeiramente diferente (especialmente em questões de tempo de carregamento ou detecção pelo site). Seria interessante testar se o scraping mantém estabilidade em modo headless, pois isso facilitaria também a execução em servidores ou em background.
* *Refinamento dos Delays:* Substituir alguns dos time.sleep por esperas condicionais mais inteligentes poderia otimizar o tempo sem comprometer a confiabilidade. Por exemplo, esperar até que o seletor do último comentário da página esteja presente em vez de esperar arbitrariamente 2 segundos. São afinações que poderiam talvez poupar algumas horas no total, principalmente em páginas onde o carregamento é rápido.
* *Output em Base de Dados:* Para um volume de dados tão grande, pode ser mais eficiente armazenar diretamente os resultados numa base de dados (SQL ou NoSQL) ao invés de um Excel. Uma base de dados permitiria consultas mais flexíveis e manipulação de dados mais eficiente que um ficheiro Excel monolítico. No contexto académico, o Excel foi suficiente, mas se este projeto fosse parte de um produto ou análise contínua, migrar para um armazenamento mais robusto seria recomendável.
* *Análise de Sentimento ou NLP:* Embora fora do escopo do scraping em si, um passo natural seguinte seria aplicar técnicas de análise de texto nos comentários recolhidos. Poder-se-ia integrar no projeto uma etapa de processamento adicional, como classificar automaticamente os comentários em positivos/negativos (além da auto-avaliação do cliente), extrair as palavras mais frequentes nas críticas, ou correlacionar pontuações com palavras-chave. Isso acrescentaria valor ao projeto no sentido de transformar os dados brutos em insights.
* *Interface Gráfica ou API:* Para tornar a ferramenta mais acessível a terceiros, poderia ser desenhada uma pequena interface (por exemplo, uma interface web ou um GUI desktop) onde o utilizador pudesse fornecer um ficheiro de hotéis e acionar o scraping com um botão, visualizando o progresso. Ou então encapsular a funcionalidade numa API/biblioteca Python para que outros projetos pudessem reutilizá-la facilmente.
* *Respeito a robots.txt e Termos de Uso:* Uma melhoria ética seria implementar leitura do robots.txt do Booking.com e/ou verificar os termos de uso para identificar restrições ao scraping. Embora muitas vezes robots.txt proíba scrapers não autorizados, no mínimo isso informa decisões conscientes. Neste projeto académico, tal verificação não foi mencionada, mas em contexto profissional seria algo a incorporar.

**Conclusão Final:** Em conclusão, o projeto evidencia a viabilidade de se automatizar a extração de informações ricas de uma plataforma web complexa combinando ferramentas apropriadas. Através deste trabalho, Diogo Amorim demonstrou competências em diversos domínios: desde a compreensão do funcionamento web (HTML, DOM, etc.), programação em Python com bibliotecas de automação (Selenium) e parsing (BeautifulSoup), manipulação de dados (Pandas) e estratégia para lidar com desafios de longa duração. O resultado é um conjunto de dados substancial que pode servir a análises no domínio do turismo, hotelaria ou ciência de dados, mostrando opiniões de hóspedes e características de hotéis de forma sistematizada.

Naturalmente, a manutenção de um scraper implica atenção contínua e disponibilidade para atualizar o código conforme as condições mudem. Contudo, a arquitetura desenvolvida – modular (separando infos gerais e comentários), com checkpoints de progresso e uso eficaz de ferramentas – fornece uma base sólida para trabalhos futuros. Seja expandir para outros sites ou escalar o projeto, as lições aprendidas aqui serão certamente aplicáveis. Em um mundo cada vez mais orientado por dados, a habilidade de extrair informação estruturada da web é valiosa, e este projeto é um excelente exemplo prático dessa capacidade.

**Citações**

**[Web scraping – Wikipédia, a enciclopédia livre](Web scraping – Wikipédia, a enciclopédia livrehttps://pt.wikipedia.org/wiki/Web_scraping)**

[https://pt.wikipedia.org/wiki/Web\_scraping](Web scraping – Wikipédia, a enciclopédia livrehttps://pt.wikipedia.org/wiki/Web_scraping)

**[Web scraping – Wikipédia, a enciclopédia livre](Web scraping – Wikipédia, a enciclopédia livrehttps://pt.wikipedia.org/wiki/Web_scraping)**

[https://pt.wikipedia.org/wiki/Web\_scraping](Web scraping – Wikipédia, a enciclopédia livrehttps://pt.wikipedia.org/wiki/Web_scraping)

**[Selenium (software) – Wikipédia, a enciclopédia livre](Selenium (software) – Wikipédia, a enciclopédia livrehttps://pt.wikipedia.org/wiki/Selenium_(software))**

[https://pt.wikipedia.org/wiki/Selenium\_(software)](Selenium (software) – Wikipédia, a enciclopédia livrehttps://pt.wikipedia.org/wiki/Selenium_(software))

**[Selenium (software) – Wikipédia, a enciclopédia livre](Selenium (software) – Wikipédia, a enciclopédia livrehttps://pt.wikipedia.org/wiki/Selenium_(software))**

[https://pt.wikipedia.org/wiki/Selenium\_(software)](Selenium (software) – Wikipédia, a enciclopédia livrehttps://pt.wikipedia.org/wiki/Selenium_(software))

**[Raspagem da Web usando Beautiful Soup de Python](Raspagem da Web usando Beautiful Soup de Pythonhttps://brightdata.com.br/blog/procedimentos/aprenda-a-usar-beautiful-soup-para-raspar-a-web-com-python)**

[https://brightdata.com.br/blog/procedimentos/aprenda-a-usar-beautiful-soup-para-raspar-a-web-com-python](Raspagem da Web usando Beautiful Soup de Pythonhttps://brightdata.com.br/blog/procedimentos/aprenda-a-usar-beautiful-soup-para-raspar-a-web-com-python)