Dokumentation zu WebScraperBackend:

Schnittstellendokumentation:

- /create-graph (Schnittstelle zum Erstellen des Graphen):
    - Relevante Codebereiche:
        - web_parser.py (create_web_page_graph_handler)
        - web_page_loader.py (add_url, get_url)
        - normalizer.py (normalize_tree)
        - web_page_graph_handler.py (create_new_web_page_element, resolve_all_links)
    - Ablauf der Funktion create_web_page_graph_handler pro neuer Webseite:
        1. Laden der neuen Webseite durch Instanz der Klasse WebPageLoader (get_url)
        2. Normalisieren der Webseite durch Funktion normalize_tree
        3. Eintragen des neuen Elements in Instanz der Klasse WebPageGraphHandler (also in dem Graph)
        4. Hinzufügen neuer Webseiten zu Instanz der Klasse WebPageLoader zum parallelen Herunterladen (add_url)
        - Nachdem alle Webseiten verarbeitet wurden, werden im Graphen alle Link-Knoten durch die Methode resolve_all_links miteinander verbunden
- /progress (Gibt dem Benutzer den Fortschritt bei der Erzeugung des Graphen zurück)
- /data (Entnimmt dem Graphen die geforderten Daten):
    - Relevante Codebereiche:
        - web_parser.py (extract_main_data)
        - web_page_graph_handler.py (get_all_web_page_data)
        - web_page_data_processor.py (resolve_web_page_node_dict, map_elements, filter_for_paragraph_elements, process_paragraph_elements, filter_for_link_elements, process_link_elements)
    - Ablauf der Funktion extract_main_data:
        1. Holen aller DataElement Knoten ausgehend von den zugehörigen WebPageElement-Knoten (get_all_web_page_data)
        2. Entnehmen der Daten aus den Knoten (resolve_web_page_node_dict)
        3. Aufteilung der Menge von Objekten der Oberklasse DataElement in ParagraphElement-Objekte und LinkElement-Objekte

(map_elements, filter_for_paragraph_elements, process_paragraph_elements, filter_for_link_elements, process_link_elements)

- /keywords (Graph nach den angegebenen Keywords durchsuchen):
  - Relevante Codebereiche:
    - web_parser.py (search_keyword_list, search_keyword)
    - web_page_graph_handler.py (search_by_keyword)
    - web_page_data_processor.py (resolve_web_page_node_dict, map_elements, filter_for_paragraph_elements, process_paragraph_elements, filter_for_link_elements, process_link_elements)
  - Ablauf der Funktion search_keyword:
    1. Holen aller DataElement Knoten ausgehend von den zugehörigen WebPageElement-Knoten durch Suche nach Keyword (search_by_keyword)
    2. Entnehmen der Daten aus den Knoten (resolve_web_page_node_dict)
    3. Aufteilung der Menge von Objekten der Oberklasse DataElement in ParagraphElement und LinkElement (map_elements, filter_for_paragraph_elements, process_paragraph_elements, filter_for_link_elements, process_link_elements)
- /normalize-page (Gibt die angefragte Webseite in normalisierter Form zurück)

Script-Dokumentation:

- web_page_loader.py (Script, um Klasse WebPageLoader zu verwalten):
  - add_url: Hinzufügen neuer Webseite zu Stack
  - _process_url: Holen der Webseite
  - get_url: Abfragen von Webseite (Blockiert bis Seite geladen)
  - end_tasks: Leeren des Stacks und beenden des Ladeprozesses
- normalizer.py (Script für Normalisierungsprozess):
  - normalize_tree: Normalisiert HTML-Baum durch Aufruf aller notwendigen Funktionen
  - _clear_comment: Entfernt Kommentare aus HTML-Baum
  - _remove_unwanted_tags: Entfernt nicht benötigte Tags
  - _normalize_text: Entfernt Textdekorations-Tags (z.B. br, strong, …)
  - _wrap_all_strings: Erzeugt neue p-Tags und umschließt damit freien Text
  - _remove_empty_leafs: Entfernt zurückbleibende Blätter im Baum
  - _clear_nesting: Entfernt überflüssige Schachtelungen
  - _transform_structures: Wandelt Strukturen um (siehe Arbeit)
  - _replace_leftovers: Erneutes Entfernen zurückbleibende Blätter im Baum

- utils.py (Script für Hilfsfunktionen)
- web_parser.py (Script zum Verwalten des Scraping Prozesses; Siehe Schnittstellendokumentation /create-graph)
- web_page_graph_handler.py (Script zum Verwalten des Graphen; Siehe Schnittstellendokumentation /create-graph, /data, /keywords)
- web_page_data_processor.py (Script zum Verarbeiten der im Graph gefunden Daten; Siehe Schnittstellendokumentation /data, /keywords)
- app.py (REST-Controller; Siehe Schnittstellendokumentation)
- app_data_handler.py (Service-Script für REST-Controller; Nur für REST-Architektur relevant)