Got it! Here’s a detailed breakdown of **developer-centric subtasks** for building the **QueryCraft** project. These tasks will help you focus on the technical aspects, step by step, from setting up the environment to implementing core functionalities, testing, and deployment.

**Developer Subtasks for QueryCraft**

**1. Initial Project Setup**

**1.1. Create Project Structure**

* **Task**: Set up the directory structure.
  + Create folders: /data, /core, /app, /tests
  + Create a virtual environment and activate it (python -m venv venv)
  + Initialize a GitHub repository and create the following files:
    - README.md (Basic project description)
    - .gitignore (to exclude venv, \_\_pycache\_\_, etc.)
    - requirements.txt (list of dependencies)

**1.2. Install Dependencies**

* **Task**: Install necessary Python libraries.
  + Install the following packages via pip:
    - pandas (for data manipulation)
    - nltk, spaCy (for NLP preprocessing)
    - matplotlib, plotly (for data visualization)
    - streamlit (for the web interface)
    - fuzzywuzzy or difflib (for fuzzy matching columns)

pip install pandas nltk spacy matplotlib plotly streamlit fuzzywuzzy

**2. Data Ingestion and Processing**

**2.1. Upload CSV File**

* **Task**: Build functionality to upload and load CSV files.
  + Use Streamlit to create a file uploader widget.
  + Use pandas.read\_csv() to load the CSV file into a DataFrame.
  + Display the first few rows of the dataset and basic info (e.g., column names, data types).

**2.2. Extract Basic Data Profiling**

* **Task**: Generate summary statistics for the dataset.
  + Get the number of rows, columns, null values per column, data types, etc.
  + Display the results in an easy-to-read format (e.g., table or text).

**3. Natural Language Processing (NLP)**

**3.1. Preprocess User Questions**

* **Task**: Implement basic text preprocessing using spaCy or nltk.
  + Tokenize input questions (breaking them into words).
  + Lowercase, remove stop words, and perform lemmatization.

**3.2. Detect Question Intent**

* **Task**: Build a rule-based approach to identify the intent of the question.
  + Implement a set of rules to classify intents (e.g., "average", "top-k", "trend").
  + Use keyword matching to identify intents, such as "average", "sum", "show me trends", etc.
  + Example: "What is the average salary?" → Intent: Aggregation (average of salary column).

**3.3. Map Columns to DataFrame**

* **Task**: Match the column names in the dataset to the user’s question.
  + Use fuzzy matching (via fuzzywuzzy or difflib) to match columns in the dataset to the terms in the question.
  + Example: If the user asks about "total\_sales", but the dataset has a column named "sales", match them.

**4. Data Query Execution**

**4.1. Implement Aggregation Logic**

* **Task**: Implement logic to handle aggregation questions like sum, average, count.
  + For example, “What is the average salary?” should trigger df['salary'].mean().
  + Implement various functions for different aggregations (sum, mean, median, count, etc.).

**4.2. Implement Filtering and Sorting**

* **Task**: Add functionality to handle filtering (e.g., "show employees who joined after 2020") and sorting (e.g., "top 5 products").
  + Use pandas functions like .loc[] for filtering, and .sort\_values() for sorting.

**4.3. Implement Time-Series Analysis (Optional)**

* **Task**: Handle questions related to time trends (e.g., “How did sales perform over time?”).
  + Ensure the column with dates is in datetime format (pd.to\_datetime()).
  + Group data by time (e.g., monthly, yearly) using groupby().

**5. Data Visualization**

**5.1. Basic Charting (Bar, Line, Pie)**

* **Task**: Implement basic chart generation based on query results.
  + If the user asks for a “top 5 products,” generate a bar chart.
  + If asking about a trend, generate a line chart.
  + Use matplotlib or plotly for creating charts.
  + Display the chart alongside the text result.

**5.2. Visualize Aggregations and Trends**

* **Task**: Visualize the aggregated data and trends.
  + Example: If the user asks about the average salary, return the number and display a simple bar chart with the result.
  + Use plotly or matplotlib to handle charts that update dynamically based on user input.

**6. User Interface (Streamlit)**

**6.1. Build Streamlit Interface**

* **Task**: Set up the basic Streamlit UI.
  + Create file uploader, question input, and output display sections.
  + Use st.file\_uploader() for file upload.
  + Use st.text\_input() or st.text\_area() for user questions.
  + Display results with st.write() and the corresponding chart using st.pyplot() or st.plotly\_chart().

**6.2. Display Results (Text + Visuals)**

* **Task**: Ensure that both text-based answers and charts are shown on the UI.
  + Display the textual answer in a readable format.
  + Show the chart (bar, line, or pie) based on the query type.

**7. Testing & Validation**

**7.1. Unit Testing for NLP**

* **Task**: Write unit tests for the NLP preprocessing and intent detection components.
  + Test that the system correctly identifies intent types.
  + Test that the column mapping works accurately with sample questions.

**7.2. Unit Testing for Query Execution**

* **Task**: Test the core query execution logic (aggregations, filtering, sorting).
  + Verify that the query results match expected outputs using predefined sample datasets.

**7.3. End-to-End Testing**

* **Task**: Test the entire workflow from uploading a CSV to asking a question and receiving results.
  + Use a sample dataset and ensure the system answers correctly (both text and chart).

**8. Deployment**

**8.1. Deploy the Application**

* **Task**: Deploy the application to a cloud service.
  + Use **Streamlit Sharing** or **Heroku** for easy deployment.
  + Ensure all dependencies are listed in requirements.txt.

**8.2. Final Testing on Deployment**

* **Task**: Test the deployed version to ensure it behaves as expected.
  + Perform tests for CSV uploads, question processing, and UI interaction.

**9. Documentation**

**9.1. Document the Project**

* **Task**: Write a comprehensive README.md.
  + Include project setup, installation instructions, and usage examples.
  + Explain the core functionality and the types of questions users can ask.

**9.2. Code Documentation**

* **Task**: Add inline comments and docstrings to all functions and key code sections.
  + Ensure that the code is easy to understand for future developers.

**10. Future Improvements (Optional)**

**10.1. Multi-Turn Conversation (Optional)**

* **Task**: Implement a system to handle follow-up questions (stateful interaction).
  + Track previous questions and use them for context in subsequent queries.

**10.2. Extend NLP with ML (Optional)**

* **Task**: Implement machine learning for better question understanding (e.g., using scikit-learn for intent classification).
  + This can be done after the core system is built.

**Summary of Milestones:**

* **Milestone 1**: Data Upload and Profiling
* **Milestone 2**: NLP Preprocessing and Intent Detection
* **Milestone 3**: Query Execution (Aggregation, Sorting, Filtering)
* **Milestone 4**: Visualization Integration
* **Milestone 5**: UI Development (Streamlit)
* **Milestone 6**: Testing & Validation
* **Milestone 7**: Deployment
* **Milestone 8**: Documentation

These **developer-centric subtasks** will guide you through the process, focusing on technical implementation and ensuring that each component is built step by step. Would you like me to help you start with any of these tasks (e.g., setting up the Streamlit app, NLP preprocessing)?