The project was implemented using the **Scrum methodology**, an agile project management approach that allows the team to adapt flexibly to changes while continuously delivering value. The Scrum team structure, whose roles can be fluid and can be carried out as a single team, included a **Product Owner** (responsible for requirements and prioritization), a **Scrum Master** (supports the effective application of the methodology), and a **Development Team** (responsible for task implementation).

Based on the project's technical components and the Scrum methodology, the project phases can be structured into the following iterative Sprints, with each Sprint delivering functional increments:

**Project Phases (Sprints)**

* **Sprint 1: Foundation – Data Management Backend & Web Interface**
  + **Objective:** To establish the core Flask application, enabling CSV data management via a web interface and data exposure through a RESTful API.
  + **Key Deliverables/Functionality:**
    - **Flask Application Setup:** Initialization of the Flask application and an in-memory Pandas DataFrame to serve as the database. This database resets with each application restart. The required libraries like Flask, request, jsonify, render\_template, and pandas are imported.
    - **Web Interface Implementation:** Creation of the root route (/) to render an HTML template (index.html) for the "CSV Data Manager" interface. The HTML interface includes a title, an upload form, and a section for "Uploaded Data".
    - **CSV Upload Functionality:** Implementation of the /upload POST route to handle file uploads. The application checks if a file is provided and if its extension is .csv. If valid, the CSV data is read using Pandas (pd.read\_csv) and appended to the global database using pd.concat. Success or error messages are returned to the user.
    - **Uploaded Data Display:** The DataFrame is converted into an HTML table using database.to\_html(index=False) for display on the web interface, showing "No data available" if the database is empty.
    - **RESTful API Endpoint for Data Retrieval:** Creation of the /api/daten GET endpoint to serve all stored data in JSON format. The data is specifically formatted as a list of JSON objects (records) using orient='records'. It also handles cases where no data is available, returning a 404 status and a "No data available" message.
    - The Flask development server is started with debug=True and use\_reloader=False.
* **Sprint 2: Predictive Modeling – Data Integration & Model Training**
  + **Objective:** To develop the core machine learning pipeline, integrating with the established data management backend to train a house price prediction model.
  + **Key Deliverables/Functionality:**
    - **API Data Retrieval:** Implementation of code to send a GET request to the /api/daten endpoint (e.g., http://127.0.0.1:5000/api/daten) using the requests library. The retrieved JSON response is then converted into a Pandas DataFrame. Error handling for API calls is also included.
    - **Data Preprocessing:** Missing values are removed from the DataFrame using data.dropna(). For example, the provided "German Property Listings" data shows instances of missing Grundstücksgröße values that would be handled here.
    - **Feature and Target Variable Definition:** The features (X) are defined as 'Grundstücksgröße', 'Zimmeranzahl', and 'Garagenanzahl', while 'Hauspreis' is identified as the target variable (y) for the prediction model.
    - **Data Splitting:** The processed dataset is divided into training and testing sets using train\_test\_split with a test size of 20% and random\_state=42 for reproducibility.
    - **Linear Regression Model Training:** A LinearRegression model is instantiated and trained using the X\_train and y\_train datasets. Relevant libraries such as sklearn.model\_selection.train\_test\_split and sklearn.linear\_model.LinearRegression are imported.
* **Sprint 3: Model Evaluation & Example Predictions**
  + **Objective:** To evaluate the trained model's performance and provide a practical demonstration of its predictive capabilities on new data.
  + **Key Deliverables/Functionality:**
    - **Model Prediction on Test Data:** The trained model is used to make predictions on the X\_test dataset, generating y\_pred.
    - **Model Evaluation (Mean Squared Error):** The Mean Squared Error (MSE) is calculated between the actual test prices (y\_test) and the predicted prices (y\_pred) using mean\_squared\_error. The calculated MSE is then printed, providing a quantitative measure of the model's accuracy. The sklearn.metrics.mean\_squared\_error library is imported for this purpose.
    - **Example Prediction Feature:** Functionality is implemented to make predictions for new, unseen sample data. This involves creating a sample\_data Pandas DataFrame with specific values for 'Grundstücksgröße', 'Zimmeranzahl', and 'Garagenanzahl'. This directly addresses the previous SyntaxError by providing concrete example data for prediction.
    - **Formatted Prediction Output:** The results of the example predictions are displayed in a clear, readable format. This includes showing the input features (e.g., 'Grundstücksgröße', 'Zimmeranzahl', 'Garagenanzahl') and the corresponding estimated house price, formatted to two decimal places.