**Problem-Solving Approach:**

The task was to write a CSV parser that reads a CSV file and provides access to individual rows and fields. A key requirement was to handle complex CSV features, such as quoted fields (which might contain commas or escaped double quotes). To solve this:

1. **File Reading**: I used a StreamReader to read the CSV file line by line. This allows efficient memory usage, especially for large files.
2. **Handling Quoted Fields**: The major challenge was correctly parsing fields enclosed in double quotes, which may contain commas or even escaped quotes. I handled this by tracking whether the parser is "inside quotes" while processing each line.
3. **Row Storage**: I stored each row as a Dictionary<string, string>, where the keys are the column headers, and the values are the fields in that row. This allows easy access to any field by column name.
4. **Edge Case Handling**:
   * Rows with fewer fields than headers are padded with empty strings.
   * Commas inside quoted fields are correctly handled.
   * Escaped double quotes (represented by "" inside quoted fields) are converted into single quotes.

**Testing Strategy:**

To ensure the parser works correctly, I developed a series of unit tests using the **xUnit** framework. The following scenarios were tested:

1. **Basic Parsing**: I tested a simple CSV file with no quotes or commas inside fields, ensuring that the parser could correctly separate rows and columns.
2. **Quoted Fields with Commas**: I created a CSV where some fields contained commas inside double quotes. This tested whether the parser correctly handled commas within quoted fields without treating them as field delimiters.
3. **Quoted Fields with Escaped Quotes**: I created a CSV where fields contained escaped quotes ("") inside quoted fields. This tested whether the parser correctly handled double quotes inside fields.
4. **Missing Fields**: I tested cases where some rows had fewer fields than the number of headers. The parser should handle these by inserting empty strings in place of missing fields.
5. **Error Handling**: I tested cases where invalid column names or row indices were requested to ensure the program throws appropriate exceptions.

Each test confirmed that the parser handled the expected scenarios, and exceptions were raised where necessary.

**Testing Results:**

* The parser successfully handled standard CSV files with and without quoted fields.
* The parser correctly parsed fields containing commas and escaped double quotes.
* Edge cases like missing fields and invalid column access were handled gracefully.
* The test cases all passed, confirming the correctness and robustness of the solution.