my best complete final answer to the task.  
  
# \*\*Business Requirements Document (BRD)\*\*  
  
## \*\*1. Document Information\*\*  
- \*\*Document Title\*\*: Calculator Flask Application BRD  
- \*\*Date\*\*: September 4, 2024  
- \*\*Author\*\*: [Your Name]  
- \*\*Version\*\*: 1.0  
  
## \*\*2. Executive Summary\*\*  
### \*\*2.1. Purpose\*\*  
The purpose of this document is to outline the business requirements for developing a simple web-based calculator application that performs basic arithmetic operations. This document aims to ensure that all stakeholders have a clear understanding of the project scope, objectives, and requirements.  
  
### \*\*2.2. Project Background\*\*  
Tapestry requires a web-based calculator application to perform basic arithmetic operations such as addition, subtraction, multiplication, and division. The application should be user-friendly and accessible via a REST API for easy integration with other systems. The project aims to provide a reliable tool for end-users to perform basic calculations online.  
  
### \*\*2.3. Scope\*\*  
The scope of this project includes:  
- Development of a web-based calculator application.  
- Implementation of basic arithmetic operations.  
- Error handling for edge cases like division by zero.  
- Deployment guidance on Azure.  
- Exclusions: Advanced mathematical functions, complex UI design, and non-arithmetic operations.  
  
## \*\*3. Business Objectives\*\*  
### \*\*3.1. Goals\*\*  
- Develop a web application that performs addition, subtraction, multiplication, and division.  
- Ensure robust error handling, particularly for division by zero.  
- Provide a simple and clean user interface.  
- Deliver the application as a REST API for ease of integration.  
- Deploy the application on Azure with guidance provided to Tapestry's internal team.  
  
### \*\*3.2. Success Criteria\*\*  
- Successful implementation of all four arithmetic operations.  
- Effective error handling for division by zero.  
- User-friendly interface.  
- REST API endpoints functioning correctly.  
- Successful deployment on Azure.  
  
## \*\*4. Stakeholders\*\*  
### \*\*4.1. Stakeholder List\*\*  
  
|\*\*Name\*\*|\*\*Role\*\*|\*\*Contact Information\*\*|  
| :-: | :-: | :-: |  
|Jane Doe|Project Manager, Tapestry|janedoe@tapestry.com|  
|John Smith|Lead Developer, Systems Ltd|johnsmith@systemsltd.com|  
  
## \*\*5. Requirements\*\*  
### \*\*5.1. Functional Requirements\*\*  
  
|\*\*Requirement ID\*\*|\*\*Description\*\*|\*\*Priority\*\*|\*\*Acceptance Criteria\*\*|  
| :-: | :-: | :-: | :-: |  
|FR-1|Addition Operation|High|Endpoint to add two numbers and return the result in JSON format|  
|FR-2|Subtraction Operation|High|Endpoint to subtract one number from another and return the result in JSON format|  
|FR-3|Multiplication Operation|High|Endpoint to multiply two numbers and return the result in JSON format|  
|FR-4|Division Operation|High|Endpoint to divide one number by another and return the result in JSON format, with error handling for division by zero|  
|FR-5|Error Handling|High|Return clear and descriptive error messages for invalid inputs and division by zero|  
|FR-6|Input Validation|High|Ensure only valid numerical inputs are accepted|  
  
### \*\*5.2. Non-Functional Requirements\*\*  
  
|\*\*Requirement ID\*\*|\*\*Description\*\*|\*\*Priority\*\*|\*\*Acceptance Criteria\*\*|  
| :-: | :-: | :-: | :-: |  
|NFR-1|Usability|High|Simple and intuitive user interface|  
|NFR-2|Performance|High|Process requests and return results promptly|  
|NFR-3|Security|High|Follow best practices in web security|  
|NFR-4|Scalability|Medium|Handle a growing number of users without significant performance degradation|  
  
## \*\*6. Assumptions and Constraints\*\*  
### \*\*6.1. Assumptions\*\*  
- The application will be developed using Python and Flask.  
- The application will be deployed on Azure.  
- Tapestry's internal team has some familiarity with Flask for future maintenance.  
  
### \*\*6.2. Constraints\*\*  
- The project must be completed within 6 weeks.  
- Budget constraints as outlined in the RFP.  
- Limited to basic arithmetic operations.  
  
## \*\*7. Risks and Mitigation\*\*  
### \*\*7.1. Identified Risks\*\*  
  
|\*\*Risk ID\*\*|\*\*Description\*\*|\*\*Impact\*\*|\*\*Probability\*\*|\*\*Mitigation Strategy\*\*|  
| :-: | :-: | :-: | :-: | :-: |  
|R-1|Delay in development|High|Medium|Regular progress reviews and agile development practices|  
|R-2|Issues with Azure deployment|Medium|Low|Provide detailed deployment guide and support|  
|R-3|Security vulnerabilities|High|Medium|Conduct thorough security testing and follow best practices|  
  
## \*\*8. Dependencies\*\*  
### \*\*8.1. Dependencies\*\*  
- Availability of Tapestry's internal resources for deployment.  
- Access to Azure for deployment.  
  
## \*\*9. Glossary\*\*  
- \*\*API\*\*: Application Programming Interface  
- \*\*BRD\*\*: Business Requirements Document  
- \*\*Flask\*\*: A micro web framework written in Python  
- \*\*JSON\*\*: JavaScript Object Notation  
- \*\*REST\*\*: Representational State Transfer  
- \*\*WSGI\*\*: Web Server Gateway Interface  
  
## \*\*10. Technical Feasibility and Clarifications\*\*  
### \*\*10.1. Technical Feasibility\*\*  
Upon reviewing the documented requirements, the following points have been identified to ensure technical feasibility:  
- \*\*Technology Stack\*\*: The use of Python and Flask is appropriate for developing a web-based calculator application. Flask is lightweight and suitable for creating RESTful APIs.  
- \*\*Deployment on Azure\*\*: Azure provides robust support for Python applications, and deployment can be streamlined using Azure App Services or Azure Functions.  
- \*\*Error Handling and Input Validation\*\*: Implementing error handling for division by zero and input validation is feasible using Python's exception handling mechanisms and Flask's request validation features.  
  
### \*\*10.2. Clarifications and Simplifications\*\*  
- \*\*REST API\*\*: For non-technical stakeholders, REST API can be described as a way for different software applications to communicate with each other over the internet using simple web requests.  
- \*\*JSON Format\*\*: JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.  
  
### \*\*10.3. Potential Technical Hurdles and Mitigation\*\*  
- \*\*Scalability\*\*: While the initial scope is limited to basic arithmetic operations, future scalability should be considered. Using a microservices architecture can help in scaling individual components independently.  
 - \*\*Mitigation\*\*: Design the application with modularity in mind, allowing for easy addition of new features or scaling of existing ones.  
- \*\*Security\*\*: Ensuring the application is secure from common web vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).  
 - \*\*Mitigation\*\*: Follow Flask's security best practices, use secure coding guidelines, and conduct regular security audits.  
  
## \*\*11. Project Name\*\*  
### \*\*11.1. Suggested Project Name\*\*  
- \*\*Project Calcify\*\*: This name captures the essence of the initiative, focusing on the core functionality of performing calculations while being simple and memorable for all stakeholders.  
  
This BRD accurately captures the project requirements and stakeholder needs as discussed in the meeting transcript and outlined in the RFP document.