Development of a Web-Based Advanced Proxy Modeling Platform

Course: ENG 573 – Capstone Project
Semester: Fall 2025 Date: December 3, 2025

Submitted By:

Hung-Jui Chen (UIN: 662551968)
Guanghao Xu (UIN: 661599403)

Project Sponsor: Ke Li, SLB (Schlumberger)

Abstract

Early-stage engineering design is often slowed down by the reliance on slow, high-fidelity models. FEA tools are accurate, but they are slow and hard to use. This bottlenecks engineers who need to iterate on designs quickly. This makes it difficult for engineers to iterate on designs quickly. Working with SLB, we built a web-based platform that lets field engineers run validated physics models in their browser, removing the need to install heavy simulation software.

We built the platform using a standard three-tier architecture: a vanilla JavaScript frontend, a FastAPI backend for high performance, and a MySQL database The core deliverable is a functional Pressure Vessel Calculator capable of performing complex structural analysis in under 0.1 seconds. Beyond standard forward analysis (calculating Safety Factors), the platform features advanced Inverse Calculation Algorithms utilizing binary search logic to solve for maximum safe loads and optimal geometries. Key features include real-time "debounced" computation, dynamic unit conversion (SI/Imperial/Metric), secure JWT authentication, and server-side graphical rendering of failure envelopes.

This report details the theoretical basis of the physics engine, the software architecture design, the algorithmic implementation of the inverse solvers, and the validation results comparing the proxy model against theoretical benchmarks. The final deployed application successfully meets all sponsor requirements, providing a scalable foundation for future engineering tools at SLB.

1. Introduction

1.1 Background and Problem Statement

In mechanical engineering, the speed of the design cycle depends heavily on how fast we can verify a design. Traditionally, engineers rely on High-Fidelity Models, such as Computational Fluid Dynamics (CFD) or Finite Element Analysis (FEA), to verify the integrity of a design [1]. While these tools provide exceptional accuracy, they come with significant costs:

1. **Time Latency:** Non-linear FEA simulations can take hours or days to finish.
2. **Resource Intensity:** These models often require high-performance computing (HPC) clusters and expensive software licenses (e.g., Abaqus, Ansys).
3. **Expertise Barrier:** Setting up high-fidelity simulations requires specialized knowledge of meshing, boundary conditions, and solver convergence, often limiting their use to dedicated simulation teams.

This slows down the Conceptual Design Phase, which is exactly when engineers need to test hundreds of variations quickly. The industry needs Proxy Models: simplified representations of high-fidelity systems that give fast feedback while staying accurate enough for design work.

1.2 Project Objectives

The primary goal of this Capstone project was to develop a centralized, web-based platform to host these proxy models. SLB specified the following critical requirements [1]:

- **Accessibility:** The tool must be web-based and cloud-deployed, removing the need for local software installation.
- **Performance:** The system must achieve a computation latency of **< 0.1 seconds** to enable a seamless "design-by-tuning" experience.
- **Scalability:** The architecture must support the future addition of 100+ different calculators (e.g., beams, plates, shells) and up to 1,000 concurrent users.
- **Functionality:** A Proof-of-Concept (POC) calculator for **Pressure Vessels** must be implemented, capable of solving for stress, deformation, and limiting loads (inverse problems).

1.3 Team Roles and Collaboration

The project was executed by a two-person team, adopting an Agile development methodology with weekly sprints [2].

- **Hung-Jui Chen (Frontend Lead):** Responsible for the client-side architecture. Key contributions include the responsive UI design using CSS, the implementation of the JavaScript logic for real-time API interaction, dynamic unit conversion algorithms, and PDF report generation.
- **Guanghao Xu (Backend Lead):** Responsible for the server-side logic and infrastructure. Key contributions include the FastAPI system architecture, the implementation of the Roark's Formulas physics engine, the development of the numerical binary search algorithms for inverse calculations, and the database schema design.

2. Literature Review and Theoretical Framework

2.1 Physics Domain: Pressure Vessel Mechanics

The core functionality of the platform is grounded in solid mechanics, specifically the analysis of thick-walled pressure vessels. The calculations implemented in the backend are derived from *Roark's Formulas for Stress and Strain* (7th Edition) [3].

2.1.1 Principal Stresses

For a thick-walled cylindrical vessel subjected to internal pressure (P_i), external pressure (P_o), and an axial load (F), the stress state at any radius r is defined by three principal stresses. The platform calculates these at the inner radius ($r = b$), where stresses are typically highest and failure is most likely to initiate.

- Hoop Stress (σ_t): The stress acting tangentially to the circumference.

$$\sigma_t = \frac{P_i b^2 - P_o a^2}{a^2 - b^2} + \frac{(P_i - P_o)a^2 b^2}{(a^2 - b^2)b^2}$$

- Radial Stress (σ_r): The stress acting normal to the cylinder wall. At the inner surface, this simplifies to the internal pressure:

$$\sigma_r = -P_i$$

- Axial Stress (σ_z): The longitudinal stress, which includes components from pressure on the end-caps and external axial loads.

$$\sigma_z = \frac{P_i b^2 - P_o a^2}{a^2 - b^2} + \frac{F}{A_{cross}}$$

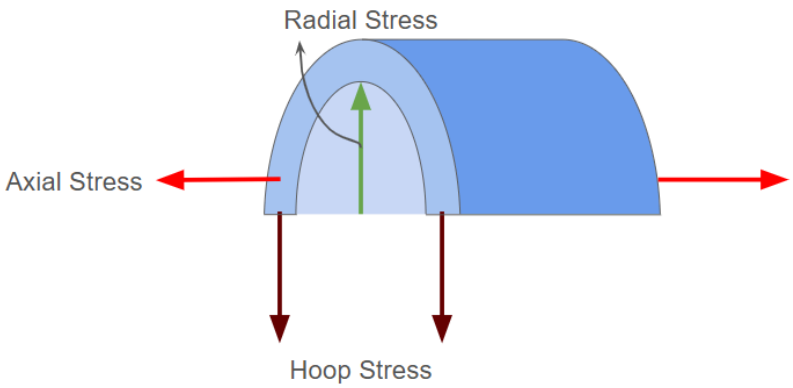

Figure 1: Diagram illustrating the Hoop, Radial, and Axial stress components acting on a differential element of a pressure vessel.

2.1.2 Failure Criteria: Von Mises Stress

To predict yielding in ductile materials (like steel), the platform utilizes the Von Mises Yield Criterion (Distortion Energy Theory). This theory states that yielding occurs when the distortion energy per unit volume exceeds that of a simple tension test specimen at yield. The equivalent Von Mises stress (σ_{vm}) is calculated as:

$$\sigma_{vm} = \frac{1}{\sqrt{2}}\sqrt{(\sigma_t - \sigma_r)^2 + (\sigma_r - \sigma_z)^2 + (\sigma_z - \sigma_t)^2}$$

2.1.3 Safety Factor Calculation

The final metric provided to the user is the Safety Factor (SF), defined as the ratio of the material's effective yield strength to the calculated Von Mises stress. We also incorporate a Temperature Correction Factor (C_T) to account for material weakening at high operating temperatures:

$$SF = \frac{\sigma_{yield} \times C_T}{\sigma_{vm}}$$

2.1.4 Deformation and Strain

In addition to failure via yielding, the platform calculates the elastic deformation of the vessel under combined loads. The radial and longitudinal deformations are derived from Roark's

Formulas for Stress and Strain. The platform calculates these values based on the specific pressure conditions (Internal vs. External) combined with the axial load.

Variable Definitions:

- $a = \text{Outer Radius}(OD/2)$
- $b = \text{Inner Radius}(ID/2)$
- $c = \text{Radius of head opening}(HOD/2)$
- $P = \text{Pressure}$
- $F = \text{Axial Load}$
- $E = \text{Young's Modulus}$
- $\nu = \text{Poisson's Ratio}$

A. Internal Pressure Case ($P = P_{internal}$)

When the vessel is subjected to internal pressure and an axial load, the deformations are calculated as follows:

- Change in Outer Diameter (ΔOD):
$$\Delta OD = \frac{2a}{E(a^2 - b^2)} \left[2Pb^2 - \nu P(b^2 - c^2) - \frac{\nu F}{\pi} \right]$$
- Change in Inner Diameter (ΔID):
$$\Delta ID = \frac{2b}{E(a^2 - b^2)} \left\{ P[a^2(1+\nu) + b^2(1-\nu)] - \nu P(b^2 - c^2) - \frac{\nu F}{\pi} \right\}$$
- Change in Length (ΔL):
$$\Delta L = \frac{L}{E(a^2 - b^2)} \left[-2\nu Pb^2 + P(b^2 - c^2) + \frac{F}{\pi} \right]$$

B. External Pressure Case ($P = P_{external}$)

When the vessel is subjected to external pressure and an axial load, the deformations are calculated as follows:

- Change in Outer Diameter (ΔOD):
$$\Delta OD = \frac{2a}{E(a^2 - b^2)} \left\{ -P[a^2(1-\nu) + b^2(1+\nu)] + \nu P(a^2 - c^2) - \frac{\nu F}{\pi} \right\}$$
- Change in Inner Diameter (ΔID):
$$\Delta ID = \frac{2b}{E(a^2 - b^2)} \left[-2Pa^2 + \nu P(a^2 - c^2) - \frac{\nu F}{\pi} \right]$$
- Change in Length (ΔL):
$$\Delta L = \frac{L}{E(a^2 - b^2)} \left[2\nu Pa^2 - P(a^2 - c^2) + \frac{F}{\pi} \right]$$

2.2 Software Architecture Patterns

We needed a response time under 0.1s and the ability to scale, so we compared a few different architecture options.

- **Backend: FastAPI vs. Flask:** We chose FastAPI over Flask specifically for its async (ASGI) capabilities. Since we need <100ms latency, we couldn't afford the blocking requests common in synchronous frameworks. [4].
- **Frontend: Vanilla JS vs. React:** We chose Vanilla JavaScript (ES6+) because React was overkill. React requires downloading a large bundle (2MB+), whereas we wanted to keep the payload light. We used native DOM manipulation to keep the frontend payload under 50KB, which helps the page load instantly.

3. Methodology: System Architecture

3.1 High-Level Architecture

The platform is built on a standard **Three-Tier Architecture [2]**:

1. **Presentation Layer (Client):** The browser runs the HTML/CSS/JS application. It handles user input, validation, and rendering.
2. **Application Layer (API):** Hosted on Azure App Service, this layer runs the Python FastAPI server. It contains the business logic (physics engine) and manages authentication.
3. **Data Layer (Storage):** A managed MySQL database stores user credentials and calculation history.

3.2 Database Schema Design

We used SQLAlchemy ORM to design the database. It uses three main tables to handle user accounts and save calculation history.

- **Users Table:** Stores id, email, password_hash, and role.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
email	varchar(255)	NO	UNI	NULL	
password_hash	varchar(255)	NO		NULL	
created_at	datetime	YES		now()	DEFAULT_GENERATED
role	varchar(50)	NO		user	

Table 1: Database schema of users table

- **Calculations Table:** Stores calculation history. It has a Foreign Key relationship with the Users table (user_id), ensuring that if a user is deleted, their data is cascaded (removed).

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
user_id	int	NO	MUL	NULL	
mode	enum	NO		NULL	
outer_diameter	float	NO		NULL	
inner_diameter	float	NO		NULL	
head_opening_diameter	float	NO		NULL	
length	float	NO		0	
internal_pressure	float	NO		NULL	
external_pressure	float	NO		NULL	
axial_load	float	NO		NULL	
bending_moment	float	NO		0	
torque	float	NO		0	
yield_strength	float	NO		NULL	
poissons_ratio	float	NO		NULL	
youngs_modulus	float	NO		NULL	
temp_correction_factor	float	NO		NULL	
material_name	varchar(128)	YES		NULL	
calculation_mode	varchar(64)	YES		NULL	
target_safety_factor	float	YES		NULL	
hoop_stress	float	YES		NULL	
radial_stress	float	YES		NULL	
axial_stress	float	YES		NULL	
von_mises_stress	float	YES		NULL	
safety_factor	float	YES		NULL	
target_pressure	float	YES		NULL	
delta_od	float	YES		NULL	
delta_id	float	YES		NULL	
delta_l	float	YES		NULL	
min_safe_od	float	YES		NULL	
max_safe_id	float	YES		NULL	
max_safe_axial_load	json	YES		NULL	
created_at	datetime	YES		now()	DEFAULT_GENERATED

Table 2: Database schema of calculations table

- **Materials Table:** A lookup table for standard material properties, allowing the frontend to dynamically populate dropdown menus.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
key	varchar(50)	NO	UNI	NULL	
display_name	varchar(100)	NO		NULL	
yield_strength	float	NO		NULL	
youngs_modulus	float	NO		NULL	
poissons_ratio	float	NO		NULL	
temp_correction_factor	float	NO		NULL	
created_at	datetime	YES		now()	DEFAULT_GENERATED

Table 3: Database schema of Materials table

3.3 Security Architecture

Because the engineering data is proprietary, we had to make sure the app was secure.

1. **JWT Authentication:** We implemented a stateless authentication mechanism using **JSON Web Tokens (JWT)**. When a user logs in, the server signs a token containing the user_id and role using a secret key (HS256 algorithm). This token must be included in the Authorization header of all subsequent API requests.
2. **Password Hashing:** Passwords are never stored in plain text. We utilize **Bcrypt** (via the passlib library) to salt and hash passwords. We used Bcrypt because its high computational cost makes it resistant to brute-force attacks [5].

4. Implementation Details

4.1 Computational Engine: Forward Calculation

The core physics logic is encapsulated in the VesselCalculator class. The method calculate_stresses serves as the foundation for all other modules.

```python
def calculate_stresses(self, Pi: Optional[float] = None, Po: Optional[float] = None,
                       F:Optional[float] = None, a: Optional[float] = None,
                       b: Optional[float] = None) -> tuple[float, float, float]:

    # Lame constants at inner radius (r=b), outer radius (R=a)
    A_term = (Pi * b**2 - Po * a**2) / (a**2 - b**2)
    B_term = (Pi - Po) * (a**2 * b**2) / (a**2 - b**2)

    s_radial = A_term - (B_term / b**2)          # σ_r(b)
    s_hoop   = A_term + (B_term / b**2)          # σ_t(b)

    # axial stress with head opening radius c
    # split internal/external parts to match Roark
    s_axial_internal =  Pi * (b**2 - self.c**2) / (a**2 - b**2)
    s_axial_external = -Po * (a**2 - self.c**2) / (a**2 - b**2)
    s_axial_pressure = s_axial_internal + s_axial_external
    cross_sectional_area = math.pi * (a**2 - b**2)
    s_axial = s_axial_pressure + (F / cross_sectional_area)

    return s_hoop, s_radial, s_axial
```

Code Snippet 1: Core Stress Calculation Logic (Python)

4.2 Inverse Calculation Algorithms

A core feature of the platform is the ability to solve "Inverse Problems"—determining the limiting geometry or loads for a specified Safety Factor (SF_{target}). We used two different algorithms depending on what variable we were solving for: a Numerical Binary Search or an Analytical Quadratic Inversion.

4.2.1 Numerical Solution: Maximum Pressure and Optimal Geometry

For variables that appear non-linearly in the Von Mises equations—specifically Pressure (P), Outer Diameter (OD), and Inner Diameter (ID)—an analytical solution is mathematically intractable due to the complex interaction of geometric terms (a^2, b^2) in the stress formulas.

To solve for these, we implemented a Binary Search Algorithm ($O(\log N)$).

- Maximum Safe Pressure: The algorithm iterates between $P = 0$ and a theoretical maximum, converging on the pressure where the calculated $SF \approx SF_{target}$.
- Minimum Safe Outer Diameter (OD): This mode calculates the thinnest possible wall thickness for a given internal diameter and load. The search space ranges from the Inner Diameter (ID) to a large upper bound ($10 \times ID$).
- Maximum Safe Inner Diameter (ID): This mode calculates how much material can be removed from the inside of the vessel (e.g., for flow assurance) without causing failure. The search operates between the Head Opening Diameter (HOD) and the Outer Diameter (OD).

```
if SF_lo >= target_SF:
    while SF_hi >= target_SF and hi < 1e9:
        hi *= 2.0
        SF_hi = SF_of(hi)
    if hi >= 1e9 and SF_hi >= target_SF:
        return VesselCalculationResponse(target_pressure=-1.0)

    best = lo
    for _ in range(60):
        mid = 0.5 * (lo + hi)
        if SF_of(mid) >= target_SF:
            best = mid
            lo = mid
        else:
            hi = mid

    # --- Compute deformation at P = best ---
    P_apply = best
    if mode == "internal":
        self.Pi = P_apply
        self.Po = 0.0
    else:
        self.Po = P_apply
        self.Pi = 0.0

    ΔOD, ΔID, ΔL = self.deformation_changes()
    return VesselCalculationResponse(
        target_pressure=P_apply,
        delta_od=ΔOD,
        delta_id=ΔID,
        delta_l=ΔL
    )
```

Code Snippet 2: Binary Search Implementation for Inverse Solving

4.2.2 Analytical Solution: Maximum Safe Axial Load

Unlike pressure or geometry, the Axial Load (F) has a simpler relationship with the Von Mises stress. The stress state can be expressed as a quadratic equation in terms of F. Therefore, instead of iteratively searching, we solve the quadratic discriminant directly to find the exact limiting forces for both Tension and Compression.

$$F = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

This analytical approach is computationally more efficient ($O(1)$) than binary search, providing instant bounds for the allowable axial force.

4.3 Frontend Implementation

4.3.1 Real-Time "Debouncing"

We wanted results to update as the user types, but sending an API request on every keystroke flooded the backend and slowed everything down. Therefore, we implemented a Debounce pattern in JavaScript.

We wrote a debounce function that resets a timer every time the user types. The API call only fires once the user stops typing for 300ms.

```
function debounce(fn, delay = 300) {
  let t;
  return (...args) => {
    clearTimeout(t);
    t = setTimeout(() => fn(...args), delay);
  };
}
//Usage: const debounced = debounce(handlePressureVesselCalculation, 300);
```

Code Snippet 3: Real-time calculation for debouncing function

4.3.2 Unit Conversion System

The backend only uses SI Units (Meters, Pascals, Newtons), so the frontend handles all the conversions. We implemented a robust state management system that tracks the currentUnitSystem. When a user switches from "Metric" to "Imperial":

1. **Iterate:** Loop through all input fields.
2. **Convert:** Apply transformation factors (e.g., value * 0.03937 for mm to inches).
3. Update DOM: Replace input values and update all labels (e.g., change "MPa" to "psi"). This ensures that the user can switch contexts seamlessly without losing their data.

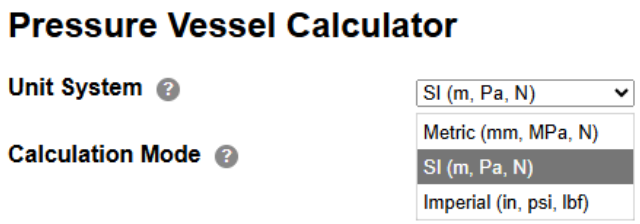

Figure 2: The Unit Conversion interface allowing users to switch between SI, Metric, and Imperial systems.

5. Results and Validation

5.1 Performance Validation

The primary Key Performance Indicator (KPI) for this project was a total response time of less than 0.1 seconds (100ms). We conducted performance testing using the browser's Network Monitor tools on the deployed Azure instance.

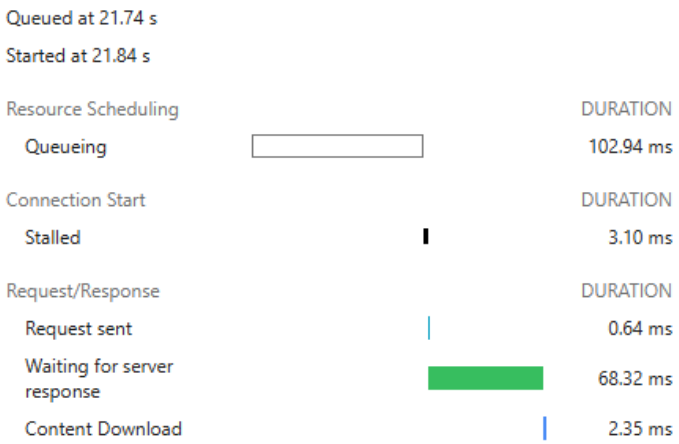

Figure 3: Performance profiling of the "preview-safety-factor" endpoint.

As shown in Figure 3, the green bar indicates a Time to First Byte (TTFB) of 68.32 ms, which includes both the network round-trip to Azure and the Python backend computation. The total latency was about 71 ms, which is well under our 0.1s target

5.2 Accuracy Verification

To ensure engineering validity, we performed Unit Testing comparing the App's output against hand calculations.

Test Case 1: Standard Pressure Vessel

- Inputs: OD=1.0m, ID=0.8m, Internal Pressure=5 MPa, Yield Strength=250 MPa.
- Hand Calculation:
  - $a = 0.5, b = 0.4$
  - Hoop Stress $\sigma_t \approx 22.77$ MPa
  - Radial Stress $\sigma_r = -5.0$ MPa
  - Axial Stress $\sigma_z \approx 8.88$ MPa
  - Von Mises ≈ 24.05 MPa
  - Safety Factor ≈ 10.39
- App Result: 10.39193893
- Status: Valid.

Test Case 2: Inverse Calculation Consistency

To validate the binary search logic, we took the result of Test Case 1 (SF = 10.39) and input it into the "Max Safe Pressure" calculator.

- Input: Target SF = 10.3919...
- Expected Output: Max Pressure = 5,000,000 Pa (5 MPa).
- App Result: 5,000,000.16 Pa.
- Error: $< 0.0001\%$.
- Status: Valid.

```
test_consistency_with_forward_Internal_calculation (test_maximum_safe_pressure.TestMaxSafePressureCalculation)
Tests if the inverse calculation can correctly find the pressure ... ok
test_consistency_with_forward_combine_calculation (test_maximum_safe_pressure.TestMaxSafePressureCalculation) ... ok
test_consistency_with_forward_combine_calculation_2 (test_maximum_safe_pressure.TestMaxSafePressureCalculation) ... ok
test_consistency_with_forward_external_calculation (test_maximum_safe_pressure.TestMaxSafePressureCalculation) ... ok
test_lower_safety_factor_allows_higher_pressure (test_maximum_safe_pressure.TestMaxSafePressureCalculation)
Tests that requiring a lower (less safe) safety factor ... ok
test_raises_error_for_invalid_safety_factor (test_maximum_safe_pressure.TestMaxSafePressureCalculation)
Tests that the function correctly raises a ValueError if the ... ok
test_external_only_pressure_monotonic (test_safety_factor.TestSafetyFactorCalculation) ... ok
test_increasing_internal_pressure_lowers_sf (test_safety_factor.TestSafetyFactorCalculation) ... ok
test_internal_and_external_both_nonzero (test_safety_factor.TestSafetyFactorCalculation) ... ok
test_numeric_correctness_against_known_values (test_safety_factor.TestSafetyFactorCalculation)
Case: ... ok
test_returns_response_model_and_fields (test_safety_factor.TestSafetyFactorCalculation) ... ok
test_temp_correction_scales_safety_factor_linearly (test_safety_factor.TestSafetyFactorCalculation) ... ok
test_zero_load_zero_pressure_gives_infinite_sf (test_safety_factor.TestSafetyFactorCalculation) ... ok

----------------------------------------------------------------------
Ran 13 tests in 0.003s

OK
[Logs from Test Case 2: Inverse Calculation Consistency]
Calculated max safe internal pressure: target_pressure=5000000.000180779 Pa

[Logs from Test Case 1: Standard Pressure Vessel]
hoop_stress=27777777.78 radial_stress=-5000000.0 axial_stress=8687010.98
von_mises_stress=24057108.27 safety_factor=10.39193893
```

Figure 4: Terminal output showing successful execution of Python unit tests

5.3 Graphical Visualization Results

The platform successfully renders "Iso-Safety Curves" (Failure Envelopes). These plots show the relationship between Axial Force (X-axis) and Pressure (Y-axis) for a constant Safety Factor. The plots show the Von Mises ellipse, giving engineers a quick way to see the safe operating region.

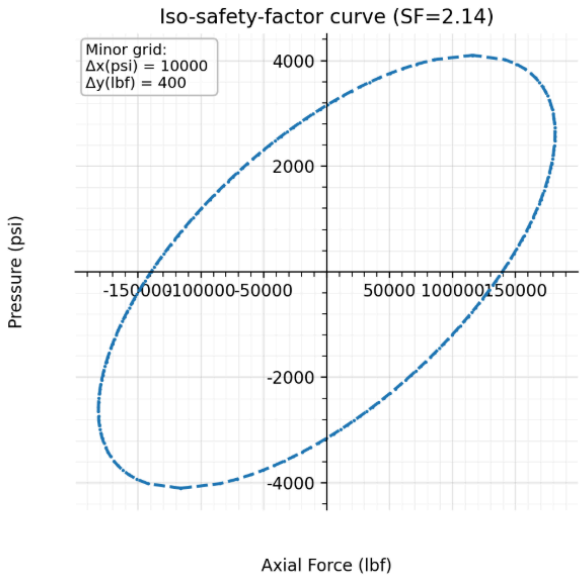

Figure 5: An Iso-Safety Curve generated by the platform. The region inside the ellipse represents safe operating conditions, while the region outside represents failure.

5.4 Deployment Results

The application was successfully deployed to **Microsoft Azure App Service** using a Continuous Deployment (CD) pipeline [6].

1. **Source Control:** Code pushed to GitHub.
2. **Build:** Azure detects the Python environment and installs dependencies from requirements.txt.

3. **Deploy:** The Gunicorn server is started, serving the FastAPI application.
4. **Database:** An Azure Database for MySQL instance was provisioned and linked via environment variables.

6. User Guide and Operations

6.1 User Registration and Login

The platform is protected behind a login screen. New users must register with an email and password. Upon registration, the password is hashed, and the user is redirected to the login page. A "Forgot Password" flow allows users to request a secure reset token, which mimics a standard email-recovery workflow.

6.2 Workflow: Running a Calculation

1. **Select Calculator:** From the Landing Page, select "Pressure Vessel Calculator".
2. **Unit Selection:** Choose preferred units (e.g., Metric) from the top dropdown.
3. **Mode Selection:** The user can select from five distinct calculation modes:
    a. Safety Factor (Forward): Standard analysis.
    b. Max Safe Pressure: Solves for P_{int} or P_{ext}.
    c. Min Safe OD: Optimizes outer shell thickness.
    d. Max Safe ID: Optimizes internal capacity.
    e. Max Safe Axial Load: Determines tension/compression limits.
4. **Data Entry:** Enter dimensions and material properties. The results panel will update automatically (after a 300ms pause).
5. **Visualization:** Enter a "Test Point" (Force and Pressure) to plot a red marker on the failure envelope graph, visually verifying if the load case is safe.
6. **Export:** Click "Export PDF" to generate a downloadable report containing all inputs, outputs, and the graph.

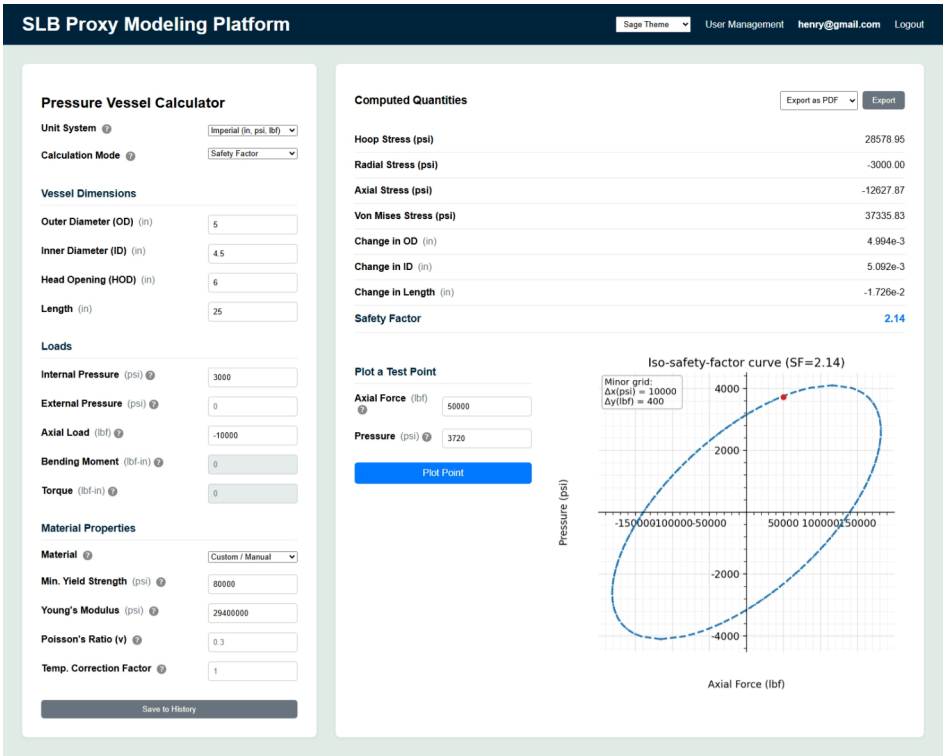

Figure 6: The main Calculator Interface. The left panel contains input fields, the center panel displays numerical results, and the right panel displays the failure envelope graph.

7. Discussion

7.1 Standardization of Engineering Workflows

This platform solves a real problem for SLB. Right now, engineers use scattered Excel sheets or random scripts to perform these proxy calculations. This leads to version control issues and potential errors. This centralized platform ensures that every engineer, regardless of location, uses the exact same validated physics engine.

7.2 Limitations

- **Geometric Simplification:** The current model assumes a perfect cylinder. It does not account for stress concentrations at nozzle junctions or weld defects, which would require full FEA.
- **Material Behavior:** The model assumes linear-elastic, isotropic material behavior. It does not model plasticity (post-yield behavior) or creep.

8. Future Work

8.1 Expansion of Calculator Library

The modular architecture allows for the easy addition of new calculators. The next phase of development should focus on:

- **Beam Deflection:** Calculators for various beam cross-sections and load conditions.
- **Buckling Analysis:** Euler buckling calculators for columns and shells.
- **Fatigue Analysis:** Miners rule implementation for cyclic loading.

8.2 FEA Integration

To bridge the gap between proxy and high-fidelity, the platform could be integrated with commercial FEA APIs (e.g., Ansys PyAnsys or Abaqus Python). This would allow the platform to serve as a "frontend" that can trigger remote high-fidelity jobs for final verification, creating a seamless workflow from concept to validation.

8.3 Advanced User Management

Currently, all registered users have equal access. Future iterations should implement:

- **Teams/Groups:** Allow sharing of calculation history within specific engineering teams.
- **Admin Analytics:** A dashboard for administrators to track which calculators are most popular, helping guide future development resources.

9. Conclusion

Our final application meets all the core requirements set by our sponsor, SLB. We have delivered a secure, cloud-deployed, and high-performance engineering tool that significantly reduces the time required for early-stage design verification.

Using a modern tech stack (FastAPI, JavaScript, Azure), we got the latency down to about 60ms— faster than our 0.1s goal. By implementing the inverse calculation algorithms, we turned the tool from a simple safety checker into a useful design utility. The project stands as a verified Proof-of-Concept, ready for scaling into a comprehensive engineering suite.

References

[1] SLB, "Development of a Web-Based Advanced Proxy Modeling Platform Project," Internal Project Description, Fall 2025.

[2] H. Chen and G. Xu, "Capstone Project Plan: SLB - Web-Based Advanced Proxy Modeling Platform," Internal Team Document, Fall 2025.

[3] W. C. Young and R. G. Budynas, *Roark's Formulas for Stress and Strain*, 7th ed. New York: McGraw-Hill, 2002.

[4] S. Tiwary, "FastAPI," [Online]. Available: https://fastapi.tiangolo.com/. Accessed: Oct. 28, 2025.

[5] "JSON Web Tokens Introduction," [Online]. Available: https://jwt.io/introduction. Accessed: Nov. 10, 2025.

[6] "Microsoft Azure App Service Documentation," [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/. Accessed: Nov. 15, 2025.