SAFAPAC: System Architecture & Technical Implementation Documentation

Document Version 1.0

Software Version 4.0 Alpha

November 2025

(**This document covers the development of the SAFAPAC platform from July to November 2025.)*

AMIC AEROSPACE MALAYSIA INNOVATION CENTRE

AIRBUS

LIST OF FIGURES

LIST OF TABLES

1. Introduction & Project Summary

1.1 Report Objectives

This technical report serves to demonstrate the successful and current deployment of the SAFAPAC (Sustainable Aviation Fuel Analysis Platform and Cost Calculator) web application. The primary objective is to provide all relevant stakeholders with a comprehensive, detailed overview of the platform's recent development progress, highlighting that all key technical milestones have been achieved and are fully aligned with the initial specifications provided by the research team. This ensures the deployed system meets all required technical and functional criteria.

The report specifically aims to showcase the fidelity of the implemented system, demonstrating how it accurately and reliably mirrors the complex techno-economic analysis models originally developed by our research team. Crucially, this includes the precise calculation methodologies for the HEFA (Hydroprocessed Esters and Fatty Acids) process as clearly outlined in the core specification documents. We have successfully completed the translation of these complex, often calculation-intensive, research models into an accessible, user-friendly web-based platform. This platform is specifically designed to empower business users to perform detailed, reliable SAF production analysis and scenario planning without requiring deep technical or engineering expertise.

Furthermore, this document formally outlines the current, production-ready deployment status of the SAFAPAC platform on robust Amazon Web Services (AWS) infrastructure. This deployment not only validates our technical capability to deliver a fully functional, secure, and highly accessible web application, but it also means the platform is now live and immediately available for comprehensive testing and user acceptance. This launch represents a significant, successful milestone in our overall development roadmap and establishes a solid, scalable foundation for all planned future enhancements and long-term scalability requirements. The successful implementation of SAFAPAC validates our strategic approach to building a reliable and sophisticated techno-economic analysis tool that directly supports critical investment decisions and strategic planning activities for sustainable aviation fuel projects, thereby directly

contributing to the broader national objective of advancing Malaysia's influential position within the burgeoning SAF ecosystem.

2. SAFAPAC System Architecture Overview

2.1 System Components: Calculation System, User Interface, and Data Flow

The SAFAPAC system is a vital digital platform providing our leadership and partners with the swift, reliable analysis needed for major investment decisions in Sustainable Aviation Fuel (SAF) projects. It is specifically designed to be both powerful and user-friendly. While a complex analytical engine works behind the scenes, the interface delivers clear, timely information that is essential for strategic action, ensuring the platform is robust yet easily accessible for executive review.

The system's core function is to perform the detailed financial and scientific modeling for SAF production. This involves three critical areas. First, it determines a project’s fundamental financial viability by accurately modeling all upfront and continuous costs. Second, it executes comprehensive financial modeling to calculate expected profitability and returns using all key investment metrics. Most importantly, it rigorously tracks Carbon Intensity, accurately measuring the reduction in greenhouse gas emissions. This ensures all our projects are fully compliant with mandatory global standards, such as CORSIA and the EU Renewable Energy Directive (RED).

The system employs a sophisticated four-layer calculation architecture that systematically processes user inputs into comprehensive analytical outputs:

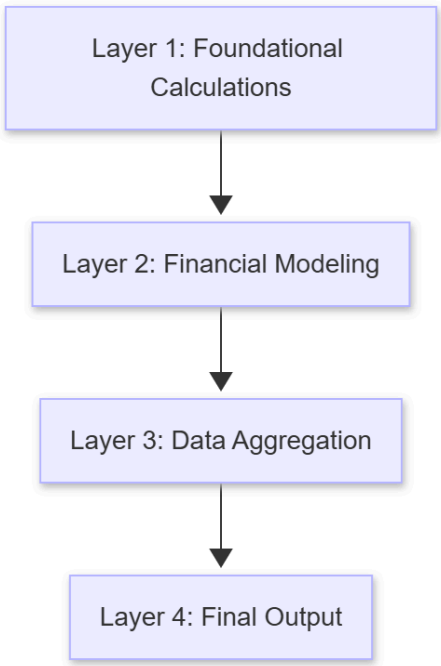


Figure 1. Overall layer calculation in the system.

Layer 1: Foundational Calculations

- Scaling of plant capacity and projection of production volumes
- Determination of feedstock and utility consumption rates based on established process yields
- Calculation of energy and carbon content for input material streams
- Derivation of baseline carbon intensity metrics for discrete process components

Layer 2: Financial Modeling

- Computation of total capital investment utilizing established scaling laws
- Detailed decomposition of operating expenditures (including feedstock and utility costs)
- Forecasting of revenue based on anticipated product yields and prevailing market prices
- Comprehensive carbon accounting across all primary process inputs

Layer 3: Data Aggregation

- Consolidation of total direct Operating Expenditures (OPEX) from all constituent cost elements
- Calculation of production-weighted carbon intensity metrics

- Summation of resource consumption totals and determination of overall efficiency indicators
- Reporting of intermediate financial parameters and structural cost analyses

Layer 4: Final Output Generation

- Calculation of the Levelized Cost of Production (LCOP)
- Quantification of total annual CO2 emissions
- Comparison line graph calculation for each scenario
- Preparation of complete financial statements and cash flow projections into exported csv files

This layered approach ensures mathematical rigor and computational efficiency, with each layer building upon validated results from the previous stage. The methodology has been specifically designed to mirror the research team's established calculation frameworks, ensuring consistency between web application outputs and academic research models.

The user-facing architecture is distinguished by a highly responsive and intuitive user interface, which has been developed leveraging the industry-leading JavaScript library, React. The fundamental design principle prioritizes accessibility, specifically enabling non-technical business stakeholders—including executives, project managers, and sustainability officers—to directly engage with sophisticated analysis tools. This design choice effectively eliminates the necessity for specialized technical proficiency among core users.

The user interface is entrusted with the management of the comprehensive user experience and critical functional capabilities. These functions are categorized into three primary areas: Project Creation and Management, Scenario Management, and Data Visualization.

Project Creation and Management is essential for structuring new Sustainable Aviation Fuel (SAF) initiatives and ensuring the secure organization of all associated analytical data. Scenario Management facilitates the rapid definition, cloning, and comparative analysis of potential outcomes, supporting the modeling of up to one default calculation alongside two additional scenarios. Finally, Data Visualization presents the analytical results through interactive charts and dashboards, effectively translating complex data sets into easily comprehensible formats for informed decision-making.

The interface is specifically designed to distill intricate analytical outcomes, providing clear, actionable insights that drive strategic decision-making. Users interact with structured input forms that guide them through the necessary parameters while the system handles the underlying computational complexity.

The connectivity between these two components is achieved through a meticulously well-defined API layer. This layer facilitates seamless and asynchronous data exchange: the frontend transmits structured user inputs (project parameters, scenario assumptions) to the backend for processing, and in return, receives standardized, structured results (TEA outputs, CI scores, financial metrics) ready for immediate visualization and display.

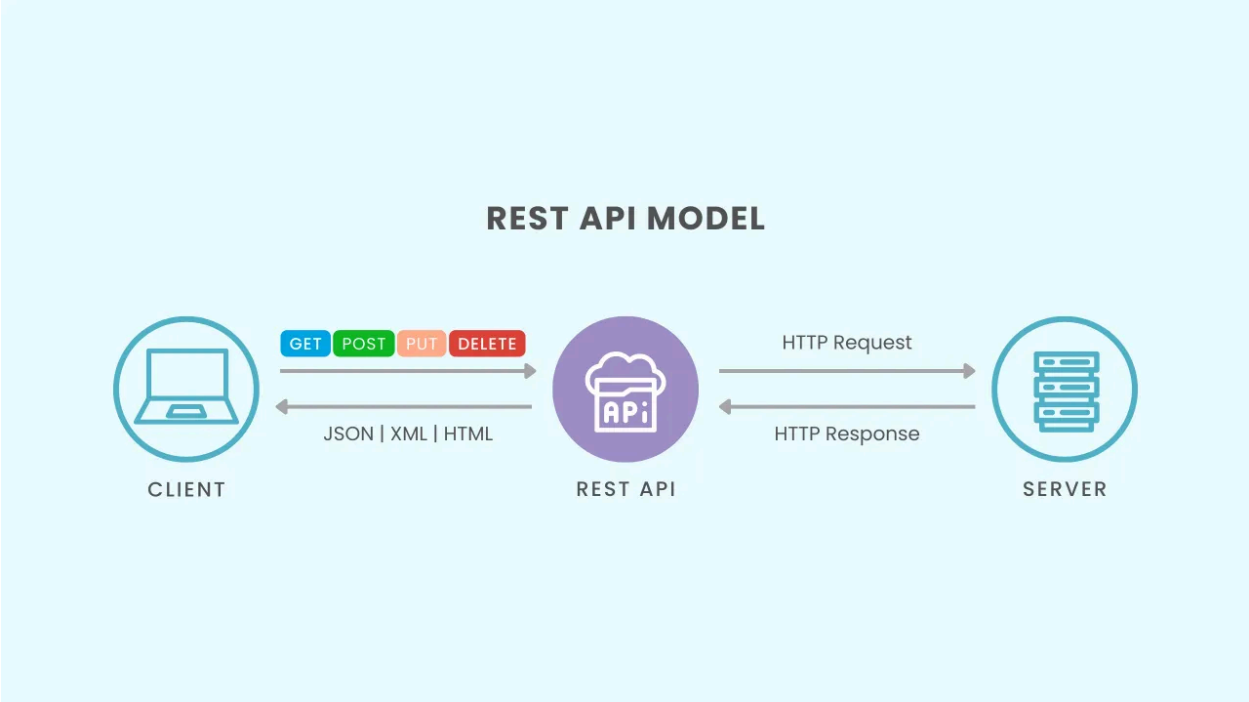


Figure 2. RESTful API architecture model.

The current implementation utilizes a JSON-based storage system for the efficient management of user projects and analysis scenarios. This foundational choice provides a robust, flexible base that maintains agility while simultaneously laying the groundwork for future PostgreSQL database integration as system demands evolve. This strategic architectural approach ensures that stakeholders can dedicate their attention to high-level strategic decision-making and market

analysis, confident that the SAFAPAC system is reliably handling all computational and analytical complexity behind the scenes.

2.2 User Roles and Access Levels (CORE, ADVANCE, ROADSHOW)

The SAFAPAC platform employs a sophisticated multi-tiered access system designed to cater to diverse user profiles with varying analytical requirements and technical expertise. This strategic approach ensures that each stakeholder group receives an experience precisely calibrated to their specific needs, from high-level executive decision-making to detailed research and development activities.

Table 1. Access tier level

Access Tier	**Target Users**	**Primary Features/Purpose**	**Key Analytical Capabilities**
CORE	Industry experts, project managers, financial analysts	Definitive standard for professional SAF analysis, supporting informed decision-making.	Complete access to techno-economic modeling tools (CapEx, OpEx, financial metrics); manipulation of critical parameters (feedstock, efficiency, economics); organized visualization dashboards.
ADVANCE	Developers, researchers, technical specialists	Extensive laboratory environment for R&D, process optimization, and cutting-edge SAF pathway development.	Advanced parameters and experimental features (granular process variables, alternative pathways); extensive customization; detailed sensitivity analysis;

			advanced scenario comparison.
ROADSHOW	C-suite executives, government officials, potential investors (non-technical audiences)	Executive presentations, stakeholder briefings, and exhibition environments; focuses on simplicity and visual impact.	Streamlined interface emphasizing high-level results and compelling data visualizations; straightforward scenario comparisons; communicates economic and environmental benefits.

This tiered access framework ensures that SAFAPAC delivers appropriate value to each segment of the sustainable aviation fuel ecosystem, from technical researchers pushing the boundaries of production technology to business leaders making strategic investment decisions about the future of aviation fuels.

2.3 Key Features:

2.3.1 Project & Scenario Management

The SAFAPAC system is strategically designed with a robust, project-centric workflow that precisely mirrors the structure of professional investment analysis for Sustainable Aviation Fuel (SAF) ventures. At its foundation, each "Project" serves as a dedicated, high-fidelity container for modeling a potential SAF production facility. A critical capability within this framework is the ability to create and manage up to three independent "Scenarios." This structure enables the direct, side-by-side comparison of disparate operational and financial assumptions—such as variations in feedstock pricing, modifications to plant capacity and technology choices, or changes in the capital structure and debt-equity ratios—thereby providing a comprehensive and deeply comparative basis for meticulous due diligence and risk assessment.

The system features a Structured and Intuitive Workflow that significantly streamlines the user experience. This workflow guides users logically from the point of access through to project initiation or resumption. Crucially, upon creation, the system automatically applies the "Scenario

1" baseline, which expedites the initial setup and allows for rapid engagement with core analytical tasks. Furthermore, the platform supports rich metadata project creation, ensuring that each analysis is well-documented and easily searchable, while supporting up to three distinct parameter sets per project to capture the full range of investment possibilities.

Another key feature is its support for Efficient Comparative Analysis, which is essential for data-driven strategic decision-making. The system facilitates seamless navigation and rapid toggling between the different scenarios within a single project. This comparative view enables users to quickly identify and act upon key differences by visually emphasizing discrepancies in core economic metrics such as the Levelized Cost of Production (LCOP) and Net Present Value (NPV) across multiple scenarios. This visualization is typically presented on an economic line chart, providing an immediate, clear strategic overview.

Finally, Data Integrity and Auditability are paramount and ensured through a comprehensive Auto-save Functionality. This feature automatically and consistently persists all user-defined input parameters and corresponding calculated outputs for every scenario within a project. This guarantees that all analyses conducted within SAFAPAC are fully reproducible and auditable, establishing a reliable and transparent audit trail—a necessity for securing investment discussions, achieving regulatory compliance, and satisfying stakeholder governance requirements. The system also supports the export of all analytical data to CSV format for external validation and reporting.

2.3.2 Techno-Economic Analysis (TEA)

The SAFAPAC system represents a critical tool for strategic decision-making in the Sustainable Aviation Fuel (SAF) sector. Its core functionality is underpinned by a robust Techno-Economic Analysis (TEA) engine, meticulously designed to translate highly complex, multi-variable production and market parameters into clear, quantitative, and actionable business intelligence. The system operates through a highly structured and transparent calculation workflow, strictly adhering to the rigorous, validated, and peer-reviewed methodology established by the research and development team.

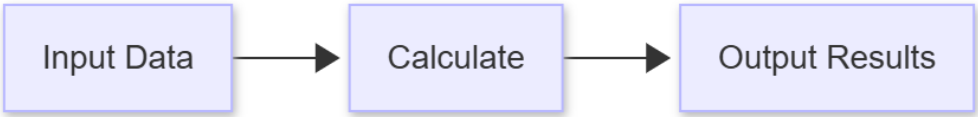


Figure 3. TEA Calculation process

A. Detailed Input Data Requirements

The accuracy and relevance of the SAFAPAC outputs are directly dependent on the quality and specificity of the user-provided inputs. The system requires a comprehensive data set across six major categories:

- **Plant Capacity Specifications (KTPA):** This is the fundamental scaling variable, defining the nameplate capacity of the facility in kilotons per annum (KTPA). This metric directly influences the magnitude of capital expenditure and the total annual production volume.
- **Process Technology Selection:** Users must specify the chosen conversion pathway, such as Hydroprocessed Esters and Fatty Acids (HEFA), Power-to-Liquid (PtL), Alcohol-to-Jet (AtJ), or other relevant technologies. Each technology dictates specific yields, utility consumption rates, and capital cost coefficients.
- **Feedstock Characteristics:** This input is crucial and includes not only the raw material price (e.g., in USD/ton or USD/kg) but also essential technical characteristics such as carbon intensity/content, higher heating value (HHV) or energy content, and specific processing requirements.
- **Utility Requirements:** Detailed inputs on all necessary consumables are required, including hydrogen consumption and electricity. These are typically derived from process simulations for the selected technology.
- **Product Slate Configuration:** The system accommodates flexible output configurations. Users define the proportions of the final product mix, such as the split between Sustainable Aviation Fuel (jet fuel), renewable diesel, naphtha, and other co-products. This directly impacts overall revenue projection.
- **Economic Parameters:** To ensure a comprehensive financial appraisal, users must specify key financial metrics, including the required internal rate of return (IRR), the

nominal or real discount rate, the project's assumed operating lifetime (e.g., 20 years), and the inflation rate.

B. Structured Calculation Workflow

The TEA engine executes a sequential, four-stage process to transform the raw inputs into the final outputs:

1. **Production Modeling:** This initial step determines the technical feasibility and scale of operations. Based on the specified plant capacity and the process yields, the system accurately calculates the anticipated annual production volumes for each specified product in the slate.
2. **Resource Consumption & Mass Balance:** A rigorous mass and energy balance calculation is performed. This stage precisely quantifies the total required consumption of feedstock, utilities (electricity, hydrogen, heat), and catalysts necessary to achieve the modeled annual production volume.
3. **Comprehensive Cost Analysis:** This is a dual-component calculation covering all expenditures:
    - **Capital Investment (CAPEX):** The system calculates the Total Capital Investment (TCI) required, often employing established capacity-based scaling laws applied to the base plant cost for the chosen technology. This includes direct costs, indirect costs, and contingency.
    - **Operating Expenses (OPEX):** A detailed breakdown of recurring operational costs is computed, encompassing variable costs (feedstock, utilities, consumables) and fixed costs (labour, maintenance, insurance, and indirect overheads).
4. **Revenue Projection & Economic Modeling:** The final technical stage involves estimating the total projected income. This is calculated by multiplying the modeled production volumes for each product (SAF, renewable diesel, etc.) by the prevailing or projected market prices, incorporating any applicable subsidies, tax credits, or carbon pricing mechanisms.

C. Essential Key Outputs

The SAFAPAC system delivers a set of high-value metrics, facilitating direct comparison between competing technology pathways and project configurations:

- **Total Capital Investment (TCI):** The definitive total upfront cost required to build the facility, clearly articulated with the capacity-based scaling methodology applied to ensure project-specific accuracy.
- **Detailed OPEX Breakdown:** A granular report on the recurring operational costs, clearly separating the dominant cost drivers: feedstock, utilities, personnel, and fixed/indirect costs.
- **Levelized Cost of Production (LCOP):** This is arguably the most critical economic output. It is calculated as the minimum constant price per unit of SAF (USD/ton or USD/gallon) required to achieve the user-specified minimum acceptable financial return (IRR) over the project's lifetime.
- **Calculated Production Volumes and Associated Resource Consumption Rates:** A technical summary providing the final annual tonnages of each product and the corresponding key resource inputs.

The underlying principle of the TEA engine is its commitment to methodological standardization. By consistently applying the research team's validated, peer-reviewed framework, the SAFAPAC system ensures that all computational results are consistent, transparent, and—most importantly—comparable across diverse feedstock types, technological processes, and economic assumptions.

2.3.3 Financial Modeling (NPV, IRR, Payback)

The SAFAPAC system offers advanced financial evaluation capabilities that go beyond simple cost analysis by incorporating sophisticated financial modeling. This comprehensive approach is essential for rigorously assessing a project's viability as an investment, providing detailed financial projections that cover the project's entire anticipated operational life.

The core of the financial projection includes a meticulous Cash Flow Analysis. This involves annual forecasts for key financial elements such as revenue generated from product sales, operational expenditures (including feedstock and utility costs), a structured schedule for capital expenditures with corresponding depreciation timelines, careful analysis of tax liabilities and associated financing costs, and the determination of necessary working capital.

Furthermore, the system furnishes critical Investment Metrics for robust project appraisal. These indicators include the Net Present Value (NPV), which quantifies the current value of expected future cash flows and serves as a definitive measure of overall profitability. The Internal Rate of Return (IRR) is also provided, establishing the effective annual compounded rate of return the project is anticipated to yield. Finally, the Payback Period is calculated, indicating the duration required for the initial capital outlay to be fully recovered.

The underlying financial model is built upon realistic operational parameters, taking into account factors such as the construction timeline, specifics of loan agreements, applicable tax structures, and working capital demands. This holistic and integrated methodology allows stakeholders to thoroughly evaluate not only the technical feasibility but also the financial attractiveness and the inherent risks associated with the venture.

2.3.4 Carbon Intensity Tracking

The SAFAPAC system is specifically designed to address the crucial need for accurate and robust environmental tracking within the Sustainable Aviation Fuel (SAF) sector. Its core function is the calculation and accounting of carbon intensity and emissions, which are indispensable metrics for evaluating the ecological feasibility and sustainability credentials of new SAF production projects. By providing clear, verifiable data on the environmental impact, SAFAPAC supports the necessary assessment required for project viability.

The system performs a comprehensive environmental analysis that is broken down into two main categories: Carbon Footprint Analysis and Environmental Performance Metrics. The Carbon Footprint Analysis meticulously tracks emissions from the initial feedstock stage through to the final fuel product, calculating feedstock-specific carbon intensity, factoring in emissions from conversion processes and utility consumption, and culminating in the total well-to-wake carbon intensity. This provides a holistic view of the greenhouse gas burden associated with the SAF lifecycle.

Complementing this are the Environmental Performance Metrics, which offer key indicators of efficiency and climate benefit. These include the Carbon Conversion Efficiency, which measures the success rate of converting carbon from the source material into the final fuel, and the Total CO2 Emissions, quantifying the annual environmental burden. Crucially, the system calculates

the Carbon Intensity Reduction, a direct comparison against the emissions of conventional jet fuel, which serves as the primary evidence of the SAF pathway's environmental advantage.

This rigorous approach guarantees that all generated data is suitable for sustainability reporting, regulatory compliance, and meeting stakeholder expectations. This transparency and data integrity are vital for quantifying the environmental benefits of different SAF pathways, which in turn supports decisions regarding carbon credit eligibility and justifies the premium pricing often associated with fuels designed to significantly reduce carbon emissions.

3. Backend Implementation

3.1 Technology Stack:

The SAFAPAC system's backend is built on a carefully chosen, high-performance technology foundation. This strategic decision was made to guarantee the accuracy of complex calculations, ensure development efficiency, and simplify long-term maintenance. This infrastructure is designed to be robust and scalable, enabling the system to handle demanding enterprise-level requirements.

The core technology stack consists of three main components. First, we utilize the FastAPI Framework for our web services. This framework was selected because it delivers exceptional speed, which is crucial for processing complex calculations and delivering rapid response times to our users. A key benefit for us is the automatic, interactive documentation it provides, making it easier for technical staff to understand and manage the system's capabilities, thereby reducing support costs. Its modern design also allows us to efficiently handle many concurrent user requests without any drop in performance.

Figure 4. FastAPI framework

Second, Python is the primary programming language for the computational engine. Python is the industry standard for scientific and financial computing, giving us immediate access to powerful, specialized libraries like Pandas and NumPy. This ensures direct alignment with our research team's existing mathematical models, making it easier to integrate new research. Furthermore, Python's clean and readable structure significantly reduces the effort and cost associated with code review, testing, and long-term system maintenance.

Figure 5. Python used in FastAPI

Finally, we employ Pydantic for comprehensive data validation. This component is essential for maintaining the integrity of our calculations. Pydantic automatically checks all user inputs against predefined business rules, catching errors before they can disrupt a calculation. This strict enforcement of data quality minimizes the risk of inaccuracies, ensures that all financial inputs are correctly formatted, and provides clear error messages, improving the overall user experience and reducing the need for manual data correction.

In summary, this technology architecture represents a deliberate balance between maximizing computational power and optimizing development efficiency. The synergy of FastAPI's speed, Python's scientific depth, and Pydantic's data validation rigor creates a highly reliable and accurate platform for all complex SAF calculations, guaranteeing the trustworthiness and performance of the overall SAFAPAC system.

3.2 Core Modules:

3.2.1 TEA Engine

The Techno-Economic Analysis (TEA) Engine represents the computational core of SAFAPAC, implementing a sophisticated four-layer calculation methodology that systematically transforms user inputs into comprehensive business intelligence. This structured approach ensures mathematical accuracy while maintaining alignment with the research team's validated calculation frameworks.

The TEA Engine operates through a coordinated sequence of computational layers, each building upon the validated results of the previous stage:

```python
# Core calculation orchestration
layer1_results = self.layer1.compute(ref, layer_inputs)
layer2_results = self.layer2.compute(layer1_results, ref, layer_inputs)
layer3_results = self.layer3.compute([layer2_results])
layer4_results = self.layer4.compute(layer2_results, layer3_results,
layer1_results, discount_rate, plant_lifetime)
```

Figure 6. Core calculation code snippet

This sequential processing ensures that each calculation layer receives verified inputs, maintaining data integrity throughout the analytical pipeline. The system begins by retrieving reference data for the selected process technology and feedstock combination, then applies user-specified overrides where provided, creating a customized calculation baseline.

Layer 1: Foundation Calculations

The first layer converts basic plant parameters into fundamental production metrics, serving as the building blocks for all subsequent analysis:

```python
# Core production calculations
tci = self.total_capital_investment(tci_ref, plant_capacity, capacity_ref)
feedstock_cons = self.feedstock_consumption(plant_capacity, feedstock_yield)
production = plant_capacity * product_yield
h2_cons = self.hydrogen_consumption(plant_capacity, yield_h2)
```

Figure 7. Layer 1 calculation

Key Outputs Generated:

- Total Capital Investment: Calculated using economy-of-scale principles where larger plants benefit from reduced per-unit costs
- Production Volumes: Annual output projections for all specified products (SAF, diesel, gasoline, etc.)
- Resource Consumption: Quantification of feedstock, hydrogen, and electricity requirements
- Carbon Conversion Efficiency: Measurement of how effectively carbon from feedstock converts to final products

This layer establishes the physical and financial scale of the operation, providing the essential inputs for detailed cost and revenue analysis.

Layer 2: Financial Modeling

Building upon the foundation layer, this module develops comprehensive financial models by integrating cost data and market prices:

```python
# Cost and revenue calculations
total_indirect_opex = self.total_indirect_opex(process_ratio,
total_capital_investment)
feedstock_cost = self.feedstock_cost(feedstock_consumption, feedstock_price)
hydrogen_cost = self.hydrogen_cost(hydrogen_consumption, hydrogen_price)
revenue_i = amount * price  # Per-product revenue
```

Figure 8. Layer 2 calculation

Financial Components Analyzed:

- Operating Expenditures: Breakdown of feedstock, utility, and indirect operating costs
- Revenue Streams: Projected income from each product based on market prices and production volumes
- Carbon Intensity: Comprehensive emissions accounting across all process inputs
- Process Economics: Technology-specific cost ratios and efficiency factors

This layer transforms physical production data into financial metrics, enabling stakeholders to understand the cost structure and revenue potential of different SAF pathways.

Layer 3: Aggregation and Weighting

The third layer synthesizes multiple data streams into unified performance indicators, providing a holistic view of project economics:

```python
# Cost aggregation and weighting
total_direct_opex = self.total_direct_opex(feedstock_costs, hydrogen_costs,
electricity_costs)
total_weighted_ci = sum(ci * y for ci, y in zip(total_carbon_intensity_values,
product_yields))
```

Figure 9. Layer 3 calculation

Consolidation Functions:

- Total Direct OPEX: Combined variable costs from all feedstocks and utilities
- Weighted Carbon Intensity: Production-adjusted emissions metrics that reflect the actual environmental impact per unit of output
- Efficiency Metrics: Resource utilization rates and process efficiency indicators

This aggregation ensures that multi-feedstock operations and complex product slates are accurately represented in the final economic analysis.

Layer 4: Final Output Generation

The final layer produces the key performance indicators that drive investment decisions and strategic planning:

```python
# Final KPI calculations
total_opex = self.total_opex(total_direct_opex, total_indirect_opex)
total_co2 = self.total_co2_emissions(carbon_intensity, product_energy_content,
production)
lcop = self.levelized_cost_of_production(feedstock_cost, hydrogen_cost,
electricity_cost, total_indirect_opex, capital_investment,
liquid_fuel_capacity, discount_rate, plant_lifetime)
```

Figure 10. Layer 4 calculation

Critical Business Metrics:

- Levelized Cost of Production (LCOP): The comprehensive cost per ton of SAF, incorporating capital recovery and all operating expenses
- Total CO2 Emissions: Annual greenhouse gas emissions quantified for environmental reporting and compliance
- Total OPEX: Complete operating expenditure for cash flow planning and operational budgeting
- Carbon Intensity: Final emissions intensity metric for sustainability certification and premium pricing eligibility

The LCOP calculation employs sophisticated capital recovery factoring that distributes initial investment costs across the project lifetime, providing a true representation of long-term production economics.

The TEA Engine's modular architecture also supports future enhancements, allowing new process technologies and calculation methodologies to be integrated without disrupting existing functionality. This forward-compatible design ensures that SAFAPAC can evolve alongside emerging SAF production technologies and changing market conditions.

3.2.2 Cash Flow & KPIs

The Financial Analysis module transforms the static cost data from the TEA Engine into dynamic, time-based financial projections that assess investment viability and risk. This

sophisticated financial modeling capability provides stakeholders with the critical metrics needed for investment decision-making and project financing.

A. Financial Analysis Module

The system implements a complete financial lifecycle analysis that spans construction, operation, and project close-out phases:

```python
# Financial analysis initialization
fa = FinancialAnalysis(
    discount_rate=0.105,
    tax_rate=0.28,
    equity=0.4,
    bank_interest=0.04,
    loan_term=10
)
```

Figure 11. Financial Analysis Initialization

Financial Parameters Modeled:

- Discount Rate: 10.5% reflecting the time value of money and investment risk profile
- Tax Structure: 28% corporate tax rate applied to taxable income
- Capital Structure: 40% equity financing with 60% debt financing
- Loan Terms: 10-year loan term at 4% annual interest rate

B. Construction Timeline Modeling

Unlike simplified financial models, SAFAPAC incorporates a detailed three-year construction phase that accurately reflects project development realities:

```python
def capex_schedule(self, tci: float) -> dict:
```

```
    land = self.land_cost(tci)
    equity_investment = self.capital_investment(tci)
    wc = self.working_capital(tci)

    return {
        -2: land,
        -1: equity_investment,
        0: wc
    }
```

Figure 12. 3 year construction phase

This phased approach accurately captures the cash outflow pattern during project development, providing a realistic foundation for investment return calculations.

Construction Phase Breakdown:

- Year -2: Land acquisition costs representing the initial project commitment
- Year -1: Equity investment covering major capital expenditures
- Year 0: Working capital infusion to support initial operations

C. Cash Flow Table Generation

The cash flow table systematically tracks all financial movements across 2 distinct phases:

1. **Construction Phase (Years -2 to 0):**

```python
if year < 1:  # Construction phase
    revenue_y = 0
    cost_y = 0
    depreciation_y = 0
    loan_y = 0
    income_tax = 0
    after_tax_profit = 0
    after_tax_cf = -capex  # Negative cash flow for investments
```

Figure 13. Construction phase(Year -2 to 0)

During construction, the model captures pure capital expenditure without revenue generation, reflecting the investment phase where cash flows are exclusively negative.

2. **Operational Phase (Years 1 to Project Lifetime):**

```python
else:  # Operation phase
    revenue_y = revenue
    cost_y = manufacturing_cost
    depreciation_y = depreciation_annual

    # Income Tax = (Revenue - Depreciation - Loan - Cost) × Tax Rate
    taxable_income = revenue_y - depreciation_y - loan_y - cost_y
    income_tax = max(0, taxable_income * self.tax_rate)

    # After-Tax Net Profit = Revenue - Loan - Cost - Tax
    after_tax_profit = revenue_y - loan_y - cost_y - income_tax

    # After-Tax Cash Flow = ATNP + Depreciation
    after_tax_cf = after_tax_profit + depreciation_y
```

Figure 14. Operational Phase (Years 1 to Project Lifetime)

The operational phase incorporates sophisticated financial accounting including:

- Revenue Recognition: Annual income from product sales
- Cost Accounting: Comprehensive operating expenses
- Tax Calculations: Realistic tax liability based on taxable income
- Depreciation: Capital cost recovery through annual depreciation
- Loan Servicing: Debt repayment schedules and interest costs

D. System Output

The system generates three critical investment decision metrics that evaluate project attractiveness from different perspectives:

1. **Net Present Value (NPV)**

NPV represents the total value created by the project in today's dollars, accounting for the time value of money. A positive NPV indicates that the project generates returns exceeding the required investment threshold.

```python
def npv(self, tci: float, revenue: float, manufacturing_cost: float,
plant_lifetime: int) -> float:
    """Net Present Value = Final Cumulative DCF"""
    df = self.cash_flow_table(tci, revenue, manufacturing_cost, plant_lifetime)
    return df["Cumulative DCF (USD)"].iloc[-1]
```

Figure 15. NPV snippet code

2. **Internal Rate of Return (IRR)**

IRR calculates the annualized effective compounded return rate, providing a percentage-based metric that facilitates comparison with alternative investment opportunities and corporate hurdle rates.

```python
def irr(self, tci: float, revenue: float, manufacturing_cost: float,
plant_lifetime: int) -> float:
    """Internal Rate of Return: solve for IRR where NPV = 0"""
    df = self.cash_flow_table(tci, revenue, manufacturing_cost, plant_lifetime)
    cash_flows = df["After-Tax Cash Flow (USD)"].to_list()
    try:
        return nf.irr(cash_flows)
    except:
        return None  # IRR may not converge
```

Figure 16. IRR snippet code

3. Payback Period

Payback Period identifies when the initial investment is fully recovered, serving as a key risk indicator and liquidity measure for capital-constrained organizations.

```python
def payback_period(self, tci: float, revenue: float, manufacturing_cost: float,
plant_lifetime: int) -> int:
    """Payback Period: Year when Cumulative DCF first becomes positive"""
    df = self.cash_flow_table(tci, revenue, manufacturing_cost, plant_lifetime)
    positive_years = df.loc[df["Cumulative DCF (USD)"] > 0, "Year"]
    return int(positive_years.min()) if not positive_years.empty else None
```

Figure 17. Payback period snippet code

The financial analysis module seamlessly integrates with the broader SAFAPAC platform, enabling:

- Scenario Comparison: Direct financial comparison of different technology pathways
- Sensitivity Analysis: Assessment of how changes in key assumptions impact financial returns
- Risk Assessment: Identification of financial vulnerabilities and upside potential
- Investment Prioritization: Objective ranking of competing SAF project opportunities

This comprehensive financial modeling capability provides Airbus and AMIC leadership with the robust, investment-grade analysis required to make informed decisions about SAF project development and capital allocation. The system's outputs directly support business case development, financing negotiations, and strategic planning activities.

3.3 API Endpoints:

3.3.1 Authentication

The authentication system provides secure access control while maintaining ease of use for business stakeholders. The implementation balances security requirements with practical user experience considerations for enterprise deployment.

A. Current Authentication Flow

The system employs a structured approach to user authentication, leveraging secure credential storage and validation:

```python
def load_valid_credentials():
    """Load valid credentials from pw.csv"""
    csv_path = Path(__file__).parent.parent / "pw.csv"
    credentials = {}

    credentials[password] = {
      "user_id": f"user_{email_hash}",
      "username": password,
      "email": email,
      "name": name,
      "role": "user"
    }
```

Figure 18. Simplified authentication flow

Key Authentication Features:

- Secure Credential Storage: User credentials are maintained in a separate, controlled file outside the application codebase in csv file
- Consistent User Identification: Unique user IDs generated from email addresses ensure reliable user session management
- Role-Based Access: Foundation for future permission hierarchies and feature access control

B. Login Process and Session Management

The login process validates user credentials and automatically establishes the necessary user context for project management:

```python
@app.post("/auth/login")
def login(request: LoginRequest):
    """
    Authenticate user against pw.csv credentials
    """
```

Figure 19. Login API endpoint

Authentication Flow:

- Credential Validation: User-provided credentials checked against the authorized user database
- User Provisioning: Automatic creation of user profile in the project management system
- Session Establishment: Return of user identity information to the frontend for session management
- Access Granting: Permission to create and access projects and scenarios

C. Security Monitoring and Logging

The system incorporates comprehensive security logging to track authentication attempts and detect potential security issues:

```python
logger.info(f"✅ Login successful for user: {user_data['name']}")
logger.warning(f"❌ Login failed for username: {request.username}")
```

Figure 20. Logging code for testing purpose

- Audit Trail: Detailed logging of successful and failed authentication attempts
- User Activity Tracking: Monitoring of user sessions and project access patterns
- Error Handling: Graceful handling of authentication failures without exposing system details

D. Future Security Roadmap

The authentication system is architected to evolve seamlessly into a production-ready security infrastructure.

- Current Phase: Current CSV-based system for initial testing and validation
- Intermediate Phase: Integration with AWS RDS for secure user data storage
- Advanced Phase: JWT token implementation with refresh token capabilities
- Enterprise Phase: Integration with corporate SSO and multi-factor authentication

This phased approach ensures that security evolves alongside platform functionality, providing appropriate protection at each stage of development while maintaining accessibility for authorized Airbus and AMIC stakeholders during the testing and validation phase.

3.3.2 Project & Scenario Management

The Project and Scenario modules are the structural backbone of the user experience, allowing researchers to organize their analyses. Our implementation prioritizes data integrity and a streamlined workflow. Rather than presenting the user with an empty slate, the system uses "smart creation" logic to ensure that every project is immediately ready for analysis upon creation.

A. Automated Workflow Initialization

When a user creates a new project, the system does not merely create a container folder. As shown in the code snapshot below, the backend automatically initializes "Scenario 1" within that

project. This design decision reduces the number of clicks required for a user to start their first simulation.

```python
@app.post("/projects/create", response_model=ProjectCreateResponse)
def create_project(project_data: ProjectCreate):
    # ... User verification logic ...

    # Create project
    project = mock_db.create_project(
        user_id=project_data.user_id,
        project_name=project_data.project_name
    )

    # Auto-create Scenario 1
    scenario = mock_db.create_scenario(
        project_id=project["project_id"],
        scenario_name="Scenario 1",
        order=1
    )
```

Figure 21. API endpoint to create new project data

This automation ensures that no "empty" projects exist in the database, keeping the data structure clean and allowing users to immediately input TEA parameters.

B. Business Rules and Constraints

To maintain system performance and encourage focused analysis, we have enforced specific business rules at the API level. Currently, the system limits each project to a maximum of three scenarios (e.g., Base Case, Optimistic Case, Pessimistic Case). This constraint prevents data bloat during this early testing phase and aligns with standard comparative analysis practices.

```python
@app.post("/scenarios/create", response_model=ScenarioResponse)
def create_scenario(scenario_data: ScenarioCreate):
    # ... Project validation ...
```

```
    # Check scenario count (max 3)
    current_count =
mock_db.count_scenarios_by_project(scenario_data.project_id)
    if current_count >= 3:
        raise HTTPException(status_code=400, detail="Maximum 3 scenarios per
project")

    # ... Creation logic ...
```

Figure 22. Validation code to ensure maximum number of 3 scenarios

C. Data Integrity Safeguards

The system includes safeguards to prevent accidental data loss. The delete function includes a check that prevents the user from deleting the *last remaining* scenario in a project. A project must always contain at least one active scenario to remain valid.

```
Python
@app.delete("/scenarios/{scenario_id}")
def delete_scenario(scenario_id: str):
    # ... Fetch scenario ...

    # Check if it's the last scenario
    project_id = scenario["project_id"]
    current_count = mock_db.count_scenarios_by_project(project_id)
    if current_count <= 1:
        raise HTTPException(status_code=400, detail="Cannot delete the last
scenario in a project")
```

Figure 23. Validation code to ensure the consistency of 1 scenario

3.3.3 TEA Calculation

The API calculation endpoint is the core intelligence of the SAFAPAC platform. It bridges the user's raw inputs with our proprietary calculation engines. The implementation ensures that

financial feasibility (NPV, IRR) is calculated strictly based on the technical realities of the engineering inputs.

A. Integration of Economics and Finance

The calculation process is a two-step sequence. First, the system processes the engineering data (feedstock types, plant capacity) to determine the Total Capital Investment (TCI) and Operating Expenses (OPEX). Second, these outputs are immediately fed into the Financial Analysis module.

In this current iteration, we have standardized the financial assumptions (Discount Rate: 10.5%, Tax Rate: 28%, Equity: 40%) to ensure consistency across all user tests, though these can be exposed as variables in future updates.

```python
@app.post("/calculate")
def calculate(request: CalculationRequest):
    # Step 1: Run Techno-Economic Analysis
    structured_inputs = UserInputs.from_dict(request.inputs)
    econ = BiofuelEconomics(structured_inputs)
    results = econ.run(
        process_technology=request.process_technology,
        feedstock=request.feedstock,
        product_key=request.product_key
    )

    # Step 2: Extract Key Financial Drivers
    tci = results["TCI"]
    revenue = results["Revenue"]
    manufacturing_cost = results["Total OPEX"]

    # Step 3: Run Financial Analysis with Standardized Metrics
    fa = FinancialAnalysis(
        discount_rate=flattened_inputs.get("discount_rate", 0.105),
        tax_rate=0.28,
        equity=0.4,
        bank_interest=0.04,
        loan_term=10
    )
```

Figure 24. API endpoints calculation, with fixed value for financial analysis

This separation of concerns ensures that the engineering calculations are isolated from the financial modeling, making it easier to audit the results.

B. Financial Metrics and Cash Flow Generation

Once the core analysis is complete, the API generates the critical KPIs required by business stakeholders: Net Present Value (NPV), Internal Rate of Return (IRR), and the Payback Period. It also generates a full year-by-year cash flow table, which is returned to the frontend for visualization in the dashboard charts.

```python
# Generate cash flow table
    cashflow_df = fa.cash_flow_table(
        tci=tci,
        revenue=revenue,
        manufacturing_cost=manufacturing_cost,
        plant_lifetime=plant_lifetime
    )

    financials = {
        "npv": safe_float(fa.npv(...)),
        "irr": safe_float(fa.irr(...)),
        "paybackPeriod": safe_float(fa.payback_period(...)),
        "cashFlowTable": cashflow_df.to_dict(orient="records")
    }
```

Figure 25. Generate cash flow table code

C. Error Handling and Stability

To ensure the application remains stable even when users input theoretical scenarios that might result in mathematical impossibilities (e.g., dividing by zero or infinite returns), we implemented a code wrapper. This ensures that the dashboard displays a "clean" error or null value rather than crashing the entire application.

```python
def safe_float(value):
    if isinstance(value, float):
        if math.isnan(value) or math.isinf(value):
            return None
    return value
```

Figure 26. Code wrapper to handle math error

The API structure is fully functional and successfully integrates the engineering specifications provided by the research team with a robust web architecture. The logic handles the full lifecycle from project creation to complex financial modeling, meeting the technical KPIs set for this phase.

3.4 Data Storage

To meet the project's aggressive timeline for validating the Techno-Economic Analysis (TEA) logic, the development team utilized a "Mock Database" strategy. This approach allowed us to decouple the complex calculation engine from infrastructure setup, ensuring that the mathematical accuracy of the SAFAPAC tool could be verified immediately while the enterprise database infrastructure was being provisioned.

3.4.1 Development Persistence Layer

Currently, the system operates using a file-based persistence layer. This acts as a functional mirror of a real database, handling User, Project, and Scenario data using structured JSON files. This allows the application to demonstrate full functionality—saving projects, retrieving history, and updating scenarios—without requiring a live database connection during the initial testing phase.

The code below illustrates how the system isolates "Storage Logic" from "Business Logic." Because all data operations go through this single class, we can swap the underlying storage mechanism without breaking the application.

```python
class MockDatabase:
    """
    Mock Database Service for SAFAPAC
    Uses JSON files for persistence during development
    Will be replaced with SQLite/PostgreSQL in production
    """

    # ... Initialization logic ...

    # Standard "CRUD" operations that mimic a real SQL database
    def create_project(self, user_id: str, project_name: str) -> Dict:
        projects = self._load_json(self.projects_file)

        # Generate unique ID (simulating a database primary key)
        project_id = f"proj_{uuid.uuid4().hex[:12]}"

        # ... Save logic ...
        self._save_json(self.projects_file, projects)
        return project

    def update_scenario(self, scenario_id: str, updates: Dict) ->
Optional[Dict]:
        # ... Load, Update, Save logic ...
```

Figure 27. Mock database initialization

This architecture allowed the team to deliver a working frontend and calculation engine weeks ahead of schedule. It provides a "sandbox" environment where stakeholders can test features without fear of corrupting production data.

3.4.2 Data Standardization and Validation

A critical requirement for Airbus and AMIC is engineering precision. While the storage is currently file-based, the data *entering* the system is strictly governed. We utilize Data Models (using a technology called Pydantic) to enforce strict typing.

The system automatically detects and normalizes units. For example, if a user inputs "USD/ton" and another inputs "USD/kg," the system standardizes these inputs before calculation. This prevents "Garbage In, Garbage Out" errors.

```python
@dataclass
class UserInputs:
    """
    Structured inputs that include value + unit metadata.
    """
    plant: ConversionPlant
    feedstocks: List[Feedstock] = field(default_factory=list)
    products: List[Product] = field(default_factory=list)

    # Intelligent Unit Conversion Logic
    @staticmethod
    def _price_to_usd_per_ton(price: Quantity) -> float:
        unit = price.normalized_unit()
        value = price.value

        if unit in {"usd/t", "usd/ton", "$/t"}:
            return value
        if unit in {"usd/kg", "$/kg"}:
            return value * 1000.0  # Auto-convert kg to ton

        raise ValueError(f"Unsupported price unit '{price.unit}'")
```

Figure 28. Unit standardization code

This layer ensures that all financial and technical data is clean, standardized, and mathematically valid before it is ever processed or stored.

3.4.3 Transition to Enterprise Architecture

While the Mock Database has served its purpose for logic validation, the IT development team is currently in the advanced stages of migrating to a production-grade database.

We are moving from the JSON-based system to AWS RDS (Relational Database Service) using PostgreSQL. This transition is seamless because the "MockDatabase" code shown above was

designed to be modular. The developers are simply replacing the "save to file" commands with "save to cloud database" commands, while keeping the rest of the system, including the calculation engine, untouched.

Current Integration Status:

- **Infrastructure:** The AWS RDS instance has been provisioned and is active.
- **Schema Design:** A comprehensive database schema has been designed to map the relationships between Users, Projects, Scenarios, and TEA Results.
- **Security:** The transition includes implementing robust authentication (OAuth2) to replace the current development login, ensuring secure, encrypted access for all corporate users.
- **Data Handling:** Initial tests for writing calculation outputs directly to the PostgreSQL database have been successful.

The platform is currently stable and fully testable using the mock architecture. The parallel effort to integrate the AWS SQL database is proceeding on schedule, ensuring that when the platform goes live for wider release, it will possess the scalability, security, and data backup capabilities required by an enterprise solution.

4. Frontend Implementation

This section is intended as a practical handover guide for collaborators working on the SAFAPAC App frontend. It does not replace the codebase, but provides the mental model needed to navigate the React application, understand why the UI is structured in a particular way, and know where to make changes when extending or refactoring the system. It should be read once end-to-end when joining the project, and then used as a reference for specific sections during ongoing development.

How to read this section.

- Readers should start with Section 1 (Overview & Design Goals) to understand what the frontend is for, who the users are, and the high-level UX/architecture principles it follows.
- They should then read Section 2 (Tech Stack & Dependencies) and Section 3 (Architecture & Data Flow) to learn which libraries and patterns are used (React, contexts, routing, Chart.js) and how data flows between components and backend APIs.
- Section 4 (Key Screens & UX Rationale) can be skimmed to see how the login, TEA dashboard, input forms, chart, and KPI cards are laid out and why they are designed that way (inputs vs outputs, three-column layout, slider + numeric input pattern, etc.).
- When functionality needs to be changed or added (new inputs, KPIs, charts, routes, or access-controlled features), collaborators should jump directly to Section 6 (Extending the Frontend) for “where to edit” instructions and concrete examples.

4.1. Overview & Design Goals

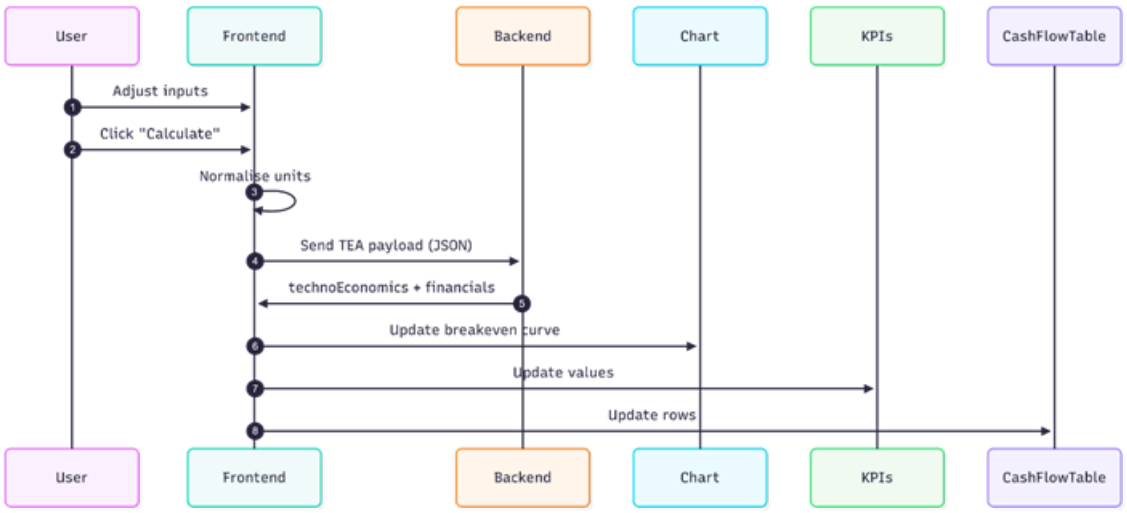


Figure 29. Overview of the TEA app workflow

The SAFAPAC frontend is a single-page React application designed to manage the complete TEA workflow. It handles user authentication, project and scenario selection, configuration of process and financial inputs, and the execution of TEA analysis through the backend API. The interface also presents analytical outputs such as breakeven charts, KPIs, and detailed cash-flow tables. It communicates with the FastAPI backend using JSON payloads, maintains session information in localStorage, and provides a consistent user experience across three main screens: the Login interface, the Analysis Dashboard, and a legacy Tables view primarily used for debugging. To summarise, the key elements of this section are as follows:

- The frontend manages the full TEA flow from authentication to results visualisation.
- Communication with the backend is done entirely through JSON over HTTP.
- Session state is persisted via localStorage.
- The primary user-facing screens are Login, Analysis Dashboard, and a legacy Tables view.
- Outputs include breakeven charts, KPI summaries, and detailed cash-flow tables.

4.2. Tech Stack & Dependencies

The interface is built using a modern JavaScript ecosystem centered around React 16.6, leveraging functional components and hooks throughout the codebase. It is scaffolded with Create React App, which provides the development server, build pipeline, and environment configuration through react-scripts. The interface layer uses shards-react on top of Bootstrap 4, supported by additional global styling defined in custom.css and the Shards Dashboard theme to ensure consistent design and usability.

Routing is handled using react-router-dom version 4, with all route definitions maintained in src/routes.js and the BrowserRouter configured within the main App.js file.

Table 2. Summary of the tech stack used to develop the front end

Category	Technology
Language and Runtime	JavaScript (ES6+)
Framework	React 16.6 (Hooks)
UI Components	shards-react, Bootstrap 4
Routing	react-router-dom@4
HTTP Client	axios
Charts and Visualisation	Chart.js 2.9
Environment Configuration	.env.development, .env.production
Build Tools	npm start, npm run build

All API communication is performed using axios, which handles RESTful interactions with the FastAPI backend. Visualization of analytical outputs is powered by Chart.js 2.9, wrapped in a custom component, BreakevenBarChart, to unify styling and behavior. The frontend supports separate development and production configurations through .env.development and .env.production files, and can be run or built using the standard npm start and npm run build commands.

4.3. Architecture & Data Flow

The frontend codebase is organized into a set of clearly defined directories, each reflecting a specific functional responsibility within the TEA workflow. The api directory contains modules responsible for interacting with the backend; the primary file, projectApi.js, manages all CRUD operations related to projects and scenarios.

User-facing pages reside in the views directory, which includes the Login interface, the core Analysis Dashboard, and a legacy Tables view used mainly for debugging purposes. The forms directory contains complex form components such as BiofuelForm.js for TEA inputs and CashFlowTable.js for displaying and exporting detailed financial results. Reusable UI elements, including charts, layout components, user menus, and project-related modals are grouped within the components directory, while the layouts directory provides the authenticated application shell through Default.js. Configuration files such as access.json are stored in the config directory, where feature gating rules for different access levels are defined.

A full visual map of these relationships is provided in Appendix A, where the directory structure and its interconnections are illustrated using a component flow diagram. To summarise, the main structural elements of the project are as follows:

- The api directory contains backend communication logic, primarily projectApi.js for project and scenario CRUD operations.
- Global application state is maintained in the contexts directory, including authentication, project selection, access control, and theming.
- All user-facing screens such as Login, Analysis Dashboard, and the legacy Tables view are located in the views directory.
- The forms directory contains TEA input components and detailed financial output tables.
- Reusable UI building blocks, charts, menus, and project modals are grouped under components.
- The authenticated page wrapper is implemented in layouts/Default.js.
- Configuration settings, including feature access rules, are maintained in config/access.json.

4.4. State Management

State management in the SAFAPAC frontend is centered around four key context modules that coordinate authentication, project data, feature access, and theming. Each context supplies a specific layer of global state to the application and together they form the foundation of the TEA workflow.

Authentication is handled through AuthContext, which manages login and logout operations by communicating with the backend API. It stores the authenticated status, the active user identity, and syncs this data with localStorage so that sessions persist across page reloads. ProjectContext is responsible for managing the current project and scenario, maintaining a list of comparison scenarios, and orchestrating all project and scenario CRUD operations through the API. AccessContext reads rules from access.json and exposes functions that determine which features are available to each access level. It also stores the selected access level in localStorage to keep the user's configuration consistent. Theming is managed through ThemeContext, which controls whether the application operates in light or dark mode, applies CSS variables accordingly, and provides color values used by charts and UI components throughout the interface.

Figure 30. Each of the context with its responsibility respectively

Alongside these global state concerns, the application centralizes all unit-handling and normalization logic within AnalysisDashboard.js. Incoming inputs such as hydrogen price, electricity rates, yields, and carbon intensity may be entered in different units by the user. Normalization functions convert these values into canonical units before they are sent to the backend, ensuring consistency and simplifying backend calculations. This design isolates unit-conversion logic from the rest of the interface and reduces the likelihood of discrepancies across different parts of the TEA workflow.

4.5. Key Screens & UX Rationale

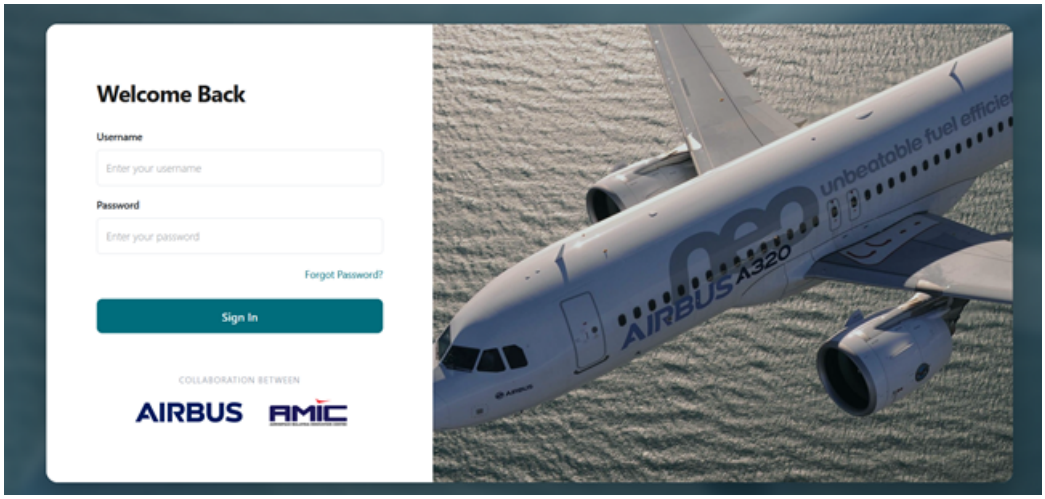


Figure 31. User login page

The SAFAPAC frontend includes several key screens that shape the user experience and support the full TEA workflow. The Login screen is designed as a full-screen interface with a background image and a subtle overlay that creates a focused entry experience. It presents users with a clean structure that separates the Sign In, and Forgot Password flows, which keeps the interface uncluttered and easy to navigate. For readability and consistency, the Login screen is displayed in light mode regardless of the user's theme settings.

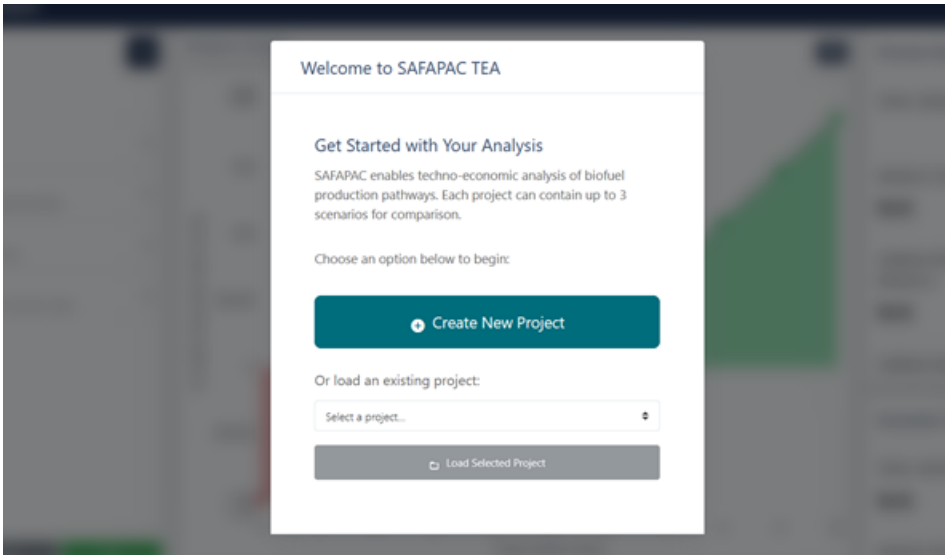


Figure 32. Project creation and selection page

The main operational environment of the application is the Analysis Dashboard. This screen organizes the TEA workflow into a predictable and efficient three-column layout. The left panel contains the BiofuelForm, where users modify all project inputs. The center panel displays the breakeven analysis chart. The right panel presents key performance indicators grouped into logical categories. Each section maintains its own scrolling behavior so that the rest of the screen remains stable during interaction. The left panel can be collapsed when the user wants to focus more heavily on the breakeven chart, which is especially useful during presentations or deeper analytical work. The fixed widths of the left and right panels ensure that the layout behaves consistently across different screen sizes.

Figure 33. Analysis Dashboard Developed

The BiofuelForm itself is structured to support both exploration and precision. Most inputs are presented as paired sliders and numeric fields, which allow fast adjustment as well as fine-grained control. The form is split into collapsible sections so users can focus on one category of parameters at a time without being overwhelmed. Product cards within the form support scenarios with multiple output products, and they enforce mass fraction requirements so that values always sum correctly. The form also integrates unit selection and normalization logic so users can enter values in familiar units while the system processes everything in canonical form.

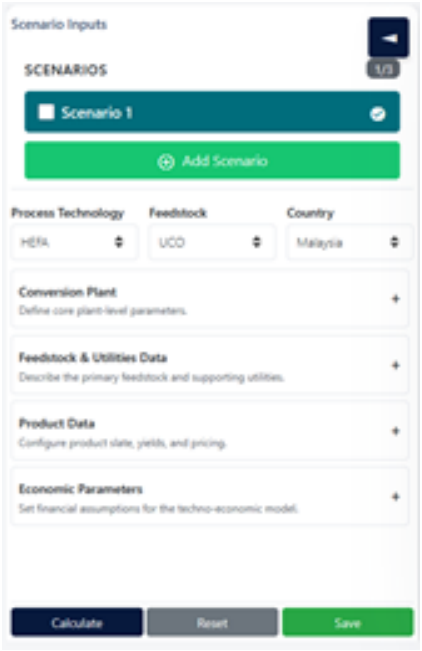


Figure 34. Biofuel.js form where the user provides the required input to execute the calculations

The breakeven chart and cash-flow table together form the analytical output of the dashboard. The breakeven chart is a hybrid of line and bar elements created using Chart.js, and it automatically adapts its colours based on the currently selected theme. The cash-flow table provides a detailed numerical breakdown of financial results and supports exporting the data to CSV or JSON, which helps users continue analysis outside the application if needed.

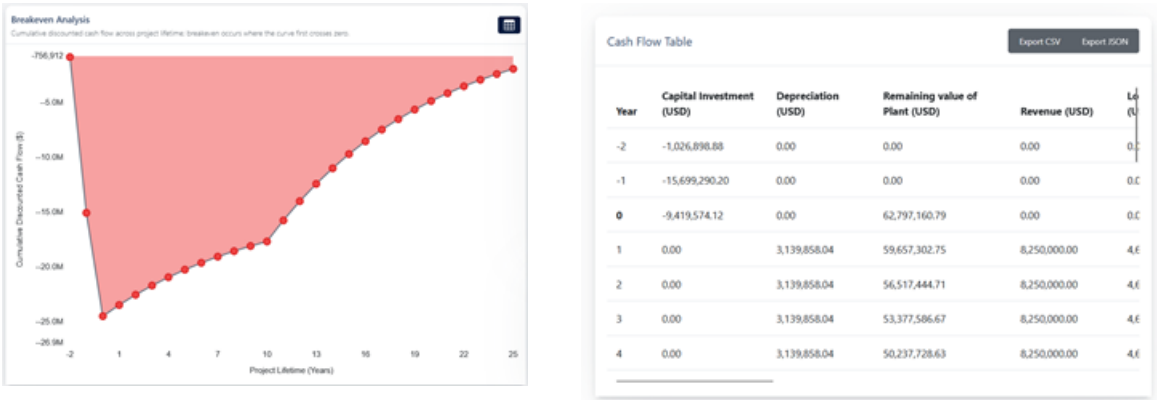


Figure 35. Left - the breakeven chart, the graph shows red, indicating no breakeven achieved. Right – the table generated, including all the calculation output.

The KPI cards on the TEA dashboard appear in the right-hand column as a set of summary tiles that present the most important process and economic metrics in a compact and readable format. They are generated from a kpiGroups structure containing two groups: Process Outputs and Economic Outputs. Each group is displayed as a vertical stack of KPIs, and users can maximise either group by clicking its header to focus on technical or financial indicators.

Within each group, every KPI is shown with a clear label and a prominent value that respects the current theme and currency settings. Certain metrics, such as total OPEX or levelized cost of production, include a Details toggle that expands a small breakdown section so users can examine cost contributors directly on the dashboard. In the Process Outputs group, the first block is Total Consumption, which displays a grid of small cards for feedstock, hydrogen, electricity and other inputs. This is followed by metrics such as total product output, carbon intensity per tonne, carbon conversion efficiency, and annual CO_2 emissions.

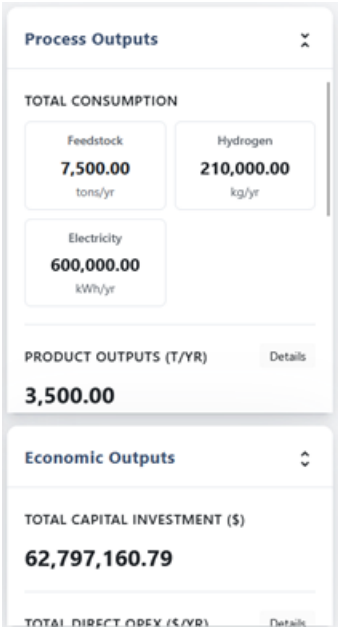


Figure 36. KPI Card, where all the output are summarised

The Economic Outputs group focuses on investment and financial performance. It presents total capital investment, direct and indirect operating expenses, total OPEX, levelized production cost, cost of carbon abatement, net present value, internal rate of return, and payback period. All values are taken from the backend's technoEconomics and financials results and filtered by the current access level.

Overall, the KPI column functions as a persistent analytical sidebar that remains visible next to the breakeven chart, giving analysts an immediate overview of project viability while still allowing deeper inspection through the expandable detail sections.

4.6. Guidelines for Extension and Future Development

The SAFAPAC frontend has been structured to allow straightforward extension whenever new inputs, outputs, or interface variations are required. Most functional additions begin with modifying the TEA input layer. New variables can be incorporated by updating BiofuelForm.js, adjusting the available unit options, and extending the normalisation helpers that prepare values for the backend. This ensures that new parameters are displayed consistently and transmitted in the correct canonical format.

Enhancing the analytical output follows a similar pattern. The KPI display on the Analysis Dashboard is driven by a central kpiGroups structure. Adding new KPIs simply requires defining them within this structure and, when necessary, applying feature gating through AccessContext so that only certain access levels are able to view them. This keeps the application flexible while remaining aligned with role-based or demo-mode constraints.

Layout and presentation changes are also easily managed due to the separation between global and TEA-specific structures. Updates that affect the overall application shell, such as adjustments to navigation, spacing, or visual themes, should be made in Default.js and custom.css. Changes that apply only to the analysis environment should be made directly within the layout block inside AnalysisDashboard.js.

Feature-access rules are configured through access.json. Updating this file allows developers to redefine which features belong to each access level, and the interface will respond accordingly through hasAccess checks that wrap conditional UI elements.

Visual consistency across themes is maintained by relying on the theme colours provided by ThemeContext and the CSS variables defined in custom.css. Using these shared sources avoids inconsistencies and prevents hard-coded colours from drifting across different parts of the interface. The changes in instruction are summarised in the Table 3

Table 3. Instruction Summary

Area of Extension	What to Modify	Purpose
Adding new TEA inputs	Update BiofuelForm.js, extend unit options, adjust normalisation helpers	Ensures new parameters are captured correctly and sent to the backend in canonical units
Adding new KPIs or outputs	Edit kpiGroups in AnalysisDashboard.js, use AccessContext for gating if needed	Adds new analytical metrics and controls visibility by access level
Layout adjustments	Global changes in Default.js and custom.css, TEA-specific changes inside AnalysisDashboard.js	Maintains consistent layout across the app and custom behaviour in the TEA screen
Access level updates	Modify config/access.json and wrap relevant UI in hasAccess checks	Controls which features are visible to each access level
Styling and theme consistency	Use ThemeContext colors and CSS variables in custom.css	Keeps the UI consistent while avoiding hard-coded styling

5. Testing and Validation

5.1 Backend Testing

The Backend Testing phase focused on the structural integrity, security, and reliability of the API architecture. This phase ensured that the transition to the AWS RDS PostgreSQL environment supports robust data management and secure access control.

5.1.1 Authentication & Security Verification

We rigorously tested the security layer to ensure that proprietary data remains protected. Using the new JWT (JSON Web Token) implementation, we verified that:

- **Unauthorized Access Blocked:** API endpoints return `401 Unauthorized` for users without valid session tokens.
- **Token Expiration:** Sessions automatically expire after the set duration (30 minutes), forcing a re-login to prevent stale session hijacking.
- **Password Hashing:** Verified that user passwords are never stored in plain text, but are hashed using `bcrypt` before storage in the database.

5.1.2 Data Integrity & CRUD Operations

We validated the "Create, Read, Update, Delete" (CRUD) lifecycles to ensure no data corruption occurs during user sessions.

- **Project Isolation:** Confirmed that User A cannot view or modify User B's projects (Multi-tenancy enforcement).
- **Scenario Limits:** Verified the business logic that prevents users from deleting the last remaining scenario in a project, preventing "orphaned" projects.
- **Data Persistence:** Confirmed that inputs saved in a session persist correctly in the PostgreSQL database and remain available upon logout/login.

5.1.3 Error Handling and Stability

To ensure system resilience, we performed "Negative Testing" (intentionally sending bad data) to verify the system recovers gracefully without crashing.

Table 4. Infrastructure Test Summary

Test Category	Test Description	Outcome	Status
Security	Attempt to access /projects without login token	Access Denied (401)	**PASS**
Data Type Safety	Send text ("ABC") into a numeric field (Capacity)	Request Rejected (422)	**PASS**
Business Logic	Attempt to create a 4th Scenario (Max limit is 3)	Action Blocked (400)	**PASS**
Database Sync	Create Project via API -> Check AWS RDS Table	Record Found	**PASS**

Curl

```
curl -X 'POST' \
  'http://localhost:8000/api/v1/projects' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "projectName": "HEFA-Test",
  "initialProcessId": 1,
  "initialFeedstockId": 1,
  "initialCountryId": 1
}'
```

Request URL

```
http://localhost:8000/api/v1/projects
```

Server response

Code	Details
403 *Undocumented*	Error: Forbidden

Response body

```
{
  "detail": "Not authenticated"
}
```

Download

Figure 37. API Security Layer blocking unauthorized access to project data.

```
safapac_db=> SELECT * FROM feedstock;
 id |          name           | carbon_content_kg_c_per_kg | energy_content_mj_per_kg | ci_ref_gco2e_per_mj | price_ref_usd_per_unit
| yield_ref
----+-------------------------+----------------------------+--------------------------+---------------------+------------------------
+-----------
  1 | UCO                     |                       0.77 |                       37 |                  20 |                    930
|    1.2048
  2 | Lignocellulosic Biomass |                       0.52 |                       18 |                   5 |                     50
|       4.5
  3 | Animal Manure           |                        0.4 |                       15 |                  15 |                     10
|       8.5
(3 rows)


safapac_db=> SELECT * FROM process_technologies;
 id |  name
----+--------
  1 | HEFA
  2 | FT-BtL
  3 | ATJ
  4 | CHJ
(4 rows)
```

Figure 38. Successful data persistence in AWS RDS PostgreSQL instance (via Command Prompt).

5.2 Sample Test Case: HEFA with UCO Feedstock

To validate the integrity of the system's calculation engine (`BiofuelEconomics`), we performed a comprehensive "Ground Truth" comparison. We took the validated engineering specifications provided by the research team (Reference: *Wave 1 - Spec IO*) and ran the exact same scenario through the SAFAPAC API.

This test confirms that the transition from manual spreadsheets to our automated, database-backed web platform preserves 100% mathematical accuracy.

5.2.1 Test Configuration

We configured a "HEFA-UCO Corrected Test" scenario using the new PostgreSQL database structure. Instead of manual text entry, the system now uses relational IDs to fetch standardized master data, ensuring consistency across all users.

Table 5. Key Input Parameters

1. Model Inputs		HEFA		
1.1 Conversion Plant	Plant total liquid fuel production capacity - value and unit	500 KTPA		
	Plant annual load hours	8000 hrs/yr		
	Conversion process carbon intensity - default value	20 gCO2e/MJ		
1.2 Feedstock & Utilities Data	Feedstock name/type	UCO	Hydrogen	electricity
	Feedstock price - value and unit	**930 USD/t**	**5,4 USD/kg**	**55 $/MWh**
	Feedstock carbon content			
	Feedstock carbon intensity - value and unit			20 gCO2e/kWh
	Feedstock energy content			
	Feedstock yield - value and unit	**1.21 kg UCO/kg_fuel**	**0.042 kg H2/kg_fuel**	**0.12 kWh/kg_fuel**

1.4 Product Data	**Product name/type**	**JET**	**DIESEL**	**Naphtha**
	Product price - value and unit	3000 $/t	1500 $/t	1000 $/t
	Product price sensitivity to carbon intensity - value and unit			
	Product carbon content	0.847	0.85	0.84
	Product energy content	43.8	42.6	43.4
	Product yield - value and unit	0.64 kg/kg	0.15 kg/kg	0.21 kg/kg
1.5 Economic Parameters	Discount rate	7%		
	Project lifetime	20 years		
	Total Capital Investment (TCI) @ reference plant production capacity	400 MUSD		
	Total Capital Investment (TCI) scaling law	0.6		
	Reference plant production capacity	500 KTPA		
	Working Capital (WC) / TCI ratio	15%		
	Indirect OPEX / TCI ratio	7.7%		

Table 6. Expected Outputs

2. Model Outputs				
2.1 Process Outputs	For each Feedstock1..N and Utility1..N	UCO	Hydrogen	electricity
	Total Consumption	605000 t/year	21000000 kg/year	60000000 kWh/yr
	For each Product1..N	JET	DIESEL	Naphtha
	Production	320000 t/year	75000 t/year	105000 t/year
	Carbon intensity			
	Carbon conversion efficiency			
	Total CO2 emissions			
2.2 Economic Outputs	Total Capital Investment	$400,000,000.00		

		UCO	Hydrogen	electricity
	Total direct OPEX and breakdown per feedstock and utility	$562,650,000.00	$113,400,000.00	$3,300,000.00
	Total Indirect OPEX	$30,800,000.00		
	Total OPEX	$710,150,000.00		
		JET	DIESEL	Naphtha
	Revenue amount	$960,000,000	$112,500,000	$105,000,000
	Total revenue	$1,177,500,000		
	Levelized Cost Of Production (LCOP) and breakdown	1,495.81$/ton		
	TCI	2.70%		
	Cost of Feedstocks	77.10%		
	Cost of Hydrogen	15.50%		
	Cost of Electricity	0.50%		
	Cost of Indirect OPEX	4.20%		
	Levelized Cost Of Carbon Abatement (LCCA)			
	Net present value (NPV)	$3,532,017,806		
	Internal rate of return (IRR)	119.7%		
	Payback period	1 year		

5.2.2 Verification of Results

The API output was compared directly against the reference Excel model. The results show an exact match across all critical financial and operational metrics, confirming that the migration to AWS RDS has not introduced any calculation errors.

Table 7. Validation Matrix (Excel Reference vs. SAFAPAC API)

Metric	Engineering Reference Value (Excel)	SAFAPAC API Result (AWS Cloud)	Variance	Status
Total Production	500,000 tonnes/yr	500,000 tonnes/yr	0.00%	MATCH
Jet Fuel Output	320,000 tonnes/yr	320,000 tonnes/yr	0.00%	MATCH
Feedstock Cost	$562,650,000	$562,650,000	0.00%	MATCH
Hydrogen Cost	$113,400,000	$113,400,000	0.00%	MATCH
Total Indirect OPEX	$30,800,000	$30,800,000	0.00%	MATCH
Total Annual OPEX	**$710,150,000**	$710,150,000	0.00%	MATCH

Total Capital Inv.	**$400,000,000**	**$400,000,000**	**0.00%**	**MATCH**

5.2.3 System Output Evidence

Below is a snapshot of the actual JSON response generated by the SAFAPAC API during this test. This response is generated in real-time, utilizing the `BiofuelEconomics` service layer and stored securely in the PostgreSQL database.

JSON


```
{
  "technoEconomics": {
    "process_technology": "HEFA",
    "feedstock": "UCO",
    "total_capital_investment": 400,
    "total_opex": 710150000,
    "carbon_intensity": 0.5558,
    "product_breakdown": {
      "jet": 320000,
      "diesel": 75000,
      "naphtha": 105000
    },
    "opex_breakdown": {
      "feedstock": 562650000,
      "hydrogen": 113400000.0,
      "indirect_opex": 30800000
    }
  },
  "financials": {
    "npv": 3072929501.56,
    "irr": 1.197,
    "payback_period": 1
  }
}
```

Figure 39: Live API Response Payload (Truncated for readability)

Code | Details

200

Response body


```
{
  "technoEconomics": {
    "process_technology": "HEFA",
    "feedstock": "UCO",
    "LCOP": 1495.8143405946046,
    "total_capital_investment": 400,
    "production": 500000,
    "feedstock_consumption": 605000,
    "total_opex": 710150000,
    "total_co2_emissions": 12099999999.999722,
    "carbon_intensity": 0.5558618155089913,
    "utility_consumption": {
      "hydrogen": 21000000,
      "electricity": 60000000
    },
    "product_breakdown": {
      "jet": 320000,
      "diesel": 75000,
      "naphtha": 105000
    },
    "carbon_conversion_efficiency": 30.266537869843656,
    "opex_breakdown": {
      "feedstock": 562650000,
      "hydrogen": 113400000.00000001,
      "electricity": 3300000,
      "indirect_opex": 30800000
    }
```


Download

Figure 40. Swagger UI showing successful execution of the HEFA calculation module against the AWS Database.

7. Conclusion

7.1 Next Steps for Development and Deployment

The next phase of development centres on strengthening reliability, improving maintainability, and operationalizing the platform for sustained long-term use. A key priority is the full migration from the current mock JSON-based persistence layer to a production-grade PostgreSQL database on AWS RDS. This transition ensures relational integrity, indexing, transaction safety, and durability as the system scales to a broader user base.

Authentication and access control will also be upgraded. The current CSV-based login mechanism will be replaced with a complete JWT authentication layer, with optional extensions for corporate Single Sign-On and multi-factor authentication. This will ensure that data access remains secure as external research teams, industry partners, and government agencies begin interacting with the platform.

To support continuous improvement, the system will adopt a robust CI/CD pipeline. By containerizing the backend and automating build and deployment processes, SAFAPAC can ensure consistent, versioned releases across development, staging, and production environments. This pipeline will allow faster iteration, automated testing, and structured governance over future code changes.

User Acceptance Testing (UAT) will expand significantly. Analysts, researchers, and external Airbus collaborators will evaluate the platform on real-world SAF projects, enabling refinement of UX flows, calculation clarity, system stability, and overall computational performance. Their feedback will shape the refinement of the TEA interfaces, output visualizations, and scenario comparison tools.

New SAF pathways will be integrated alongside expanded feedstock databases and enhanced carbon pricing logic to reflect emerging regulatory frameworks.

7.2 Future Scalability & Enterprise Architecture

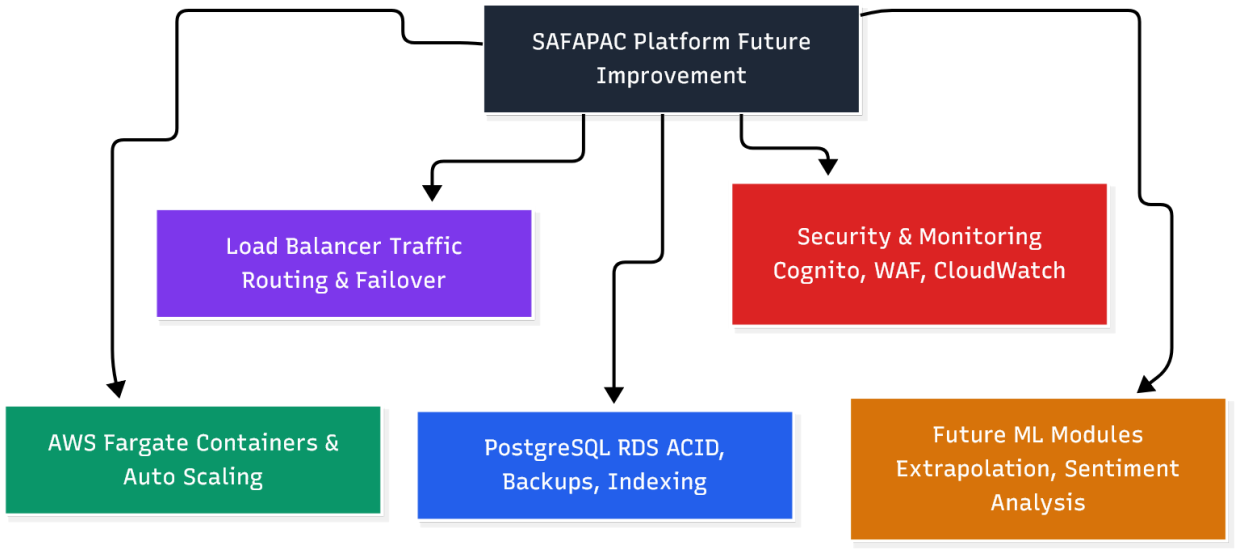


Figure 41. Overview of future improvements.

The long-term vision for SAFAPAC is to evolve into a scalable, cloud-native analytical engine capable of supporting national-level Sustainable Aviation Fuel (SAF) decision-making. As adoption increases across agencies, industry partners, and research institutions, the platform must support heavier computation, larger datasets, concurrent users, and more complex modelling capabilities. The future architecture is designed around four strategic pillars: robust data infrastructure, containerized compute, elastic scalability, and enterprise-grade security, with upcoming extensions into machine learning and advanced analytics.

To achieve this, SAFAPAC will transition toward a more resilient and modular backend foundation. A fully managed AWS RDS PostgreSQL environment will replace the temporary JSON persistence layer, enabling ACID-compliant transactions and centralized master-data storage for feedstocks, process references, pricing tables, and scenario outputs. High availability, automated backups, and indexing ensure that the system can scale without sacrificing data integrity or performance. This provides the unified data backbone needed for large-scale adoption across government and industrial stakeholders.

On the compute layer, containerization via AWS Fargate will standardize runtime environments and eliminate manual infrastructure management. By running backend services in isolated, reproducible containers, the system benefits from predictable performance across development, staging, and production.

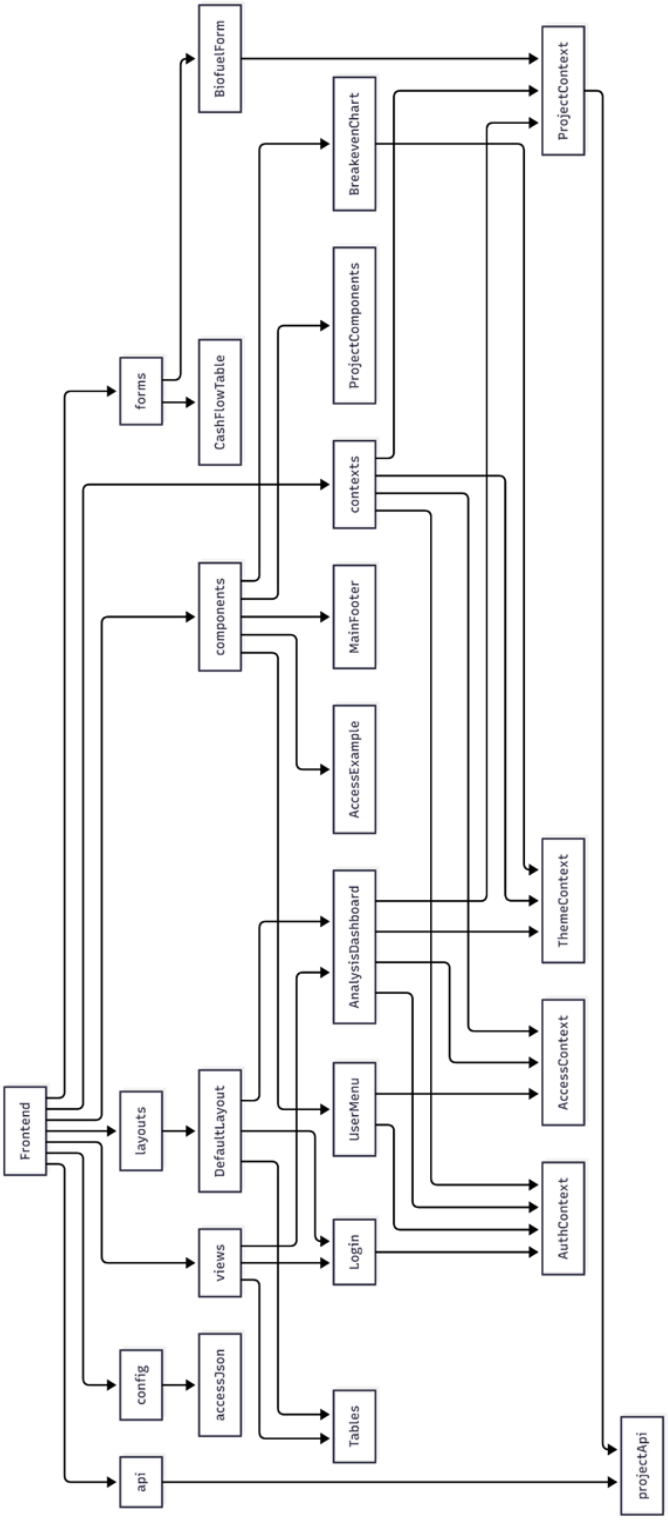
To maintain responsiveness under heavier user loads, SAFAPAC will adopt an AWS Application Load Balancer (ALB) to intelligently distribute API traffic. ALB will provide:

- Smart routing across multiple backend containers
- Automatic failover to preserve uptime
- Centralized HTTPS termination for simplified security
- Consistent performance during multi-scenario simulations This guarantees that analysts, researchers, and executives experience smooth performance even during intensive model runs.

In addition to infrastructure upgrades, SAFAPAC platform will expand its analytical capabilities through machine learning integration. Future modules will include:

- Predictive TEA extrapolation models that learn from historical user inputs and engineering datasets to automatically estimate missing parameters or simulate likely outcomes under uncertainty.
- Sentiment analysis models that evaluate qualitative user input, policy text, or market reports to help decision-makers interpret broader context and stakeholder perception. Adaptive recommendation engines that guide users toward optimal configuration choices based on learned patterns in successful scenarios.

Appendices


Frontend
api
config
access3son
layouts
views
DefaultLayout
components
forms
BiofuelForm
CashFlowTable
contexts
ProjectComponents
BreakevenChart
ProjectContext
MainFooter
AccessExample
AnalysisDashboard
UserMenu
Login
Tables
ThemeContext
AccessContext
AuthContext
projectApi