

University of Glasgow | School of Computing Science

Opus Equi: A Desktop Application for Financial Time-Series Forecasting

Dylan Fitzsimmons

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at the University of Glasgow

17th of September 2025

Abstract

Opus Equi is a full-stack desktop application – C#/WPF front end over a Python FastAPI service backend – for financial time-series forecasting. It ingests multiple CSVs, concatenates and cleans the files, then trains and serves machine learning models (Quantile Random Forest and per-quantile XGBoost) to produce median forecasts with accompanying uncertainty intervals; results are returned as JSON and visualised in OxyPlot, while timestamped dataset snapshots and model artefacts are banked for reproducibility. The system delivers an end-to-end, offline workflow from raw data to interpretable charts under MVVM, with clear separation of routes, services, utilities and models. Evaluation through unit and integration tests across the upload–train–predict path demonstrates functional correctness and practical responsiveness on representative datasets, yielding coherent forecasts with defensible uncertainty bands suitable for exploratory analysis and extension to richer financial workflows.

The system architecture comprises a set of loosely coupled modules with well-defined interfaces – a modular structure that permits rapid extension in future builds, whether that be integrating alternative forecasting models or UI components. The design follows established separation of concerns and inversion of control principles so that core services, models and presentation layers are isolated behind stable contracts.

Abbreviations

AI – Artificial Intelligence

ARIMA – Autoregressive Integrated Moving Average

BI – Business Intelligence

CLI – Command Line Interface

CME – Chicago Mercantile Exchange (CME Group)

CRPS – Continuous Ranked Probability Score

DSWS – Datastream Web Service

DTO – Data Transfer Object

ECMA – European Computer Manufacturing Association

ETS – Error, Trend, Seasonality

GUI – Graphical User Interface

IEC – International Electrotechnical Commission

IEEE – Institute of Electrical and Electronics Engineers

ISO – International Organization for Standardization

LSEG – London Stock Exchange Group

MAE – Mean Absolute Error

ML – Machine Learning

MVVM – Model View View Model

PNG – Portable Network Graphics

QRF – Quantile Regression Forests

REST – Representational State Transfer

RMSE – Root Mean Squared Error

SDK – Software Development Kit

UML – Unified Modelling Language

XGB – XGBoost

Contents

Chapter 1 Introduction 1

1.1 Context 1

1.2 Problem Statement 1

1.3 Aim and Objectives 2

1.3.1 Aims 2

1.3.2 Objectives 2

1.4 Document Roadmap 3

Chapter 2 Analysis and Requirements 4

2.1 Background on the Industry 4

2.1.1 Financial Markets 4

2.1.2 Financial Time-Series Modelling 5

2.2 Background on Similar Systems and Platforms 6

2.3 Synthesis and Implications for Design 7

2.4 Stakeholders and User Tasks 8

2.4.1 Stakeholder Persona 8

2.4.2 User Tasks 8

2.5 Requirements and MoSCoW Prioritisation 8

2.6 Constraints and Risks 9

Chapter 3 Design and Implementation 10

3.1 Design Methodology 10

3.2 Data Science – CSV Cleaning and Concatenation 11

3.3 Machine Learning – Nonlinear Forecasting 12

3.3.1 XGBoost with Pinball Loss 12

3.3.2 Quantile Random Forest 13

3.3.3 Shared Feature Spine and Forecasting Loop 13

3.4 System Architecture 13

3.4.1 Backend 13

3.4.2 Front End 15

Chapter 4 Testing and Evaluation 17

4.1 Scope and Methodology 17

4.2 Unit Tests 17

4.2.1 Backend Unit Tests 17

4.2.2 Frontend Unit Tests 18

4.3 Integration Tests 18

4.4 Acceptance Tests 18

4.5 User Tests 19

Chapter 5 Conclusion 20

5.1 Achievements 20
5.2 Challenges and Limitations 20
5.3 Future Development 21

References 22

Appendix A MoSCoW Prioritisation 29

Appendix B Risk Register 31

Appendix C Requirements Traceability Matrix 33

Appendix D Unit Tests 35

Appendix E Integration Tests 38

Appendix F Acceptance Test Matrix 39

Appendix G User Test Matrix 42

Appendix H System Architecture 43

Appendix I Frontend UI Wireframe 44

Appendix J Actual Frontend UI 45

Appendix K Data and Storage Flow 46

Appendix L Backend UML 47

Appendix M Frontend UML 48

Chapter 1 Introduction

1.1 Context

In the commerce of commodities the future is bought and sold by contract, yet it is the past – noisy, irregular, and often capricious – that supplies the evidence by which tomorrow's market price is determined. Time-series forecasting is the art and discipline of extracting structure from temporal data – trend, seasonality, geopolitics, and dependence on short-term changes – to issue predictions fit for decision under uncertainty [1][2]. For practitioners in financial and commodity domains, the desideratum is not merely a point forecast but a distributional view: intervals that acknowledge heteroskedasticity and model fallibility, and that can be interrogated visually and reproduced with sobriety.

This dissertation presents *Opus Equi*, a name derived from the Latin for 'work of the horse' or 'horse's work', invoking the English term 'work horse' – implying heavy lifting. The heavy lifting is indeed conducted by the software, converting a complex task into a simple process of analysis. This is achieved using a full-stack desktop application that carries an analyst from CSV to chart: a C# WPF client (using the Model-View-ViewModel paradigm) orchestrates a Python FastAPI service which ingests heterogeneous CSV exports, concatenates and cleans these datasets into a canonical modelling table, trains and serves uncertainty aware learners, and returns compact JSON payloads for interactive visualisation in OxyPlot [3]-[6][8][12]. The design is purposefully local and reproducible: datasets are snapshotted; trained models persisted as artefacts; configuration is explicit; and interfaces are strictly typed. In short, the system weds modern desktop ergonomics to a small, testable inference service, keeping the machinery plain to the eye and tractable to the hand.

The modelling side adopts tree-based methods that have proved effective for tabular and potentially nonlinear dynamics: gradient boosted decision trees (XGBoost) for sharp point (and per-quantile) predictions, and Quantile Regression Forests (QRF) to estimate conditional quantiles without parametric assumptions, thereby furnishing prediction intervals by construction [5][6]. Such ensembles handle interactions and monotone effects with aplomb; they regularise naturally; and admit efficient, production grade implementations [5][7][9]. Employed on the software side is infrastructure consonant with industrial practice: WPF/MVVM for testable desktop composition; REST-style HTTP for process boundaries; typed request/response contracts; and explicit logging and configuration for audit and maintenance [3][4][10][11][15][17].

1.2 Problem Statement

Analysts of financial time-series, particularly in commodities, frequently labour under three impediments that conspire to erode rigour and repeatability:

1. Fragmented, heterogeneous data. Historical prices arrive as multiple CSV files with discordant schemas, timestamp granularities, and missing observations. Manual collation in spreadsheets is error prone and unreproducible [1][16].

2. Opaque processing and inconsistent modelling. Ad hoc scripts conceal the lineage of transformations; model training and inference are not bound to artefacts and cannot be reconstructed faithfully; uncertainty is elided or inferred post hoc [1][2][7].

3. Tooling gap at the desktop. Notebooks demand coding fluency; cloud services impose operational overhead; few local tools offer an end-to-end path from ingestion and concatenation through cleaning, model training, interval prediction, and publication grade visualisation [3][10][12].

Opus Equi addresses this gap with a two-tier architecture – WPF/.NET 8 front end and FastAPI backend – joined by a narrow, typed HTTP surface:

- POST /upload to accept and snapshot multiple CSVs;
- POST /train/{model} to fit a named learner and persist its artefact;
- GET /predict/{model}/{date}?with_history=true to return history with a median forecast and upper/lower bounds.

The pipeline performs schema reconciliation, temporal alignment, coercion of types, and basic imputation; the service encapsulates learners behind a common BaseForecastModel interface, serialises fitted estimators, and emits JSON data transfer objects (DTOs) that the client parses into strongly typed models for plotting [3][4][6][8][12][16][19].

1.3 Aim and Objectives

1.3.1 Aims

To design and implement a local, reproducible forecasting application that transforms heterogeneous CSV time-series data into uncertainty aware predictions of financial/commodity prices, rendered interactively in a desktop client.

1.3.2 Objectives

1. Architecture (process separation). Implement a WPF/.NET 8 client (MVVM) and a FastAPI microservice on localhost, communicating via REST with schema validated JSON, conforming to HTTP/REST constraints (resource identification, statelessness) [3][4][10][11][15].

2. Data engineering. Build ingestion that accepts multiple CSVs, performs concatenation, schema harmonisation, time zone policy, and resampling/alignment to a canonical frequency; produce timestamped snapshots of merged datasets for provenance; log transformations for audit [1][16][17][19].

3. Model abstraction and persistence. Define a BaseForecastModel with fit(df) and predict(df, target_date); implement QRF and XGBoost (per-quantile) variants; persist artefacts (*.joblib) with deterministic paths; maintain a model registry for discovery and configuration [5][6][7][9].
4. Serving and contracts. Expose /upload, /train/{model}, and /predict/{model}/{date} endpoints with typed DTOs and explicit error semantics; ensure idempotence where feasible [10][14].

5. Visual analytics. Render historical series and prediction intervals in OxyPlot (PlotModel, LineSeries, AreaSeries); enforce legible axes and consistent date handling; support exportable figures [12].

6. Quality attributes. Provide structured logging (category, level, correlation IDs), light weight configuration, and unit/integration tests across the upload-train-predict path; document deterministic setup (Conda environment spec, .NET SDK) [3][4][17][18][19].

7. Deferred features. Reserve seams for a futures pricing module (cost-of-carry) and local authentication (login/token plumbing).

1.4 Document Roadmap

Chapter 2 – Analysis and Requirements. Background, related work and similar systems, design implications, shareholders and user tasks, MoSCoW prioritisation, constraints and risks.

Chapter 3 – Design and Implementation. Design methodology, data processing, machine learning models, and system architecture.

Chapter 4 – Testing and Evaluation. Scope of testing and methodology, unit tests, integration tests, acceptance tests, user tests.

Chapter 6 – Conclusion. Achievements, challenges and limitations, discussion of future work.

Chapter 2 Analysis and Requirements

2.1 Background on the Industry

2.1.1 Financial Markets

Financial markets – e.g., equities (stocks), debt instruments (government bonds and fixed-income products), foreign exchange (currencies), commodities (physical goods such as grains, oil, etc.), and derivatives (contracts derived from assets in the previously stated markets) – are foundational to the global economy and yet are often poorly understood due to their complexity and the opacity of the industry [19][20].

Commodities are grouped into categories, the most frequently traded being: agricultural, which includes grains and oilseeds, softs; metals, which includes precious metals, base or industrials; and energy, which includes crude oil, natural gas, and coal [22][23]. The underlying asset, in this case commodities, are listed on commodity exchanges and priced according to market forces [24][25]. Using soybeans as an example, external forces that factor into the price are grouped by supply-side (e.g., weather anomalies such as El Nino and La Nina), demand-side (e.g., import volume of large consumers such as the Chinese), macroeconomic (e.g., USD exchange rate and interest rates, tariffs to high-consumer nations such as China), and the influence of external markets (e.g., shipping/freight rates, and the price of substitutes or complementary goods, such as corn or rapeseed oil) [26]-[31]. Purchasing commodities can be achieved through direct means with a supplier for a negotiated rate, or more commonly, in larger quantities via brokers or exchanges [24][25]. In the latter case, the spot price reflects the price paid more accurately and is usually delivered immediately or within a short delivery window. The spot price, for clarity, typically reflects the midpoint of bids (demand-side, what buyers are willing to pay) and asks (supply-side, what sellers are willing to accept) from brokers on commodity exchanges. It is therefore possible to purchase or sell between this variation in prices, which is known as the spread [20][32]. There is however another method to trade commodities, a market which doesn't always require the purchaser to accept delivery of physical goods – derivatives [21].

Derivatives, as the name suggests, are contracts derived from the price of the underlying asset (e.g., a commodity) which grant certain privileges to the holder depending on the type. Additionally, other factors impact prices such as supply and demand, and market and asset volatility. The most common types are futures (a standardised agreement to buy or sell an asset at a fixed price at a future date), and options (which give the holder the right but not the obligation to buy or sell an asset at a fixed price at a later, sometimes variable date). These are known as exchange traded derivatives and are standardised, therefore heavily traded (liquid) and used extensively for speculation [19][21][24][25][33]. Other common types are swaps and forwards, however those are known as over-the-counter and are agreed upon between two parties, underwritten by investment banks. These contracts are harder to benchmark and test against due to the lack of publicly available pricing data, and as such are beyond the scope of this project [19][34]. There are also exotics and derivatives of derivatives such as options on futures, however with limited information it is harder to verify the accuracy of models against a common benchmark [19]. The most liquid are futures contracts on WTI crude oil, gold, corn, soybeans, and natural gas [24][25][35]. These are bought and sold by various

parties ranging from speculative traders to industrials such as airlines and to the very people that produce the goods for sale [37]. They provide a vehicle to hedge against risk; in the case of agricultural products like corn and soybeans, that could be adverse weather that impacts crop yields (bad weather in the American Midwest lowers yields across states, prices rise as supply is short but demand stays strong – buyers can lock in lower prices months in advance) [26][36][37]. Therefore, commodity markets are extremely important both politically and financially, and their subsequent derivatives are highly complex in nature [38].

2.1.2 Financial Time-Series Modelling

In the traffic of commodities, prices are not merely numbers in procession but the visible trace of inventories, logistics, seasonality, convenience yields, and capital flows. Classical expositions – Working's theory of storage and the cost-of-carry relation – describe how spot, futures, storage and scarcity codetermine term structures and basis, and why pricing regimes may shift abruptly under stress [40][41][42]. Modern surveys show that commodity data is commonly heteroskedastic, regime dependent and heavy tailed; it bears structural breaks from policy shocks and harvests, as well as calendar peculiarities from trading holidays and delivery conventions [43]. Any practical forecasting system therefore requires both discipline in data preparation – schema harmonisation, robust timestamp handling, reconciliation of irregular frequencies – and models that admit nonlinearity and uncertainty quantification rather than point estimates alone.

Two grand traditions shape the field. The first is stochastic time-series: Autoregressive Integrated Moving Average (ARIMA) and exponential-smoothing families, cast either in the Box-Jenkins difference autoregression idiom or in state-space form with Kalman filtering and likelihood-based estimation [2][38][39][44][45]. These methods enforce parsimony and make assumptions explicit (stationarity, invertibility, error structure). The second is the tabular-learning tradition, in which the series (often with engineered lags and calendrical features) becomes a supervised learning problem for flexible regressors. Gradient-boosted decision trees (GBDT) supply strong nonlinear approximants through additive function expansions; the seminal formulation – Friedman's gradient boosting – yields second order updates, shrinkage, and tree structure regularisation that have proved adept on structured data [46].

For distributional forecasts, two families are especially pertinent. Quantile regression estimates conditional quantiles by minimising the pinball loss, furnishing lower/upper bounds without parametric distributional commitments [47][48]. In the ensemble realm, Quantile Regression Forests (QRF) extend random forests to conditional quantiles by aggregating leaf distributions rather than means, thereby offering calibrated intervals even under complex interactions [49].

To promise intervals is to submit to sterner evaluation. Modern probabilistic forecasting distinguishes calibration (statistical consistency: events forecast at level α occur with frequency α) from sharpness (concentration of the predictive distribution) and demands proper scoring rules that reward both [49][50]. The continuous ranked probability score (CRPS) measures the distance between predictive and empirical distributions [51]; for simple interval forecasts one may employ Winkler's score, which balances narrowness against coverage [52]. Point forecast metrics – MAE, RMSE and the scale-free MASE – remain workhorses, but MAPE is known to misbehave under zeros and small denominators [53]. Backtesting must respect temporal order: rolling-origin (or 'walk-forward') evaluation and blocked cross-validation avert look-ahead bias and leakage

that would flatter the modeller unduly [54][55]; comparative accuracy may be adjudged with the Diebold–Mariano test [56]. The M-competitions (M4/M5) have further clarified that hybrid or ensemble methods, tempered by careful data handling, dominate simplistic single-model dogma in practice [57][58].

With raw data, historical prices arrive as CSV exports of varying pedigree. Reproducible pipelines must therefore perform: schema harmonisation (canonical column names and types); temporal alignment (coherent frequency, time zone, and business day calendars); missing data policy (interpolation/forward-fill under explicit constraints); deduplication; and concatenation into a single modelling table. The service must retain timestamped snapshots for provenance and audit and persist trained artefacts for deterministic reuse. These are not fripperies but the girders of scientific work; codified data hygiene, à la Wickham's 'tidy data', reduces anarchy to order and makes downstream inference legible [59]. In Python, pandas and scikit-learn provide the numerical substrate and model APIs; in the .NET client, MVVM enforces separation of concerns, enabling testable composition and latency-free interaction [3][11][60].

2.2 Background on Similar Systems and Platforms

Refinitiv's Workspace/Eikon paired with Datastream provides a decades deep, global macro-financial time-series repository alongside rich desktop charting – most notably Datastream Charting and Chart Studio – through which analysts assemble multi-series charts, apply studies, annotate, and export artefacts; programmatic access is furnished by the Datastream Web Service (DSWS) via SOAP/REST/.NET and an official Python client governed by entitlements [65]66][67]. In practice, the terminal emphasises data access and charting, while statistical forecasting is typically executed in Excel plugins or external Python/R over DSWS rather than as a terminal application [65][66][67]. Bloomberg's BQuant Desktop similarly integrates Python/Jupyter on its terminal, provides data access through BQL and related APIs, and allows quants to publish notebooks as apps into Launchpad – powerful for research and team sharing, albeit still fundamentally notebook centric rather than a fixed desktop forecasting workflow [68][69]. FactSet offers a comparable pattern: time-series delivery through multiple APIs (e.g., Global Prices API and Real-Time Time Series API) aimed at workstation and web clients, with certain analytics (rollups, currency transforms, window functions) computed server-side prior to delivery; forecasting is again commonly carried out in Excel or Python atop these APIs [70][71][72]. Against this backdrop, Opus Equi deliberately diverges by making probabilistic forecasting an explicit product feature (with native uncertainty intervals) within a local, auditable pipeline rather than deferring modelling to notebooks or spreadsheets.

Public cloud offerings foreground automation and scale. Amazon Forecast supplies managed probabilistic forecasting with classical and deep 'recipes', built in backtesting, and hosted endpoints consumable via SDKs, but it remains cloud bound and partially blackbox in configuration and feature handling [73][74][76]. Azure Automated ML for time-series provides auto-featurisation (lags, holidays), automated model selection, and one click deployment through Azure ML Studio/CLI/SDK, trading rapid prototyping for platform lock in and opacity in the search process [74][78][87]. Google Vertex AI Forecasting extends AutoML to tabular/TS with strong pipeline integration and MLOps hooks, again as a managed service with attendant governance and cost considerations [76][79]. By contrast, Opus Equi's localhost architecture prioritises offline reproducibility and inspectable pipelines.

On the desktop, Microsoft Excel and Power BI offer immediate forecasting through FORECAST.ETS (triple exponential smoothing with configurable seasonality and confidence intervals), a workflow that is widely understood yet harder to audit and version rigorously than code driven systems [77][80]. At the econometrics end, MATLAB Econometrics and EViews provide interactive GUIs and scriptable workflows for ARIMA/state-space/VAR/GARCH models and forecasting, with strong diagnostics and pedagogy; these are licensed environments and not naturally aligned with a C# client to Python service product architecture out of the box [81]-[84]. Opus Equi, in contrast, packages a repeatable product workflow (upload-train-predict) with testable seams and a fixed DTO between client and service.

Within the open-source ecosystem, statsmodels (ARIMA/SARIMAX/state-space) remains a transparent baseline with rigorous diagnostics [80][84], while Prophet provides additive trend/seasonality/holiday models suited to analyst in the loop forecasting at scale [65]. Furthermore, sktime supplies a unified interface and backtesting/pipeline constructs across model families, and Darts layers a user-friendly API over classical methods, tree ensembles, and deep architectures with probabilistic outputs [66][67]. At the research frontier, GluonTS (probabilistic deep time-series) and Uber Orbit (Bayesian forecasting) emphasise distributional predictions and proper scoring rules [88]-[92]. These are libraries rather than finished applications; Opus Equi productises a narrow, opinionated subset (QRF/XGBoost with quantile loss; Fourier/lags) behind a persistent REST seam and a desktop UI.

Across these categories the trade-offs are consistent. For data and access, terminals/APIs dominate breadth and entitlement control; cloud services add turnkey pipelines; BI/econometrics smooth the GUI path; Opus Equi intentionally assumes local CSVs to simplify governance [65][70][73][77][81]. Regarding modelling and uncertainty, terminals generally rely on user code for interval estimation; BI tools expose basic confidence bands; cloud AutoML produces probabilistic outputs; Opus Equi natively surfaces 5%/50%/95% quantiles as a first-class DTO and shaded band [67]-[69][73]-[80]. For reproducibility, cloud services are reproducible within their respective stacks; terminals organise data lineage but not necessarily users' modelling code; GUI tools can be scripted yet are less contract driven; Opus Equi emphasises snapshots, persisted artefacts, fixed JSON schemas, and end-to-end tests [65][73][74][76][77][81]. Finally, in operations, terminals and BI are chiefly desktop applications, while cloud AutoML is managed; Opus Equi is a two process, local architecture (WPF client to FastAPI) designed for a single user research tool and straightforward academic evaluation [68][73][76][76].

2.3 Synthesis and Implications for Design

The literature thus counsels three design precepts. First, respect the data-generating process: conciliate irregularities and preserve lineage; prefer walk-forward validation; measure both accuracy and uncertainty [51]-[55]. Second, choose models proportionate to data and task: tree ensembles with quantile mechanisms (QRF; boosted quantiles) supply robust distributional forecasts for tabular commodity series without the operational bulk of deep nets [46][49][68]. Third, enshrine reproducibility in the architecture: typed boundaries, explicit configuration, environment capture, snapshotting, and persisted artefacts [60][61]. These precepts inform the concrete choices reported in Chapter 3: a WPF/MVVM client over a FastAPI service; /upload, /train, /predict as the whole public surface; disciplined CSV consolidation; and uncertainty aware ensembles rendered in OxyPlot.

2.4 Stakeholders and User Tasks

2.4.1 Stakeholder Persona

Analyst (primary user) – a practitioner responsible for exploring and forecasting financial/commodity price series on a workstation within a desktop environment. The analyst's chief desiderata are: a local, reproducible pipeline from CSV inputs to interpretable forecasts; uncertainty aware outputs (prediction intervals rather than mere point forecasts); and responsiveness sufficient for exploratory 'what if' analysis.

2.4.2 User Tasks

- T1: Ingest and reconcile data. Select one or more CSV files exported from diverse sources; the system validates headers, concatenates chronologically, enforces schema and types, and writes a timestamped snapshot of the merged dataset [16][59].
- T2**:** Train a forecasting model. Choose a model (e.g., QRF or XGB); the service loads the latest merged dataset and fits the learner, persisting the fitted artefact for deterministic reuse [9][60].
- T3: Predict at a target date. Request a forecast for an ISO date; the service returns a compact JSON DTO: historical series, median forecast, and upper/lower bounds (quantiles), which the client renders as lines plus an area band in OxyPlot [12].
- T4: Inspect and iterate. Change the model or date; plot quickly without repeated ingestion unless files change; errors (e.g., malformed CSV, missing column) are reported in plain terms.
- T5: Audit. Retrieve the snapshot and model artefact that produced a chart; confirm configurations used (model registry entry, parameters).

2.5 Requirements and MoSCoW Prioritisation

The requirements are stated as functional (F) and non-functional (N) and prioritised using MoSCoW – with M (must), S (should), C (could) and W (will not) marked accordingly. Below in table 1 is a summary of the full document stored as appendix A. This is complemented with a requirements traceability matrix (appendix C) which links each requirement to its design/implementation element (discussed in chapter 3) in addition to its test artefact (discussed in chapter 4). ISO/IEC/IEEE guidance has been adopted on requirements quality (unambiguous, testable, traceable) and software product quality (reliability, usability, maintainability, portability) [93][94].

Priority	Requirements (IDs)
Must	F-M1, F-M2, F-M3, F-M4, F-M5; N-M1, N-M2, N-M3, N-M4
Should	F-S1, F-S2, F-S3, F-S4; N-S1, N-S2
Could	F-C1, F-C2; N-C1
Will Not	F-W1, F-W2

Table 1 - MoSCoW Summary

2.6 Constraints and Risks

A full risk register table can be found in the appendix listed as B.

- Time: single developer effort over approximately 9 weeks at roughly 30 hours/week; scope controlled via MoSCoW and explicit deferrals (authentication, futures pricing).
- Platform: Windows desktop; .NET 8/WPF client and Python 3.10/FastAPI service on localhost; no external services; no GPU required.
- Data: heterogeneous CSVs of financial/commodity prices; univariate series per request; business-day frequency; no third-party APIs; size bounded by workstation memory.
- Environment and dependencies: Conda environment pinned by environment.yml; .NET SDK pinned; model artefacts persisted with joblib; reproducibility depends on version pinning and snapshot discipline.
- Architecture: two processes joined by a narrow REST boundary only (/upload, /train/{model}, /predict/{model}/{date}); no database; local file system for snapshots and artefacts.
- Quality model: Prioritised non-functionals: reproducibility, reliability, usability, and performance per ISO/IEC 25010.

Chapter 3 Design and Implementation

3.1 Design Methodology

This project adopts a contract first, standards aware methodology that privileges low coupling and high cohesion. The public surface was fixed early as a narrow REST boundary documented by OpenAPI [97]; internal modules then evolved behind that seam. This stabilises collaboration by interface while allowing independent evolution of presentation and modelling concerns, consistent with Fielding's uniform interface constraint and modern interface description practice [10].

Design work and rationale were recorded using architecture views rather than monolithic prose: a small set of logical/process/development/physical views organise decisions and keep assumptions explicit. This follows ISO/IEC/IEEE guidance for architecture description and is congenial to Kruchten's 4+1 model, which is employed to structure the remainder of this chapter [95][96].

Cohesion is enforced within roles; coupling exists only at intentional seams. Two seams matter: the HTTP contract (typed JSON DTOs) and the model interface (a stable fit/predict policy shared by all estimators). Within those seams, components are shaped by a small repertoire of patterns used for their coupling effects, rather than for ornament: strategy captures model variability behind a single forecasting contract; a registry backed factory constructs concrete estimators by name; a process scoped singleton centralises configuration and artefact policy to avoid configuration drift; and service/gateway layers encapsulate I/O and transport so that higher layers remain ignorant of infrastructure concerns. This mix reflects widely accepted enterprise practice and keeps the axis of change local to the components that own it [94].

A second pillar is configuration as data and explicit artefacts. Hyper parameters and feature policies live in a simple JSON registry; trained estimators persisted as versioned artefacts; and merged datasets are snapshotted. Treating configuration and outputs as first-class deliverables ensures the workflow is runnable and auditable on a workstation and aligns with the product quality characteristics of ISO/IEC 25010 (reliability, maintainability, portability). The approach also mirrors common guidance on scientific computing practice (environment capture and immutable artefacts) [93][99][100].

Work proceeded in vertical slices that mirror the analyst's path – upload-train- predict – so that each increment yielded both a demonstrable capability behind the stable contract and a durable state (snapshot and artefact) proving that the capability exists independent of any single run.

Traceability from requirements to design decisions is maintained bidirectionally. Each requirement in Chapter 2 maps to a view and to a contract or policy in this chapter (e.g., reproducibility to snapshots/artefacts; maintainability to pattern guided module boundaries). This satisfies IEEE 29148 expectations for requirement quality and traceability [93]. The requirements traceability matrix is stored as appendix C.

Error handling is an architectural impediment. Service boundaries normalise faults into deterministic, typed results, preventing raw exceptions from crossing layers and keeping the UI from blocking while internals remain encapsulated [10][98]. This yields graceful degradation, diagnosability through structured telemetry, and user facing

clarity with technical detail confined to logs [15]. Input validation, idempotent side effects, and timeouts with back off prevent cascades. Dedicated mappers translate domain faults to stable, protocol appropriate responses, making failure modes explicit, testable, and traceable in line with established architectural guidance [95][96].

Finally, the methodology is intentionally conservative about what it can achieve: it fixes small interfaces, minimises runtime entanglement, and designs for extension rather than prediction. Additional endpoints (e.g., evaluation or explanation), alternative storage backends, or new forecasting strategies can be accommodated by adding to the seams rather than editing existing contracts – an explicit maintainability objective under ISO/IEC 25010 [93].

3.2 Data Science – CSV Cleaning and Concatenation

The ingestion path is specified by a strict data contract and a deterministic algebra of transformations. Each input file supplies a single temporal key Date in ISO-8601, one required real-valued target $y_t \in \mathbb{R}$, and zero or more exogenous variables $X_{t,j}$. A business day calendar is imposed via $\beta \subset \mathbb{Z}$ and requires a strictly increasing index $T = \{t_1 < \cdots < t_n\} \subset \beta$; nonconformant timestamps are rejected or mapped explicitly. Headers are canonicalised by a schema map $\varphi : headers \to \{Date, price, X_1, \dots\}$ (one-to-one on keys), with types coerced to datetime64[ns] for dates and float64 for numeric series; conversions and renames are logged for audit. These rules reflect the tidy data prescription – one variable per column, one observation per row [46] – and align with the previously stated policies on expected CSV schema, business day frequency, and type enforcement (date to datetime64, price to float64). The approach keeps the ingestion stage inference free: for each file F_k header canonicalisation is performed via φ, along with strict typing, stable sort by t, and a deterministic duplicate rule; let $C(F_k)$ denote this normalisation. Methodologically this conforms to established data engineering guidance in time-series workflows and the pandas idiom for columnar alignment [1][16][8]. Full diagram of this section's logic is stored in appendix K.

Combination proceeds by vertical concatenation then chronological canonicalisation. Updates of the same series are concatenated and reduced by:

$$D = canon(\oplus_{k=1}^{m} C(F_k),$$

where canon is a stable sort on t, dropping duplicates on t by a fixed rule with the latest-file-wins principle used to resolve collisions on identical t. For multi-source tables we align by a left/outer join on Date, letting the target's timeline define the modelling horizon while exogenous series contribute opportunistically; this shape minimises leakage and preserves the authority of the primary key (date) [1][8] and matches the CombineCSV outer-merge specification and column disambiguation (suffixing collisions, never renaming the date key) later shown in this chapter.

Missing data handling is causal by construction: the target y_t is never imputed and structural gaps bound the training window. Exogenous variables may be filled by a bounded forward-fill operator F_τ applied after joining,

$$\tilde{X}_{t,j} = F_\tau\big(X_{\le t,j}\big), \text{ run length} \le \tau,$$

or by short-horizon linear interpolation under the same bound; all fills are past-measurable and logged. Calendar conformance is enforced by the projection $\prod \beta$ so that

$T \subset \beta$ with strict monotonicity. The whole path – typed read, concatenate, sort, deduplicate – is $O(N)$ in rows and predominantly I/O bound via vectorised pandas routines; writes are atomic (temp → replace) to preclude torn outputs. These procedures accord with authoritative practice in forecasting pipelines and with Peixeiro's treatment of lagged/tabular workflows in Python [1][8][16].

Reproducibility and lineage are guaranteed by immutable snapshots and a stable alias: every successful combine emits a timestamped snapshot (app/data/…) and an updated canonical 'latest' file; downstream consumers address either a named snapshot or the alias, never an in-memory frame. This artefact discipline – configuration as data; immutable outputs – meets widely cited norms for computational research and facilitates workstation local audit [99][100]. The hands-off contract to modelling is intentionally minimal yet complete: a single index T, and required target $y \in \mathbb{R}$, and $0 - N$ exogenous $X_j \in \mathbb{R}$ with provenance of any bounded fills; feature construction (lags, rolling statistics, Fourier seasonality) consumes this tidy, typed table without recleaning, matching the expectations of mainstream Python tooling (pandas for wrangling, scikit-learn for downstream estimation) [16][64][101]. The acceptance criteria ('monotone timestamps, enforced dtypes, no duplicates'; 'write snapshot and latest on combine') are thereby satisfied by construction in the ingestion method.

3.3 Machine Learning – Nonlinear Forecasting

The system produces interval forecasts (a median path with lower/upper bands) by using two nonparametric, tree-based learners that natively handle nonlinearities and mixed features. The XGBoost pathway trains separate gradient-boosted models, one per requested quantile (e.g., 5%, 50%, 95%), each optimised for its percentile via the pinball (check) loss. The QRF pathway trains a single random forest and derives quantiles from the dispersion of its trees. Both consume the same engineered design matrix (calendar harmonics, target lags, optional rolling statistics, forward-filled exogenous series) and both generate future trajectories by a recursive day ahead loop that feeds the predicted median back into the lag buffer until the target date.

3.3.1 XGBoost with Pinball Loss

Let (x_t, y_t) be features and target on business-day index T. For a quantile level $\tau \in (0,1)$, the pinball loss for residual $e_t = y_t - \hat{y}_t$ is:

$$L_\tau(e_t) = \max(\tau e_t, (\tau - 1)e_\tau) = \begin{cases} \tau e_t, & e_t \geq 0, \\ (\tau - 1)e_t, & e_t < 0. \end{cases}$$

Minimising $\sum_t L_\tau(e_t)$ estimates the conditional τ −quantile $\mathbb{Q}_\tau(y \mid x)$ without assuming a parametric noise law [47][48]. In gradient boosting, the predictor is an additive expansion $\hat{f}(x) = \sum_{m=1}^{M} \upsilon h_m(x)$ of regression trees, whereby each tree h_m is fit to (sub)gradients of the objective; XGBoost supplies second order updates, shrinkage, column/row subsampling and regularisation that make this efficient and robust on tabular data [5][9][46]. In its implementation, one booster is trained per requested τ with a custom pinball objective (piecewise constant gradient – unit Hessian), and the model's base_score is initialised to the empirical τ − quantile to stabilise early rounds; trained boosters are kept in a map self.models[τ] and queried at serve time. Complexity is $O(N \cdot T \cdot d)$ per quantile (rows N, trees T, depth d); predition is $O(T \cdot d)$ per row per quantile. To produce the canonical (ℓ, m, u) = (5%, 50%, 95%) triplet, the service queries

the nearest trained quantiles and enforces monotonicity $\ell \leq m \leq u$ as a standard crossing repair step [47][48].

3.3.2 Quantile Random Forest

A random forest forms an ensemble $\{T_b\}_{b=1}^{B}$ of decorrelated trees; classical regression forests average leaf means to produce $\mathbb{E}[y \mid x]$ [7]. Quantile Regression Forests instead estimate the conditional distribution $F(y \mid x)$ by aggregating leaf sample distributions from all trees and then read off:

$$\mathbb{Q}_\tau(y \mid x) = \inf\{\, q : \hat{F}(q \mid x) \geq \tau \} \text{ [6].}$$

In its implementation, a single Random Forest Regressor is trained on the shared feature spine and then approximate QRF by taking empirical quantiles across per-tree predictions $\{\hat{y}_b(x)\}_{b=1}^{B}$:

$$\widehat{\mathbb{Q}}_\tau(x) \approx Quantile_\tau\left(\hat{y}_1(x), \ldots, \hat{y}_B(x)\right).$$

This is a pragmatic, dependency free variant that behaves well on structured data – although textbook QRF (leaf-sample aggregation) can yield wider and better calibrated intervals under strong heteroskedasticity [6][49][50]. Complexity is $O(B\,N\,d log N)$ to fit, where B represents trees, N represents rows, and d is approximated to all features in the dataset. On a per prediction basis, complexity is $O(BD + qB)$ where D is the depth of each tree with quantile partitioning.

3.3.3 Shared Feature Spine and Forecasting Loop

Both learners inherit an identical pipeline: strict schema and sort; calendrical features – ordinal day, weekday/year and cyclical harmonics (sine/cosine) to represent periodicity smoothly; lagged targets y_{t-k} and an optional rolling mean; forward-filled exogenous columns; and a frozen column order to prevent train serve skew. For historical dates, predictions are made on the reconstructed row; for future dates, the system advances one day at a time, recomputing time features and shifting a lag buffer whose next value is the predicted median $\hat{y}_{t+1\mid t}^{(0.5)}$. This yields a trajectory $\{(\ell_t, m_t, u_t)\}_{t=t_0+1}^{t^*}$ consistent with the same information set used in model training.

3.4 System Architecture

3.4.1 Backend

A single FastAPI application (served by Uvicorn) exposes a narrow REST surface (/upload, /train, /predict) with OpenAPI generated from route metadata. Requests are stateless and process local coordination (logger, model registry, paths) is module scoped. Interfaces are stabilised at two seams: the HTTP contract (typed JSON DTOs) and the model interface (shared fit/predict policy), aligning with REST's uniform interface and interface description practice [4][10][97]. Refer to appendices H and L for architecture diagram and backend UML.

Structure and Responsibilities

- Routes (HTTP adapters): app/routes/{upload,train,predict}.py – validation, error mapping, delegation.

- Services (domain logic): app/services/combine_csv.py (ingestion/canonical union), app/services/model_manager.py (config, registry, artefacts).
- Models: app/models/{base_forecast_model,xgb_model,qrf_model}.py – shared feature spine in the base; specialisations for quantile XGBoost and QRF.
- Config as data: app/config/models.json (lags, rolling window, quantiles, learner hyper-parameters).
- Data/artefacts: app/data/*.csv (immutable) and app/data/combined_data.csv (stable 'latest'); app/models/{xgb|qrf}.joblib.

This partitioning yields high cohesion by role and low coupling at seams, using strategy (model base), a registry backed factory (model selection), and a process local singleton by module for the manager [89].

Public API Contacts

- POST /upload – multipart CSV batch; persists files; outer merges on date, stable sort, deterministic duplicate policy; writes snapshot and latest; returns manifest and canonical paths.
- POST /train/{model} – trains {qrf|xgb} on the latest dataset (or an explicit path) using models.json; persists joblib artefact; returns summary.
- GET /predict/{model}/{date} – historical dates return observed; future dates return an interval forecast {lower, median, upper} at configured quantiles via a recursive day-ahead loop.

OpenAPI documentation is emitted automatically from route signatures [4][97].

Request Lifecycles

Upload: route streams files to CombineCSV.combine() and performs outer join on date, stable sort, deterministic dedupe; writes an immutable snapshot and overwrites latest by atomic replace (vectorised pandas; I/O bound) [16].

Train: route resolves dataset (prefer latest) to ModelManager and loads hyper parameters (models.json) before selecting the concrete learner via registry and calling fit(...) to persists artefacts.

Predict: route branches by date; for future dates the model's predict(...) advances one business day at a time, recomputing time features and updating a lag buffer with the predicted median to preserve causality.

Data Ingestion Service

CombineCSV implements the methodology from section 3.2 verbatim: outer merge on date, stable sort, deterministic duplicate rule ('latest file wins' principle), collision safe column disambiguation (suffixing non-date duplicates), and atomic write of a timestamped snapshot plus a stable latest file. This yields a reproducible hands-off file for modelling without recleaning. Complexity is linear in rows; operations are dominated by I/O and vectorised joins/sorts in pandas [16].

Model Layer – Quantile Learners on a Shared Spine

BaseForecastModel fixes the contract (fit/predict/save/load) and the feature spine: strict schema checks; ordinal time features; cyclical harmonics (DOW/DOY); target lags and

optional rolling mean; forward-filled exogenous; frozen column order to avoid train/serve skew.

- XGBoost (pinball loss): one booster per quantile τ, optimised with a custom pinball objective (piecewise linear check loss). The class initialises base_score to the empirical τ to stabilise early rounds, then stores boosters in a quantile indexed map. This follows gradient boosting theory and XGBoost's extensible objective interface [5][9][46][47][48].
- Quantile Random Forest: a single Random Forest Regressor is trained; quantiles are computed across per-tree predictions at inference (a pragmatic approximation). Intervals are returned at requested levels (e.g., 5%/50%/95%) [6][7][64].

Both models persist artefacts to app/models/*.joblib and are fully driven by models.json. (Appendix, Fig. A3.)

Data, Artefacts and Operations

Datasets under app/data/…; models under app/models/{qrf|xgb}.joblib; config in app/config/models.json; logs via a central logger. Health is inferred by fetching /openapi.json. Input validation (file type, date parse, numeric coercions), defensive exception mapping, and deterministic write semantics provide robustness on a desktop orchestrated deployment (HTTP/1.1 default, Uvicorn) [4][14][97][99][100].

3.4.2 Front End

The desktop client is a WPF application structured around MVVM: views are declared in XAML, view-models expose state and ICommand actions, and a thin services layer isolates all I/O. This yields high cohesion within layers and low coupling at seams (UI to view-model; view-model to backend gateway), in line with standard MVVM guidance and enterprise pattern practice [11][98]. Refer to appendices H and M for architecture diagram and frontend UML.

Structure and Responsibilities

- Views: Views/AnalysisView.xaml declares the analysis screen (file upload, model/date inputs, chart). The use of OxyPlot is explicit via the XAML namespace xmlns:oxy="http://oxyplot.org/wpf" and a PlotView bound to the chart model [12].
- View-models: ViewModels/AnalysisViewModel.cs coordinates the user flow (upload-train-predict). It exposes bindable properties (selected model, ISO date, readiness/busy flags, status) and three async commands: UploadCommand, TrainCommand, PredictCommand. Relay/INPC scaffolding lives in Framework/RelayCommand.cs and Framework/ViewModelBase.cs [11].
- Services (backend gateway and parsing):
    - Services/BackendClient.cs centralises HTTP calls to the Python service (/upload, /train/{model}, /predict/{model}/{date}); it uses a single static HttpClient with BaseAddress = new Uri('http://127.0.0.1:8000') and Timeout = Timeout.InfiniteTimeSpan. This keeps transport concerns out of the view-model and avoids socket churn.
    - Services/PredictionParser.cs converts backend JSON into lightweight DTOs for charting, deduplicating by date and merging partial bounds into consistent {lower, median, upper} triplets.

- Models/DTOs: Models/{PredictionDto,History,Forecast}.cs carry typed, chart ready series (history and forecast).
- Charting: Charting/{ChartFactory,ChartBuilder}.cs encapsulate OxyPlot setup and updates: a DateTime X-axis, a linear Y-axis, a median LineSeries, an observed LineSeries, and an AreaSeries for the interval band [12].
- Infrastructure (backend process supervision): Infrastructure/BackendHost.cs locates the backend, ensures the Conda env, launches Uvicorn, and polls /openapi.json until ready. Notable constants in code: EnvName = 'commodity-predictor' and HealthUrl = new($"http://{Host}:{Port}/openapi.json"). This lets the client host the API locally without user orchestration.

End-to-End Control Flow

Startup – BackendHost is constructed early; it checks Conda, (idempotently) ensures the 'commodity-predictor' environment, spawns uvicorn app.app:app --host 127.0.0.1 --port 8000, and only then signals readiness once /openapi.json responds.

User flow – The operator selects CSVs to UploadCommand and calls BackendClient.UploadAsync(...) (multipart), updates status, and refreshes UI readiness. Next, TrainCommand hits /train/{model} with the selected model key ('qrf'/'xgb'). Finally, PredictCommand validates the ISO date, calls /predict/{model}/{date}, and passes the JSON to PredictionParser. The resulting PredictionDto is applied to the chart.

Chart refresh – ChartBuilder clears/repairs series and repopulates: observed history to LineSeries; median forecast to LineSeries; interval to AreaSeries; axes are fitted to data and the PlotModel invalidated.

Errors and robustness – BackendClient funnels HTTP/network failures through a single SendAsync wrapper that maps transport exceptions to predictable messages; PredictionParser is defensive (invalid inputs produce an empty DTO). Commands are disabled while operations run to prevent re-entrancy [11][12].

Contracts at the UI Seam

The view-model is the only object seen by XAML. The bindings cover: inputs (model selector, ISO date), commands (Upload/Train/Predict), and outputs (status text, busy flags, and the PlotModel). This enforces separation of concerns: views are declarative; view-models are testable POCOs; services hide I/O details. The pattern choices – MVVM for presentation, Gateway for HTTP, and a small DTO model – minimise ripple effects and keep the UI ignorant of transport, schema, and chart library specifics [11][98].

Data and Process Lifetimes

No user data persisted by the client; artefacts live on the backend side (app/data/..., app/models/...). The frontend holds only transient in-memory state (view-model properties, OxyPlot series). BackendHost owns the backend process lifetime and disposes it on exit, closing pipes and cancelling any in-flight requests. The single shared HttpClient instance prevents socket exhaustion and concentrates defaults (BaseAddress, timeout) in one place.

Chapter 4 Testing and Evaluation

4.1 Scope and Methodology

The test programme is requirements driven and evidence centred. It adopts the structure recommended by ISO/IEC/IEEE 29119: a clear separation of unit, integration, acceptance, and user tests, each justified by the kind of assurance it provides and each traceable to the project's MoSCoW requirements [102]. Unit tests exist to localise defects and to stabilise internal contracts at module boundaries. They run hermetically (without external processes or networks) and with deterministic conditions (fixed seeds, bounded clocks, temporary workspaces). On the backend, unit tests therefore target module behaviour such as CSV ingestion, configuration and registry handling, feature engineering, and route adapters exercised through FastAPI's test client while I/O and model layers are stubbed; on the frontend, they target view-model commands, gateway behaviour with mocked HTTP, DTO parsing, and chart composition under MVVM, thereby verifying behaviour without a UI thread [4][11][12].

Integration tests serve a narrower but essential purpose: they confirm that interfaces remain compatible across seams. The backend integration test executes the full /upload to /train to /predict path with real files, model selection via the registry, and artefact persistence; the frontend integration test starts the local API through the host component, issues gateway calls, and verifies the resulting view-model state and chart model. Keeping integration tests few and deliberate concentrates their diagnostic value on interface compatibility rather than unit level details [4][97].

Acceptance tests verify that the Must/Should set is satisfied on a reference workstation with objective pass criteria (for example, snapshot plus latest written, quantile monotonicity, and offline operation). Their role is to demonstrate conformance to the initial requirements rather than to explore implementation alternatives [102]. User tests complement acceptance by examining usability and learnability in short sessions. They focus on task completion (upload-train-predict-interpret) and produce aggregated qualitative themes that directly prioritise the backlog [12].

Traceability is maintained end-to-end. Each test case maps to at least one requirement identifier; inputs, outputs, and logs are retained; any dataset or model produced during tests is written as an immutable snapshot with a stable alias for repeated execution. This ensures auditability and reproducibility in line with 29119's guidance on oracles, evidence, and coverage [102]. Detailed catalogues of unit tests, acceptance procedures, and user test protocols are provided in the Appendices.

4.2 Unit Tests

4.2.1 Backend Unit Tests

The backend suite (pytest) exercises the system at module seams to maximise fault localisation and to keep refactors inexpensive. The ingestion service (app/services/combine_csv.py) is checked for canonical outer merge on date, stable chronological ordering, the deterministic duplicate policy, and collision safe column disambiguation; every combine must emit a timestamped snapshot and atomically refresh the stable 'latest' file. The loader (app/utils/data_loader.py) is verified to prefer

combined_data.csv and to fall back deterministically to the newest raw CSV, while the model manager (app/services/model_manager.py) validates the registry/factory mapping (e.g., "qrf" to QRFModel, "xgb" to XGBoostForecastModel) and the integrity of models.json. The feature spine (app/models/base_forecast_model.py) enforces schema checks, ordinal and cyclical time encodings, lag and rolling features, bounded forward-fill for exogenous variables, and frozen column order to prevent train/serve skew; assertions use Pandas dtypes and shapes to guarantee structural invariants. Route adapters (app/routes/*.py) are exercised via FastAPI's TestClient to verify request/response contracts and error mappings, with downstream dependencies stubbed so the tests remain at unit level [4][97]. See appendix D.

4.2.2 Frontend Unit Tests

The frontend suite (MSTest/NUnit) leverages MVVM to test behaviour without rendering. AnalysisViewModel and its commands are verified for precondition enforcement, state transitions (IsBusy, status messages), and INotifyPropertyChanged notifications; the gateway (BackendClient) is exercised with mocked HTTP to confirm DTO serialisation, error/timeout handling, and resilience to non-200 responses; the parser (PredictionParser) produces chart ready data with deduplication by date and consistent {lower, median, upper} triplets. Chart composition is validated at the model level (ChartBuilder/ChartFactory), confirming a DateTime X-axis, a linear Y-axis, observed and median LineSeries, and an interval AreaSeries constructed correctly for OxyPlot without invoking the WPF thread. This design keeps tests fast, deterministic, and diagnostic while conforming to Microsoft's MVVM guidance and the OxyPlot composition model [11][12]. See appendix D.

4.3 Integration Tests

The integration layer verifies interface compatibility across components using one purposeful scenario per tier. On the backend, the end-to-end path exercises /upload to /train to /predict with real CSV files, model selection through the internal registry, artefact persistence, and JSON contract conformity; assertions include response codes, schema validity, monotone quantiles, and the presence of both a snapshot and a refreshed latest dataset. On the frontend, the test starts the local API via BackendHost, invokes BackendClient operations against http://127.0.0.1:8000, and routes results through the AnalysisViewModel to the chart model; assertions cover host readiness (OpenAPI available), error free command execution, DTO correctness, and a chart model containing the expected axes and series (observed, median, interval band). Both tests are deterministic and retain their artefacts to support replay and audit in line with ISO/IEC/IEEE practice [4][11][102]. See appendix E.

4.4 Acceptance Tests

Acceptance testing establishes that Must/Should requirements are satisfied on a reference workstation without external services. The executed set covers: upload and merge (outer join on date, stable ordering, deterministic deduplication, snapshot plus latest), train (named model; artefact persisted and reloadable), predict (interval – observed and future responses, monotone {lower, median, upper}), visualise (single chart with legible axes), and local/offline operation. Non-functional checks include reproducibility (artefacts and configuration recorded), reliability (400 series diagnostics, no unhandled exceptions), usability baseline (single screen workflow with responsive bindings), and portability (deterministic setup via Conda and .NET SDK). Detailed

scripts, evidence paths, and status fields are provided in the accompanying appendix – F. The design and documentation of these tests follow 29119 guidance on pass/fail criteria, test oracles, and evidence retention [102].

4.5 User Tests

User tests were conducted as short, device sessions using the desktop client against the local API. The objective was to assess usability and learnability along the main task (upload to train to predict to interpret), not to re-establish functional correctness. Aggregated themes informed the backlog: requests for export (PNG/PDF), stronger command gating (disable Predict until trained), brief model descriptions in the selector (QRF/XGB), progress indication, and clearer error messages (include filename and first offending row). These observations guide UI refinements and complement acceptance testing by addressing operator efficiency [11]. Table 2 below is a summary and the full list per participant is stored as appendix G.

Theme	Count (n = 10)	Comments
Export chart/data (PNG)	4	"Export chart as PNG."
Prevent premature predict/clearer affordance	3	"Disable Predict until trained."
Dark theme/visual options	3	"Dark theme would help."
Model dropdown help text	2	"Explain QRF vs XGB briefly."
Value tooltips	2	"Exact values on hover."
Remember last upload folder	1	"Remember last folder path."
Training progress indicator	1	"Show spinner/percentage."
Richer error details for bad CSV	1	"Include filename/row in message."
Legend toggles to show/hide series	1	"Toggle history/forecast via legend."
Date picker defaults to next business day	1	"Default date to next business day."

Table 2 - User Test Summary

Chapter 5 Conclusion

5.1 Achievements

This project has delivered a coherent forecasting application that moves from raw CSV imports to distributional predictions within a disciplined, reproducible workflow. The system integrates a WPF/.NET frontend, structured under MVVM, with a Python FastAPI service that executes data reconciliation, model training, and quantile prediction behind a narrow REST surface. The ingestion pipeline admits multiple heterogenous CSVs, canonicalises schema and types, enforces chronological monotonicity, and writes both an immutable snapshot and a stable latest artefact at each combine, so that downstream modelling proceeds from a single, typed table with an auditable lineage in the file system. This arrangement satisfies the core desiderata of traceability and reproducibility that were prioritised in the requirements and quality models adopted in Chapter 2.

On the modelling axis, two complementary non-parametric learners were implemented over a shared feature spine: XGBoost with the pinball objective, training one booster per requested quantile, and an efficient approximation of Quantile Regression Forests that aggregates per-tree predictions to form empirical quantiles. Both inherit identical calendar encodings, lag structures, and deterministic column ordering; both emit median paths with lower/upper bounds suitable for charting and further analysis. This results, in a single desktop product, the distributional orientation recommended in modern forecasting practice whilst remaining faithful to robust, tabular methods that are well supported in production libraries.

The user interface renders observed history, the forecast median, and the interval band in a single chart using OxyPlot, with a deliberately small set of interactions that reflect the operator's task sequence: upload-train-predict-interpret. Contracts at the seam (typed JSON DTOs) and in the code (fit/predict on a common base) keep coupling low and cohesion high, enabling extensive unit coverage without UI threads and a limited number of integration paths that exercise the full system across the process boundary. Acceptance criteria aligned to the MoSCoW "Must/Should" set were validated on a reference workstation, and short user sessions produced clear, actionable themes (e.g., gating of commands, concise model descriptions, export affordances) for the backlog. Collectively, these results show that Opus Equi constitutes a credible workstation tool for exploratory time-series forecasting with defensible uncertainty bands, built on open standards and testable seams.

5.2 Challenges and Limitations

Four constraints delimit the present work. First – project scope. The scope of the project had to be cut due to time constraints, therefore analytics such as MAE/RMSE, adverse weather patterns and derivatives pricing were not included. This is particularly noticeable when comparing the initial frontend wireframe in appendix I to the finished product in appendix J.

Second – probabilistic calibration. Training independent quantile boosters delivers sharp predictions but can produce quantile crossing; post-hoc monotonicity repairs are applied, yet they do not guarantee nominal coverage across regimes. The QRF

approximation used here behaves well on structured data but may yield wide intervals under strong heteroskedasticity; full leaf distribution aggregation or conformal calibration would strengthen coverage guarantees. These are recognised limitations of the chosen estimators and are acceptable in an MVP whose remit is a transparent, offline tool rather than a fully automated production forecaster.

Third – artefact lifecycle. The current service persists one artefact per model name and overwrites prior versions by design; while this simplifies usage, it constrains comparative evaluation and rollback. Timestamped artefact versioning and a minimal registry would make model lineage as explicit as dataset lineage, bringing symmetry to the evidence chain established elsewhere in the system.

Four – confidence intervals. A further practical observation concerns performance envelopes. Training and prediction targets were set for laptop hardware and achieved under default hyper parameters; nevertheless, hyper parameter optimisation, multi-core scheduling policies, and memory-aware batching were out of scope and would be prudent precursors to operating on very large histories or broad exogenous panels.

5.3 Future Development

Several concrete extensions follow directly from the foregoing analysis. For uncertainty, a calibration stage can be included over a rolling validation window, e.g., split conformal or quantile mapping, to align empirical coverage with nominal levels; evaluate with proper scoring rules (CRPS) alongside MAE/RMSE to balance sharpness and calibration. For modelling breadth, admit additional tree-based quantile learners or direct multi-horizon formulations and, where justified by data volume and governance, include probabilistic deep models as part of a secondary backend or API without disturbing the public contract. For data engineering, integrate optional exogenous series with explicit time of availability checks, extend the ingestion log to a lightweight provenance bundle, and expose bounded imputation policies to the UI. For operations, implement timestamped artefact versioning and a discoverable model registry, add export to PNG, and surface short help text for model selection; these are specifically demanded by the user test themes and can be implemented without architectural change. Finally, for governance and quality, tie acceptance scripts and performance runs to the snapshot and artefact IDs already emitted, thereby closing the loop from requirement to test evidence to published figure in a manner consistent with the standards cited in the requirements chapter.

To summarise, the project has established a principled baseline: a compact, inspectable system that enforces data hygiene, produces quantile forecasts with well understood learners, and preserves artefacts for audit. It is intentionally conservative in architecture and generous in testability. The next iteration should deepen calibration and artefact governance, widen the data aperture, and maintain the present discipline of typed seams and preserved evidence. In doing so, it will remain useful to the analyst and credible to the auditor.

References

[1] R. J. Hyndman and G. Athanasopoulos, Forecasting: Principles and Practice, 3rd ed. OTexts, 2021. [Online]. Available: https://otexts.com/fpp3/ (Accessed: 13 July. 2025).

[2] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, Time Series Analysis: Forecasting and Control, 5th ed. Hoboken, NJ, USA: Wiley, 2015.

[3] A. Troelsen and P. Japikse, Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. Berkeley, CA, USA: Apress, 2022.

[4] S. Ramírez, "FastAPI-High performance, easy to learn, fast to code, ready for production," FastAPI Documentation, 2025. [Online]. Available: https://fastapi.tiangolo.com/ (Accessed: 13 July. 2025).

[5] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785–794.

[6] N. Meinshausen, "Quantile Regression Forests," J. Mach. Learn. Res., vol. 7, pp. 983–999, 2006.

[7] L. Breiman, "Random Forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[8] M. Peixeiro, Time Series Forecasting in Python. 2022.

[9] The XGBoost Developers, "XGBoost Documentation," 2025. [Online]. Available: https://xgboost.readthedocs.io/ (Accessed: 14 July. 2025).

[10] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000.

[11] Microsoft, "Model-View-ViewModel (MVVM)," .NET Desktop Guide, 2025. [Online]. Available: https://learn.microsoft.com/dotnet/desktop/wpf (Accessed: 14 July. 2025).

[12] OxyPlot Project, "OxyPlot-PlotModel and Series," Documentation, 2025. [Online]. Available: https://oxyplot.github.io/ (Accessed: 16 July. 2025).

[13] ECMA International, "ECMA-404: The JSON Data Interchange Syntax," Oct. 2013. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ (Accessed: 17 July. 2025).

[14] R. Bray, P. Nottingham, and M. Belshe, "HTTP/2," IETF RFC 7540, May 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7540 (Accessed: 19 July. 2025).

[15] Python Software Foundation, "Logging—logging facility for Python," Python 3.x Docs, 2025. [Online]. Available: https://docs.python.org/3/library/logging.html (Accessed: 28 June. 2025).

[16] The pandas Dev Team, "pandas: powerful Python data analysis toolkit," Documentation, 2025. [Online]. Available: https://pandas.pydata.org/ (Accessed: 28 June. 2025).

[17] The scikit-learn Developers, “scikit-learn: Machine Learning in Python,” Documentation, 2025. [Online]. Available: https://scikit-learn.org/stable/ (Accessed: 28 June. 2025).

[18] Encode OSS, “Uvicorn-ASGI web server implementation for Python,” Documentation, 2025. [Online]. Available: https://www.uvicorn.org/ (Accessed: 28 June Sep. 2025).

[19] F. S. Mishkin and S. G. Eakins, Financial Markets and Institutions, 10th ed. Pearson, 2021. [Online]. (Accessed: 16 June. 2025).

[20] Z. Bodie, A. Kane, and A. J. Marcus, Investments, 13th ed. New York, NY, USA: McGraw-Hill, 2024. [Online] (Accessed: 16 June. 2025).

[21] J. C. Hull, Options, Futures, and Other Derivatives, 11th ed. Pearson, 2022. (Accessed: 16 June. 2025).

[22] Library of Congress, “A Resource Guide: Commodities Classification,” 2025. [Online]. (Accessed: 16 June. 2025).

[23] Food and Agriculture Organization (FAO), “Definition and Classification of Commodities,” 2000. [Online]. (Accessed: 14 June. 2025).

[24] CME Group, “Learn to Trade Futures and Options,” 2025. [Online]. Available: https://www.cmegroup.com/trading/why-futures/learn-to-trade-futures.html (Accessed: 14 June. 2025).

[25] ICE Futures U.S., “Products – Futures & Options,” 2025. [Online]. Available: https://www.ice.com/products/Futures-Options (Accessed: 17 June. 2025).

[26] U.S. Department of Agriculture, Foreign Agricultural Service, “Oilseeds: World Markets and Trade”. [Online]. Available: https://apps.fas.usda.gov/psdonline/circulars/oilseeds.pdf Accessed: 14 June. 2025).

[27] W. Anderson, “ENSO as a climate conductor for global crop yields,” NOAA Climate.gov (blog), 2023. [Online]. Available: https://www.climate.gov/news-features/blogs/enso/enso-climate-conductor-global-crop-yields (Accessed: 14 June. 2025).

[28] P. Cashin, L. Cespedes, and R. Sahay, “Commodity Currencies and the Real Exchange Rate,” IMF (book chapter), 2004. [Online]. Available: https://www.elibrary.imf.org/downloadpdf/display/book/9781589067288/ch001.pdf (Accessed: 14 June. 2025).

[29] U.S. Department of Agriculture, Economic Research Service, “Soybeans and Oil Crops - Market Outlook,” Aug. 15, 2025. [Online]. Available: https://www.ers.usda.gov/topics/crops/soybeans-and-oil-crops/market-outlook (Accessed: 12 June. 2025).

[30] The Baltic Exchange, “Dry (Bulk) Market Information & Assessments,” 2025. [Online]. Available: https://www.balticexchange.com/en/data-services/market-information0/dry-services.html (Accessed: 12 June. 2025).

[31] U.S. Department of Agriculture, Economic Research Service, "Demand for Alternative Feed Grains for Broiler Production in an Era of High Price Volatility (ERR-350)," Apr. 2025. [Online]. (Accessed: 12 June. 2025).

[32] U.S. Securities and Exchange Commission, Investor.gov Glossary, "Bid-Ask Spread," 2025. [Online]. Available: https://www.investor.gov/introduction-investing/investing-basics/glossary (Accessed: 17 June. 2025).

[33] CME Group, "A Trader's Guide to Futures," 2013. [Online]. Available: https://www.cmegroup.com/education/files/a-traders-guide-to-futures.pdf (Accessed: 17 June. 2025).

[34] Bank for International Settlements, "About derivatives statistics," 2025. [Online]. Available: https://www.bis.org/statistics/dataportal/derivatives.htm (Accessed: 17 June. 2025).

[35] CME Group, "CME Group Sets New ADV Record of 35.9 Million Contracts in April," Press Release, May 2, 2025. [Online]. Available: https://www.cmegroup.com/media-room/press-releases/2025/5/02/cme_group_sets_newadvrecordof359millioncontractsinaprilup36yearo.html (Accessed: 17 June. 2025).

[36] CME Group, "Self-Study Guide to Hedging with Grain and Oilseed Futures and Options," 2015 (updated). [Online]. Available: https://www.cmegroup.com/trading/agricultural/files/grain-oilseed-hedgers-guide.pdf (Accessed: 17 June. 2025).

[37] U.S. Department of Agriculture, Risk Management Agency, "Building a Risk Management Plan," 1998 (reprinted). [Online]. Available: https://www.rma.usda.gov/sites/default/files/topics/risk_management_plan1998.pdf (Accessed: 18 June. 2025).

[38] G. Gorton and K. G. Rouwenhorst, "Facts and Fantasies about Commodity Futures," Financial Analysts Journal, vol. 62, no. 2, pp. 47–68, 2006. [Online]. (Accessed: 17 Sep. 2025).

[39] Anaconda, Inc., "conda-Package, dependency and environment management," 2025. [Online]. Available: https://docs.conda.io/ (Accessed: 28 June. 2025).

[40] J. D. Hamilton, Time Series Analysis. Princeton, NJ, USA: Princeton Univ. Press, 1994.

[41] J. Durbin and S. J. Koopman, Time Series Analysis by State Space Methods, 2nd ed. Oxford, U.K.: Oxford Univ. Press, 2012.

[42] H. Geman, Commodities and Commodity Derivatives. Chichester, U.K.: Wiley, 2005.

[43] J. C. Hull, Options, Futures, and Other Derivatives, 10th ed. Harlow, U.K.: Pearson, 2018.

[44] H. Working, "The theory of the price of storage," Amer. Econ. Rev., vol. 39, no. 6, pp. 1254-1262, 1949.

[45] G. B. Gorton and K. G. Rouwenhorst, “Facts and fantasies about commodity futures,” J. Finance, vol. 62, no. 3, pp. 1235-1268, 2006.

[46] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” Ann. Statist., vol. 29, no. 5, pp. 1189–1232, 2001.

[47] R. Koenker and G. Bassett Jr., “Regression quantiles,” Econometrica, vol. 46, no. 1, pp. 33-50, 1978.

[48] R. Koenker, Quantile Regression. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[49] T. Gneiting and A. E. Raftery, “Strictly proper scoring rules, prediction, and estimation,” J. Amer. Stat. Assoc., vol. 102, no. 477, pp. 359-378, 2007.

[50] T. Gneiting and M. Katzfuss, “Probabilistic forecasting,” Annu. Rev. Stat. Appl., vol. 1, pp. 125-151, 2014.

[51] H. Hersbach, “Decomposition of the continuous ranked probability score for ensemble prediction systems,” Wea. Forecasting, vol. 15, no. 5, pp. 559-570, 2000.

[52] R. L. Winkler, “A decision-theoretic approach to interval estimation,” J. Amer. Stat. Assoc., vol. 67, no. 337, pp. 187-191, 1972.

[53] R. J. Hyndman and A. B. Koehler, “Another look at measures of forecast accuracy,” Int. J. Forecast., vol. 22, no. 4, pp. 679-688, 2006.

[54] L. J. Tashman, “Out-of-sample tests of forecasting accuracy: An analysis and review,” Int. J. Forecast., vol. 16, no. 4, pp. 437-450, 2000.

[55] C. Bergmeir and J. M. Benítez, “On the use of cross-validation for time series predictor evaluation,” Inf. Sci., vol. 191, pp. 192-213, 2012.

[56] F. X. Diebold and R. S. Mariano, “Comparing predictive accuracy,” J. Bus. Econ. Stat., vol. 13, no. 3, pp. 253-263, 1995.

[57] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, “The M4 competition: Results, findings, conclusion and way forward,” Int. J. Forecast., vol. 34, no. 4, pp. 802-808, 2018.

[58] S. Makridakis et al., “The M5 accuracy competition: Results, findings and conclusions,” Int. J. Forecast., vol. 38, no. 4, pp. 1346-1364, 2022.

[59] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, “DeepAR: Probabilistic forecasting with autoregressive recurrent networks,” Int. J. Forecast., vol. 36, no. 3, pp. 1181–1191, 2020.

[60] H. Zhou, S. Zhang, J. Peng, et al., “Informer: Beyond efficient transformer for long sequence time-series forecasting,” in Proc. AAAI Conf. Artif. Intell., 2021, pp. 11106-11115.

[61] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, “A state space framework for automatic forecasting using exponential smoothing methods,” Int. J. Forecast., vol. 18, no. 3, pp. 439–454, 2002.

[62] P. J. Brockwell and R. A. Davis, Introduction to Time Series and Forecasting, 3rd ed. New York, NY, USA: Springer, 2016.

[63] H. Wickham, “Tidy data,” J. Stat. Softw., vol. 59, no. 10, pp. 1-23, 2014.

[64] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[65] LSEG (Refinitiv), “Datastream Web Service (DSWS) – API Overview,” Product documentation, 2025.

[66] LSEG (Refinitiv), “Datastream Web Service – Getting Started with Python,” Technical guide, 2020.

[67] LSEG (Refinitiv), “Datastream Chart Studio / Datastream Charting – Getting Started,” User guide, various editions, 2020-2024.

[68] Bloomberg, “BQuant Desktop – Product Overview,” Product documentation, 2025.

[69] Bloomberg, “Introduction to BQuant Desktop and App Publishing,” Webinar materials, 2025.

[70] FactSet, “API Catalog – Overview of Data and Analytics APIs,” Developer documentation, 2025.

[71] FactSet, “Real-Time Time Series API – Reference,” Developer documentation, 2025.

[72] FactSet, “Global Prices API – Reference,” Developer documentation, 2025.

[73] Amazon Web Services, “Amazon Forecast – Developer Guide,” Service documentation, 2025.

[74] Amazon Web Services, “Amazon Forecast – Code Examples,” Service documentation, 2025.

[75] Microsoft, “Automated ML for Time-Series Forecasting (Azure ML v2),” Product documentation, 2024.

[76] Google Cloud, “Vertex AI – Forecasting with AutoML,” Product documentation, 2025.

[77] Microsoft, “FORECAST.ETS Function – Excel,” Product documentation, 2025.

[78] Microsoft, “Automated ML Forecasting – YAML/CLI (Azure ML v2),” Product documentation, 2024.

[79] Google Cloud, “Vertex AI – Platform Documentation,” Product documentation, 2025.

[80] Microsoft, “Forecasting Functions – Reference (Excel),” Product documentation, 2025.

[81] MathWorks, “Econometrics Toolbox – Documentation,” Product documentation, 2025.

[82] IHS Markit, EViews 12 User's Guide I & II, Manuals, 2020.

[83] EViews, "Forecasting Tutorial – Learning Series," Product documentation, 2025.

[84] MathWorks, "Econometric Modeler App – Overview," Product documentation, 2025.

[85] statsmodels Developers, "Time Series Analysis (tsa) – User Guide," Project documentation (v0.14.x), 2025.

[86] Amazon Web Services, "Amazon Forecast – Resources and Best Practices," Service documentation, 2025.

[87] Microsoft, "AutoML Training for Time-Series (Azure ML SDK v1)," Product documentation, 2024.

[88] GluonTS Team, "GluonTS: Probabilistic Time-Series Modeling in Python," Project documentation, 2025.

[89] A. Alexandrov et al., "GluonTS: Probabilistic and Neural Time Series Modeling in Python," J. Mach. Learn. Res., vol. 21, pp. 1-6, 2020.

[90] Uber Technologies, "Orbit: A Package for Bayesian Time Series Forecasting – Documentation," Project documentation, 2025.

[91] Uber Technologies, "Introducing Orbit – An Open-Source Package for Time Series Forecasting," Engineering blog, 2021.

[92] Unit8, "Darts: Time Series Made Easy in Python – Documentation," Project documentation, 2025.

[93] ISO/IEC/IEEE 29148:2018, Systems and Software Engineering – Life Cycle Processes – Requirements Engineering. Geneva, Switzerland: ISO, 2018.

[94] ISO/IEC 25010:2011, Systems and Software Engineering—System and Software Quality Models. Geneva, Switzerland: ISO, 2011.

[95] P. B. Kruchten, "The 4+1 view model of architecture," IEEE Software, vol. 12, no. 6, pp. 42–50, Nov. 1995.

[96] ISO/IEC/IEEE 42010:2011, Systems and Software Engineering—Architecture Description, 2011.

[97] OpenAPI Initiative, "OpenAPI Specification 3.1.0," 2021. [Online]. Available: https://spec.openapis.org/oas/latest.html

[98] M. Fowler, Patterns of Enterprise Application Architecture. Boston, MA, USA: Addison-Wesley, 2002.

[99] ACM, Artifact Review and Badging - Version 1.1, 2020. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-badging

[100] G. Wilson et al., "Good enough practices in scientific computing," PLOS Computational Biology, vol. 13, no. 6, e1005510, 2017.

[101] The statsmodels Developers, “Time Series Analysis (tsa)—User Guide,” v0.14.x, 2025. [Online]. Available: https://www.statsmodels.org/stable/tsa.html

[102] ISO/IEC/IEEE 29119, Software and Systems Engineering – Software Testing. Geneva, Switzerland: ISO, 2013–2022.

Appendix A MoSCoW Prioritisation

Type	MoSCoW	ID	Requirement	Achieved (Y/N)
Functional	Must	F-M1	Upload and merge. The system shall accept ≥ 1 CSV per session and produce a merged, type coherent dataset with snapshotting.	Y
Functional	Must	F-M2	Train. The system shall train a named model on the latest merged dataset and persist the artefact.	Y
Functional	Must	F-M3	Predict. The system shall return median and upper/lower prediction bounds for a requested date, together with historical series.	Y
Functional	Must	F-M4	Visualise. The client shall render history, forecast, and interval band in a single chart with legible axes.	Y
Functional	Must	F-M5	Local, offline operation. All functions shall execute on the user's workstation without external services.	Y
Functional	Should	F-S1	Model selection from registry. Available models shall be discoverable via models.json and selectable in the UI.	Y
Functional	Should	F-S2	Artefact/version management. The service should support timestamped artefacts and deterministic retrieval.	Y
Functional	Should	F-S3	Futures pricing. Provide a placeholder route and client affordance for cost-of-carry pricing to be completed later.	N

Functional	Should	F-S4	Authentication. Provide a placeholder login view and token plumbing consistent with later RBAC.	N
Functional	Could	F-C1	Export. Export chart (PNG) and data as a PDF.	N
Functional	Could	F-C2	Dataset preview. Grid preview of merged dataset in the client.	Y
Functional	Will Not	F-W1	Cloud deployment. Docker/Kubernetes packaging.	N
Functional	Will Not	F-W2	Live data feeds. External API integrations.	N
Non-functional	Must	N-M1	Reproducibility. Every merged dataset is snapshotted; every trained model persisted; configurations are explicit.	Y
Non-functional	Must	N-M3	Reliability. Fail fast with actionable diagnostics on malformed inputs; no unhandled exceptions leak to the UI.	Y
Non-functional	Must	N-M4	Usability. Single screen workflow (upload to train to predict to plot) with progress feedback; MVVM bindings remain responsive.	Y
Non-functional	Must	N-S1	Portability. Deterministic environment setup via Conda + .NET SDK; minimal external dependencies.	Y
Non-functional	Should	N-S2	Maintainability. Clear separation of concerns (Views, ViewModels, Services; routes, services, utils, models); typed DTOs; consistent logging.	Y
Non-functional	Could	N-C1	Security posture. Local only operation; no network exposure beyond localhost.	Y

Appendix B Risk Register

ID	Risk	Probability	Impact	Mitigation	Residual
R1	Data heterogeneity (schema drift, malformed timestamps)	Medium	High	Canonical schema map; strict dtype coercion; fail-fast with diagnostics; snapshot raw and merged datasets.	Medium
R2	Schedule overrun	Medium	Medium	MoSCoW scope; vertical slices; defer futures/auth; time-boxing.	Medium
R3	Interval miscalibration	Medium	Medium	Quantile models (QRF/XGB); report empirical coverage and Winkler/CRPS; consider conformal calibration later.	Medium
R4	Overfitting/ leakage	Low	High	Rolling origin evaluation; blocked CV; separate fit/forecast; DM test.	Low
R5	Dependency breakage	Medium	Medium	Pin versions; lock Conda; record artefact framework versions; smoke tests.	Low
R6	UI unresponsiveness	Medium	Medium	MVVM async commands; cancellation; progress indicators.	Low
R7	Performance regressions	Medium	Medium	Latency budgets; simple benchmarks; log timings at INFO.	Low
R8	Reproducibility gaps	Low	High	Snapshots; artefact persistence; config immutability; RTM.	Low
R9	Security/privacy issues	Low	Medium	Local only binding; no PII; minimise logs; path sanitisation; auth deferred.	Low

R10	Evaluation bias (invalid metrics)	Low	Medium	Prefer MAE/RMSE/MASE; avoid raw MAPE on zeros; report uncertainty scores.	Low

Appendix C Requirements Traceability Matrix

ID	Verification Target	Design and Implementation (Ch. 3)	Test (Ch. 4)
F-M1	CSV ingestion to merged snapshot	CombineCsv.save_uploads; CombineCsv.combine_latest; DataLoader.latest_dataset	IntegrationTests (upload to merge); DataLoader tests (row counts, monotone dates)
F-M2	Train persists artefact	ModelManager.train; artefact path policy; model registry (models.json)	Backend tests (artefact exists, loadable); IntegrationTests (train to predict without retrain)
F-M3	Predict returns hist plus median plus bounds	ModelManager.predict (QRF/XGB quantiles); /predict/{model}/{date} schema	ParserTests (DTO shape); IntegrationTests (alignment, keys)
F-M4	Visualise series plus band	ChartFactory; ChartBuilder; AnalysisViewModel bindings	ChartingTests (series counts, band presence); UI smoke
F-M5	Local/offline operation	BackendHost (spawn Uvicorn); Conda plus .NET scripts	Ops check (no external calls); README runbook
F-S1	Models selectable from registry	models.json contract; VM binding	VM tests (selection flows); manual acceptance
F-S2	Versioned artefacts	Artefact naming plus registry metadata	Backend tests (old/new resolution)
F-S3	Futures pricing stub	Placeholder route plus UI affordance	Contract test (501/Not Implemented)
F-S4	Auth stub	Login view shell; client token pipe	UI smoke (view loads); contract test
N-M1	Reproducibility	Snapshot directory; artefact policy; logging	IntegrationTests (snapshot written); log inspection
N-M2	Performance targets	Predict/train timing budgets	Benchmark harness (wall-clock); report table

N-M3	Reliability	Exception handling; error DTOs	Negative tests (malformed CSV, missing column)
N-M4	Usability	MVVM async commands; progress feedback	VM tests (can-execute toggling); UX notes
N-S1	Maintainability	Package boundaries; typed DTOs	Static review (structure); unit tests coverage
N-S2	Portability	Conda env; .NET SDK versions	Cold start run on clean machine

Appendix D Unit Tests

Component	File	Test	Purpose
Backend	test_api_errors.py	test_train_unknown_model	train unknown model
Backend	test_api_errors.py	test_predict_invalid_date_returns	predict invalid date returns
Backend	test_api_errors.py	test_train_explicit_missing_csv	train explicit missing csv
Backend	test_api_errors.py	test_predict_without_trained_model	Ensure no trained artifact is discoverable, then call predict. Redirect the manager's models_dir to an empty temp dir so the API must respond
Backend	test_api_forecast.py	test_train_and_predict_future_is_monotone	Happy path: train then predict a date approximately 1 week into the future.
Backend	test_api_forecast.py	test_predictions_are_deterministic	With TEST_MODE and fixed seeds, identical inputs should yield identical outputs.
Backend	test_manager_config.py	test_config_missing_key_raises	config missing key raises
Backend	test_models_prepare.py	test_prepare_training_data_shapes_and_columns	Build a tiny frame with gaps in exogenous features to exercise forward fill. Assert shapes and required feature columns after lag/rolling tr

Backend	test_services_combine.py	test_disambiguates_duplicate_columns	Two inputs share column 'x' one should be suffixed (e.g., 'x__2').
Backend	test_services_combine.py	test_missing_date_column_raises	A CSV without DATE_COLUMN should raise ValueError during combine().
Backend	test_services_combine.py	test_outer_join_preserves_all_dates	Outer join behavior preserves union of dates in ascending order.
Backend	test_upload.py	test_upload_single_csv	"Happy path: upload a tiny CSV and assert the combined file exists locally.
Backend	test_upload.py	test_upload_invalid_missing_date	Error path: uploading a CSV without a date column should return 422 (validation error).
Backend	test_utils_data_loader.py	test_prefers_combined	If a combined file exists it should be selected even if a single CSV is newer.
Backend	test_utils_data_loader.py	test_falls_back_to_latest	Without a combined file, choose the newest *.csv by modification time.
Frontend	ChartingTests.cs	ChartFactory_BuildsExpectedAxesLegendAndSeriesOrder	Chart Factory Builds Expected Axes Legend And Series Order
Frontend	ChartingTests.cs	ChartBuilder_PopulatesSeries_FitsXAxis_AndSkipsHalfIntervals	Chart Builder Populates Series Fits XAxis And Skips Half Intervals
Frontend	CommandAndViewModelTests.cs	RelayCommand_DisablesWhileExecuting_ThenReenables	Relay Command Disables While Executing Then Reenables

Frontend	CommandAndViewModelTests.cs	AnalysisVM_Predict_Enablement_FollowsInputs_AndBackendReady	Analysis VM Predict Enablement Follows Inputs And Backend Ready
Frontend	CommandAndViewModelTests.cs	AnalysisVM_Predict_UpdatesStatusAndBusy_OnSuccessAndFailure	Analysis VM Predict Updates Status And Busy On Success And Failure
Frontend	ParserTests.cs	Parse_BlankOrInvalid_ReturnsEmptyDto	Parse Blank Or Invalid Returns Empty Dto
Frontend	ParserTests.cs	Parse_DedupsHistory_FiltersForecastAfterLastHistory_AndMergesBounds	Parse Dedups History Filters Forecast After Last History And Merges Bounds

Appendix E Integration Tests

Component	File	Tests included	Preconditions	Steps (key flow)	Expected outcome
Backend	integration_test.py	test_integration_full	Local backend process available (Uvicorn) or started by test; tests/testdata present	Copy test only models.json into a temp app dir; patch ModelManager to use it to POST /upload with CSVs from tests/testdata to POST /train/qrf with explicit data_path to GET /predict for (i) in-sample date to returns observed, (ii) future date to returns lower/median/upper with forward trajectory	HTTP 200s throughout; snapshot + latest written; artefact persisted; in-sample prediction returns actual; future prediction returns ordered quantiles (lower ≤ median ≤ upper) and valid history/forecast payload
Frontend	Integration Tests.cs	Integration test methods	Conda available; backend folder accessible; network localhost reachable; tests can mark Inconclusive if environment missing	Start backend via BackendHost (Conda env); poll /openapi.json until healthy to Upload sample CSVs using BackendClient to Train a model via BackendClient to Predict for a valid date; parse JSON into DTO; update view-model/chart (indirectly)	Integration methods (e.g., Backend_Healthcheck_Responds, FullFlow_Train_Then_Predict) succeed: healthcheck true; upload/train/predict requests complete; DTO parse without exceptions

Appendix F Acceptance Test Matrix

ID	Requirement (MoSCoW)	Objective	Preconditions	Procedure	Expected result / Acceptance criteria	Evidence artefacts
AT-F-M1	F-M1 Upload & merge	Validate ingestion with multi-CSV upload, canonical outer-merge on `date`, stable sort, deterministic deduplication, and snapshot plus latest writing.	Backend running on localhost; sample CSVs available.	POST /upload with ≥2 CSVs; inspect combined outputs.	HTTP 200; union of dates preserved in ascending order; non-date duplicate columns suffix-disambiguated; timestamped snapshot written; 'app/data/combined_data.csv' replaced atomically.	app/data/snapshots/combined_*.csv; app/data/combined_data.csv; server logs.
AT-F-M2	F-M2 Train	Train a named model on the latest merged dataset and persist the artefact.	Latest dataset present; models.json valid.	POST /train/qrf and /train/xgb.	HTTP 200; artefacts written under app/models/*.joblib; reload succeeds without error.	app/models/qrf.joblib; app/models/xgb.joblib; server logs.
AT-F-M3	F-M3 Predict (interval)	Return median and lower/upper bounds for requested date with history.	Trained artefact available; backend running.	GET /predict/{model}/{date} for observed and future dates.	Observed date returns actual; future date returns {lower, median, upper} with lower ≤ median ≤ upper; payload includes history and only future horizon dates.	Saved JSON responses; server logs.

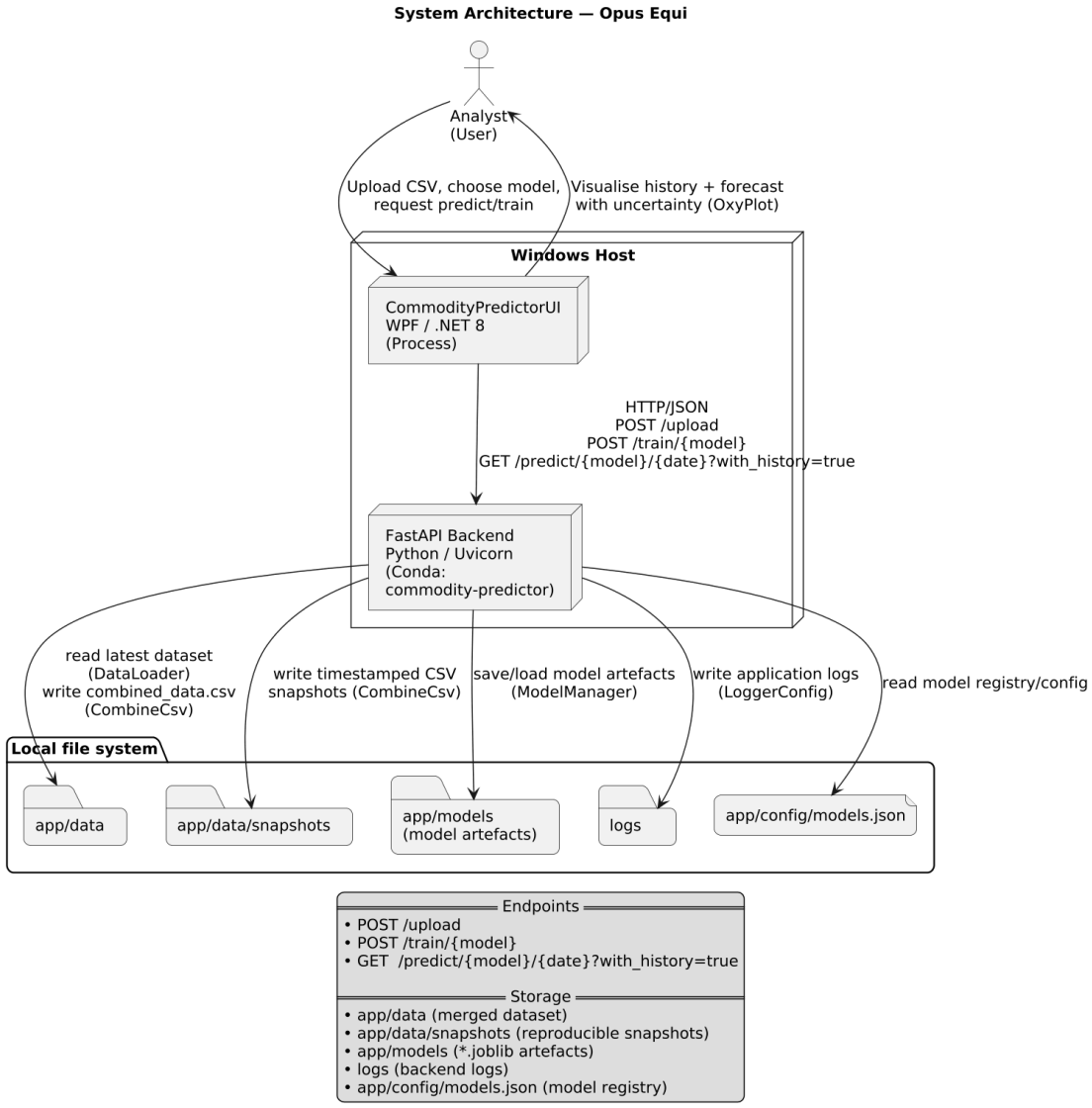
AT-F-M4	F-M4 Visualise	Render observed, median, and interval band in a single chart with legible axes.	Frontend launched; backend available; prediction payload available.	Load DTO into chart; inspect PlotModel.	DateTime X-axis and linear Y-axis present; LineSeries for observed and median; AreaSeries for interval band; series counts match payload.	Screenshot; exported PNG (if enabled).
AT-F-M5	F-M5 Local/offline	Demonstrate end-to-end operation on localhost without external services.	Air-gapped or network disabled environment.	Run upload to train to predict using local files only.	All steps succeed; no network requests beyond 127.0.0.1.	Network capture (optional); logs showing localhost only.
AT-F-S1	F-S1 Model selection from registry	Selectable models are discoverable via models.json and exposed in UI.	Frontend running; models.json contains qrf/xgb.	Open model selector; choose qrf then xgb; train both.	UI lists available models; training dispatches accordingly; artefacts named per model.	UI screenshot; artefact paths; models.json.
AT-F-S2	F-S2 Artefact/version management	Support timestamped artefacts and deterministic retrieval.	Two training runs executed sequentially.	Train model twice; list app/models; load latest.	Two artefacts exist with distinct timestamps; latest resolved deterministically; both load successfully.	Directory listing; logs of load path.
AT-N-M1	N-M1 Reproducibility	Every merge is snapshotted; every model persisted; configurations explicit.	As per F-M1/F-M2.	Execute F-M1 and F-M2 once; capture artefacts.	Snapshot present after upload; artefact present after train; models.json recorded in logs/config bundle.	Snapshot path; artefact path; config capture.

AT-N-M3	N-M3 Reliability	Fail fast with diagnostics; no unhandled exceptions leak to UI.	Backend/Frontend running.	POST /upload with malformed CSV; call /predict with invalid date.	HTTP 4xx with informative error JSON; frontend shows user-friendly message; no stack traces visible.	HTTP payloads; UI screenshot; logs without tracebacks.
AT-N-M4	N-M4 Usability (baseline)	Single-screen workflow remains responsive and discoverable.	Frontend running; backend started by host.	Upload to train to predict in one screen; observe busy/status updates.	Commands disabled while running; status/progress visible; no UI freezes.	Screen recording or screenshots.
AT-N-S2	N-S2 Portability	Deterministic setup via Conda + .NET SDK; minimal external dependencies.	Clean workstation; internet optional.	Create/activate Conda env; build/run frontend; start backend.	Environment resolves without manual patching; both tiers start successfully.	Setup transcript; versions output.

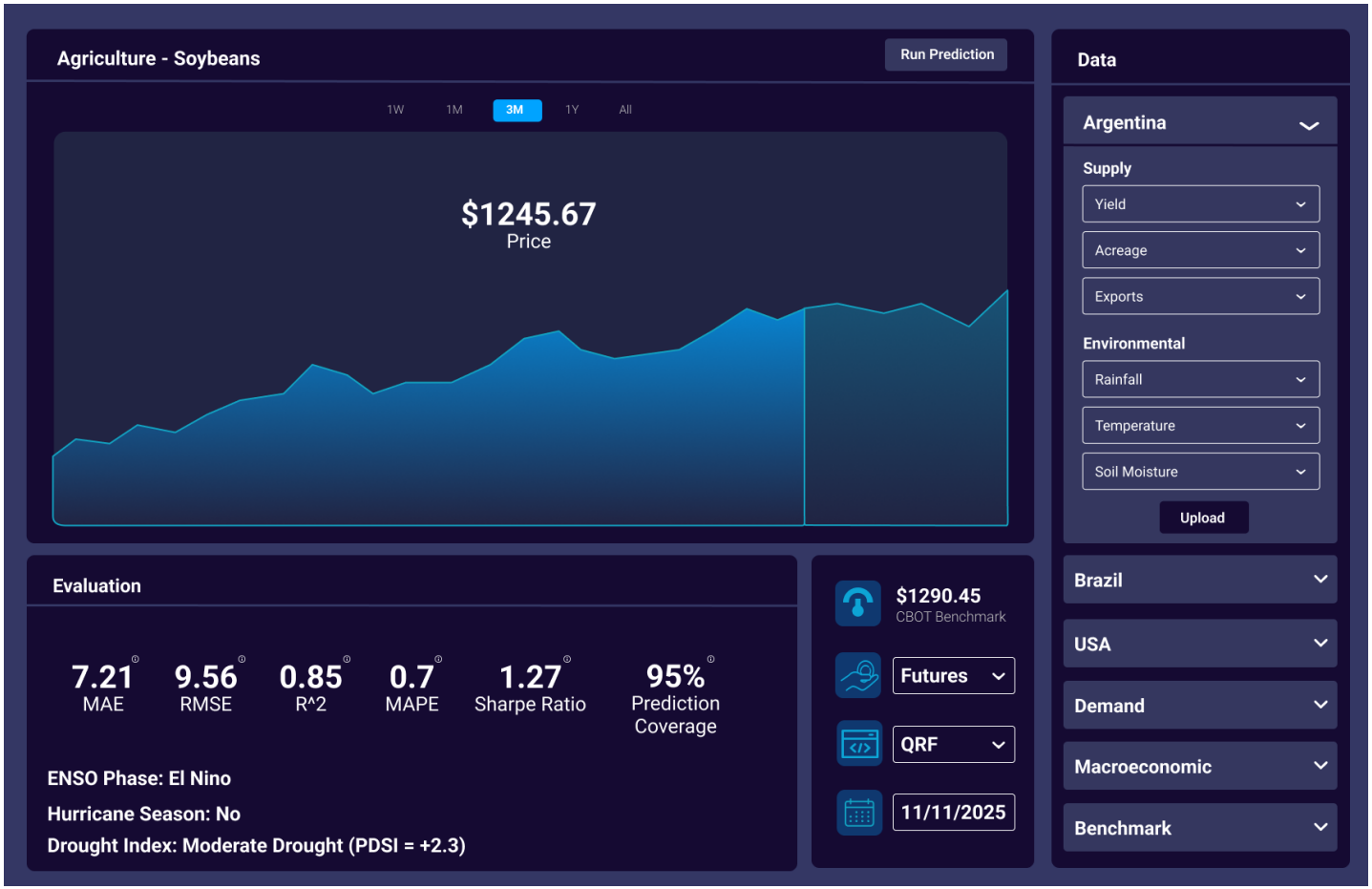
Appendix G User Test Matrix

Participant	Session Length (min)	Issues Observed	Notable Comments / Observations
P01	6	0	“It would be nice to export the chart as an image.”
P02	7	1	Tried Predict before Train; message helped.
P03	5	0	“Dark theme option would help; band is easy to read.”
P04	9	1	Confused by model names. “Add short descriptions for QRF/XGB in the dropdown.”
P05	8	0	“Tooltip with exact values on hover would be useful.”
P06	6	0	“Remember last folder path when uploading multiple times.”
P07	7	0	“Show training progress (percentage).”
P08	10	2	Hit malformed CSV once; error was readable. “Include filename and first bad row in the message.”
P09	6	0	“Let me toggle history/forecast series via the legend.”
P10	7	0	“Date picker could default to next business day.”

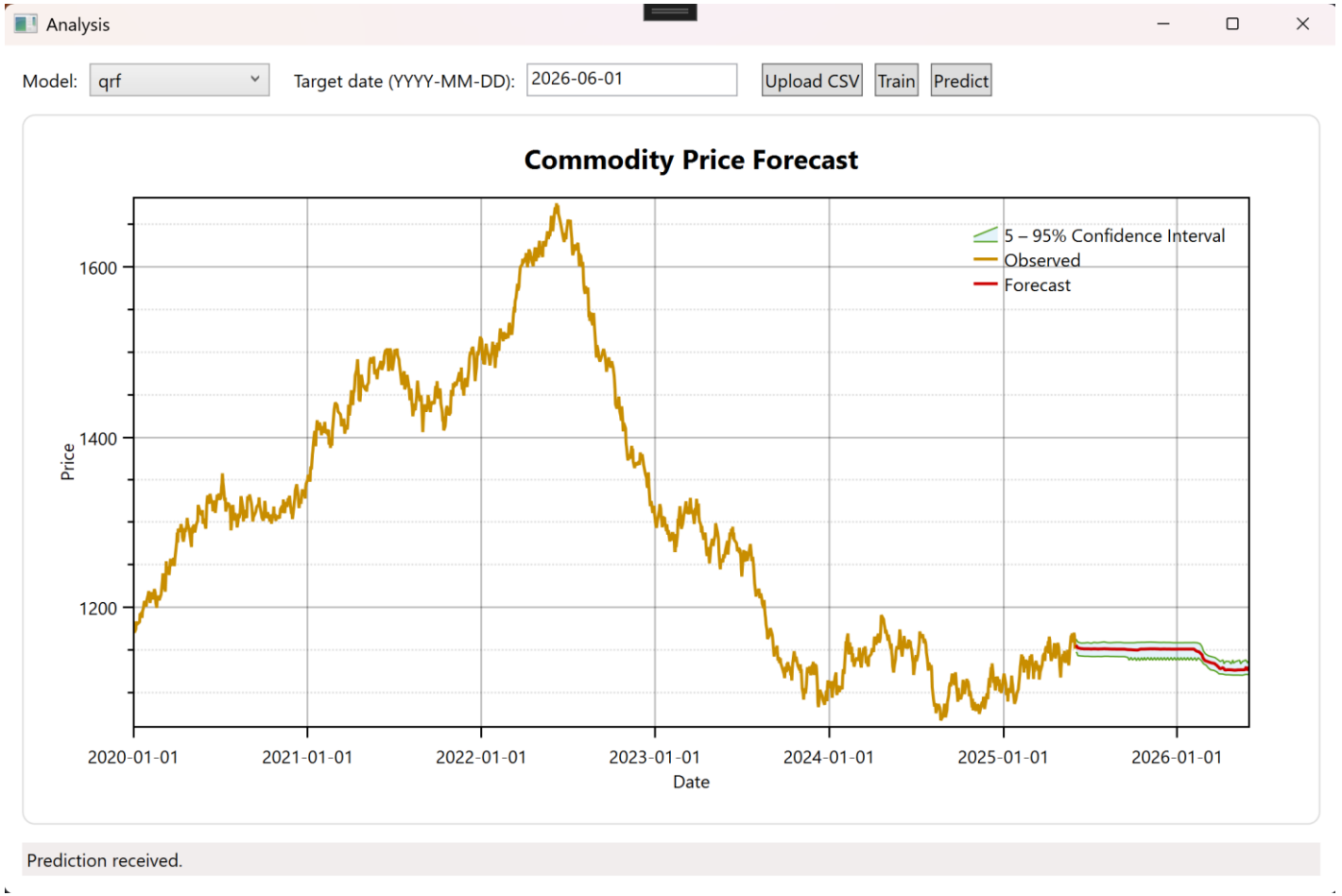
Appendix H System Architecture



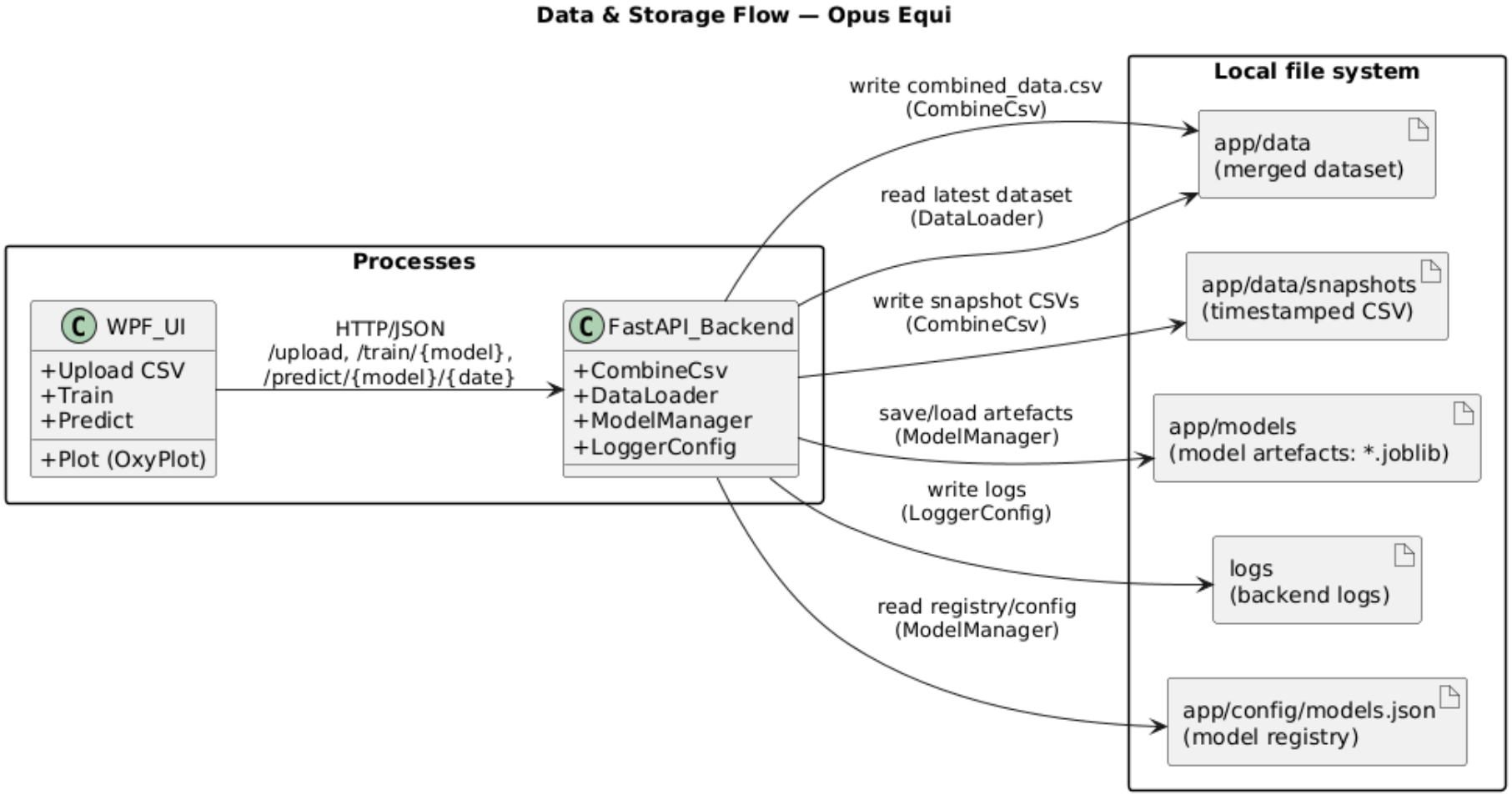
Appendix I Frontend UI Wireframe



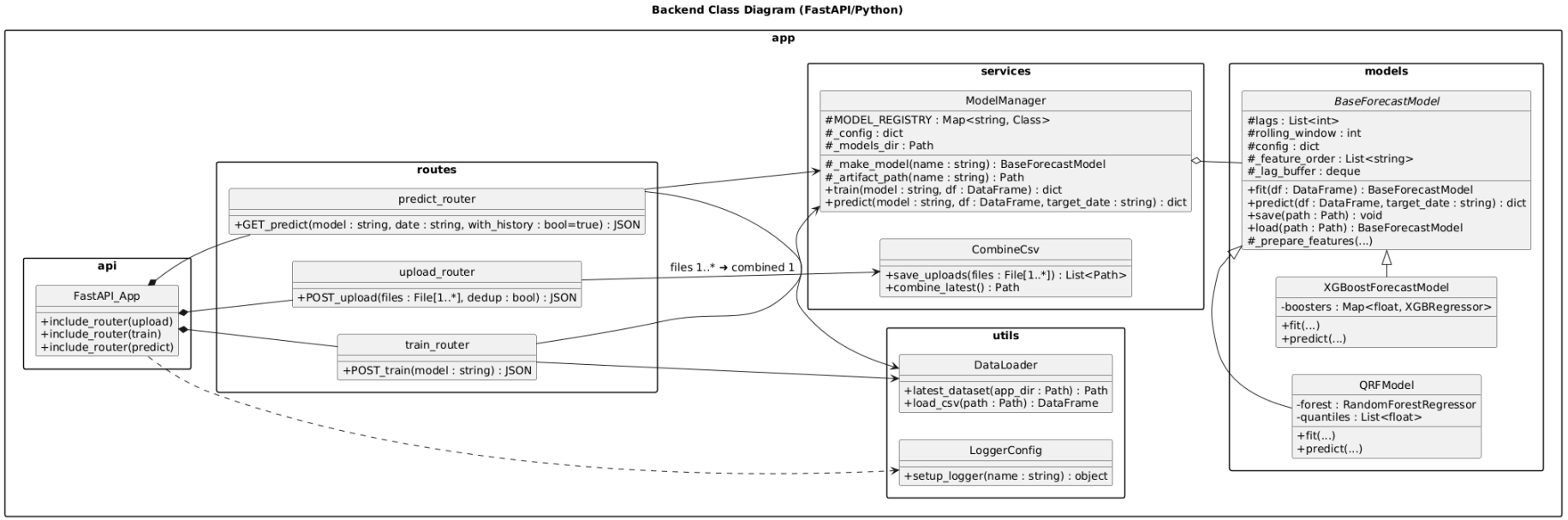
Appendix J Actual Frontend UI



Appendix K Data and Storage Flow



Appendix L Backend UML



Appendix M Frontend UML

