aiimi

FRC Codex Blueprint.

DATA
DIGITAL
AI

WORKPLACE
AI THAT MEETS
BUSINESS

Delivering a Scalable and Trusted Financial Data Platform

Contents

Overview

Executive Summary

This document outlines a practical and reusable blueprint for building a modern, insight-driven data platform, one that is specifically designed to help regulatory bodies and public sector organisations confidently integrate, manage, and report on XBRL data without incurring unsustainable costs or operational overhead.

At the core of CODEx is a cloud-native, modular platform that leverages trusted, industry standard components, supported by a robust medallion architecture (bronze, silver, gold layers) that structures and refines data through every stage of the lifecycle. This allows:

- Seamless ingestion and processing of XBRL files alongside other structured and semi-structured formats (e.g. CSV, JSON, XML).
- Embedded data quality, transformation, and validation logic using scalable Spark-based pipelines.
- Direct exposure of analytics-ready datasets for downstream reporting tools, including Power BI, through secure and performant access layers.

What distinguishes CODEx is not just the technology stack—it is the way in which the architecture has been designed with repeatability and scalability along with incredibly low overheads. The platform is designed to allow a rapid implementation, creating with an MVP environment that can be utilised quickly, and which can grow iteratively based on emerging needs and available budget for short term success which can stand the test of time. By aligning to industry best practices in cloud resource management, governance, and access control, the solution minimises unnecessary complexity and overly cumbersome maintenance.

This blueprint is intended to be agnostic of technology but does focus on a Microsoft Azure implementation to provide a case study of a successful implementation. It is intended to be reused and adapted by other organisations with similar regulatory, compliance, or analytical mandates. Every element, from the definition of environments (development, testing and production) to the handling of XBRL parsing and transformation, has been designed to provide a strong foundation for others to review, follow and amend to suit their needs. This reduces delivery risk, accelerates time to value, and ensures consistency across data platforms that share common goals.

By following this blueprint, organisations can not only gain control over XBRL data, but do so with a proven, budget-conscious approach that supports long-term scalability and interoperability. CODEx empowers organisations to build data infrastructure that is not only modern and compliant, but also ready for the future.

Value Statement

The CODEx platform delivers a centralised, cloud-native foundation for the secure, automated, and scalable management of financial data. Purpose-built for regulatory environments, it ingests and processes a wide array of data formats—including complex standards such as

XBRL—while ensuring that data quality, security, and governance are upheld throughout its lifecycle.

Through extensive automation of ingestion, validation, and transformation, CODEx reduces operational risk, minimises manual handling, and accelerates the delivery of actionable insights. Its medallion architecture provides full data lineage and auditability, aligning with regulatory expectations and supporting transparency.

Designed for flexibility and reuse, the platform supports phased implementation and cost-effective scaling through Azure-native services. Its architecture promotes consistent data management practices across environments and agencies, offering a repeatable, future-proof solution.

With integrated analytics and self-service reporting tools, CODEx empowers data scientists, analysts, and executives to derive insights confidently and quickly, enabling faster, more accurate decisions and fostering a culture of data-driven oversight.

Key Outcomes

- Centralised and scalable data platform using modern, cloud-native infrastructure
- Automated ingestion and processing of diverse data types, including CSV, JSON, and XBRL
- Secure, role-based access control with private networking and regulatory-grade compliance
- Clear data lineage and governance via medallion architecture (Bronze, Silver, Gold layers)
- Interactive dashboards and self-service reporting through Power BI and SQL integration
- Reduced operational risk through automated pipelines and environment standardisation
- Improved regulatory compliance, auditability, and oversight
- Phased implementation model supporting early value delivery and budget control
- Repeatable and extensible framework suitable for broader agency and industry adoption
- Foundation for future innovation in advanced analytics, AI, and cross-agency collaboration

High Level Requirements

Below is a consolidated list of functional and non-functional requirements for the CODEx system. The list covers the specific needs of CODEx and reflects best practices for a data platform

Functional Requirements

- **Data Ingestion**

    - Handle structured, semi-structured, and unstructured data (e.g. CSV, JSON, XML, XBRL)
    - Support automated, scheduled, and on-demand data loads
    - Enable extensibility for new data formats and sources
    - Allow for a staging environment for manual data validation prior to moving to production
- **Data Storage**
    - Store raw XBRL filings
    - Process and store data in a format optimised for reporting
    - Provide secure, scalable cloud-native storage
- **Data Processing**
    - Automate data validation, cleansing, transformation, and enrichment
    - Support batch and real-time data processing pipelines
    - Enable flexible pipeline orchestration using modular and reusable components
- **Data Access & Consumption**
    - Provide governed access to curated datasets for analysts and data scientists
    - Enable integration with reporting and BI tools (e.g. Power BI, SQL)
- **Monitoring & Alerting**
    - Log all operations and generate alerts for anomalies, failures, or security breaches
    - Provide or integrate with monitoring tools

Non-Functional Requirements

- **Automation**
    - Automate routine operations such as ingestion, validation and transformation
    - Minimise manual intervention to improve efficiency and reduce risk
- **Security & Compliance**
    - Implement Role based access
    - Encrypt data at rest and in transit
    - Align with any GDPR and other regulatory requirements
    - Secure all endpoints and network traffic
- **Scalability & Performance**

    - Offer fine grain control to scale storage and compute independently
    - Support increasing user and data volumes without performance degradation
- **Reliability & Availability**
    - Meet or exceed 99% availability
    - Consider disaster recovery and backup strategies
- **Maintainability & Flexibility**
    - Support modular code and reusable components
    - Use industry standard components where possible for ease of support
    - Consider using Open-Source software
    - Facilitate ease of updates, environment promotion (Dev → UAT → Prod), and maintenance
- **Data Governance & Documentation**
    - Maintain lineage, metadata, and audit trails across all processing steps
    - Provide comprehensive documentation for users and administrators
- **Cost Efficiency**
    - Use consumption based cloud services to optimise cost
    - Review costs as the system grows to maintain this optimal cost benefit
- **Testing & Quality Assurance**
    - Define clear data quality rules and automated validation checks
    - Enable UAT and regression testing before production release
- **User Experience**
    - Provide intuitive access to curated data for different user personas
    - Offer self-service capabilities for approved users

Team Composition

Delivering the CODEx platform required a multidisciplinary team, combining strategic oversight, technical expertise, and operational delivery. The project brought together a range of professionals who collaborated across disciplines to ensure the platform aligned with evolving regulatory and business needs.

The core team included:

- Project Managers, with a main project manager in FRC who co-ordinated all internal and external work packages, and project managers within each supplier. Together they collaborated to coordinate the delivery roadmap, managed timelines and resources, and ensured effective communication across internal and external stakeholders.

- Enterprise and Solution Architects, who defined the overall platform architecture, aligned it to organisational standards, and ensured technology choices supported long term goals, scalability, and integration.
- Business Analysts, who engaged with users and stakeholders both directly and through data sprints to capture detailed requirements, refine use cases, and ensure that platform capabilities aligned with real world needs.
- Data Analysts, who help define, transform and test the queries and visualisations, who validated outputs, and ensured end-users could derive actionable insights.
- Data Scientists, who designed advanced models and analytical processes, contributing to the platform's capacity for future innovation and predictive capabilities.
- Developers and Data Engineers, who built the automated pipelines and designed advanced models and analytical processes, contributing to the platform's capabilities.
- Operations Teams, who advised, built and provided access to the cloud tenancy, and who have ownership of the code repository.

Additionally, stakeholders from across the organisation played an integral role. These included senior leadership, data owners, and regulatory experts who provided strategic direction, prioritised functionality, and validated the platform against both internal standards and external regulatory expectations.

Crucially, collaboration with third-party delivery partners, Aiimi and an XBRL Parser Service Provider, was fundamental to the success of CODEx. These partners contributed deep technical expertise, particularly in cloud-native development, XBRL data handling, and automation. Their role was not just to build, but to co-create, working side-by-side with the internal team as part of a unified delivery model.

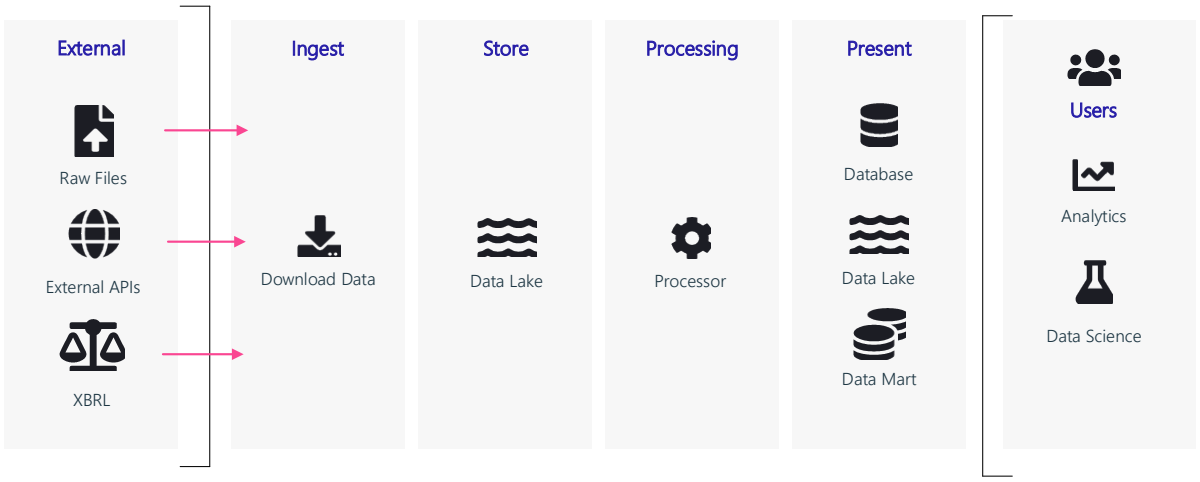
The team operated under agile delivery principles, allowing for flexibility and iterative delivery through structured sprints. This approach fostered a responsive, user-centric development cycle while ensuring that major milestones, such as the Minimum Viable Product (MVP), were achieved on schedule.

While the above provides a wide range of people and roles that were involved in the project, by following the blueprint, the need for some of these roles would diminish, and the team could potentially be smaller. However, each had a crucial part to play in the delivery of the final product.

Architecture and Delivery Approach

Conceptual Overview

To meet the needs of the FRC, a conceptual architecture was developed. The concept was to build a standard data pipeline to ingest, process, store and present data.



Data could be downloaded from multiple data sources. These data sources may be API's they may be external file shares, or they could even be manually provided files. A factory will be built to ingest the raw data and store it in its raw format for auditability and traceability. The data is then processed, either by internal systems, or by calling external parsers, in particular for XBRL. Then finally the processed data is stored in either or all of a database, a data lake, or a data mart which has a further semantic layer over the top, to make this easier to process for end users.

The end users have a multitude of different needs. For this reason, the main access to the data is through the strongly typed and controlled data mart which will be used for the majority of the business intelligence needs. Data scientists and more advanced users with the correct privileges will have direct, read only access to raw data in the data lake, to processed data in the data lake and to the SQL instance.

XBRL Parsing

Even at this early stage of the project, understanding how the XBRL files would be accessed and parsed was a major architectural decision that needed to be made before we could progress. There was a need to only download details of specific companies that met the required criteria, but the data has to be downloaded in the first place to see if the company meets that criteria. Parsing that data quickly and efficiently was also an essential need for the success of the project.

As such the team explored three primary options for implementing XBRL parsing functionality. These included: building a proprietary parser in-house, using an open-source solution, or partnering with a third-party supplier. Each option was assessed based on development effort, risk, maintainability, cost and alignment with project timelines.

Developing a custom parser internally would have offered complete control and deep integration, but at a high cost in both time and resource investment. It would have required the team to build significant domain expertise in XBRL schema design, validation logic, and regulatory frameworks, introducing a steep learning curve that could delay delivery and divert focus from other core objectives. The time to deliver an MVP for use within data sprints would also have been impacted.

Open-source parsers presented a potentially lower cost alternative, but raised concerns about robustness, long-term support, and compatibility with specific taxonomies used in the regulatory environment. While some libraries showed promise, they lacked the enterprise grade documentation, support, and assurances and equally presented a great risk to the project.

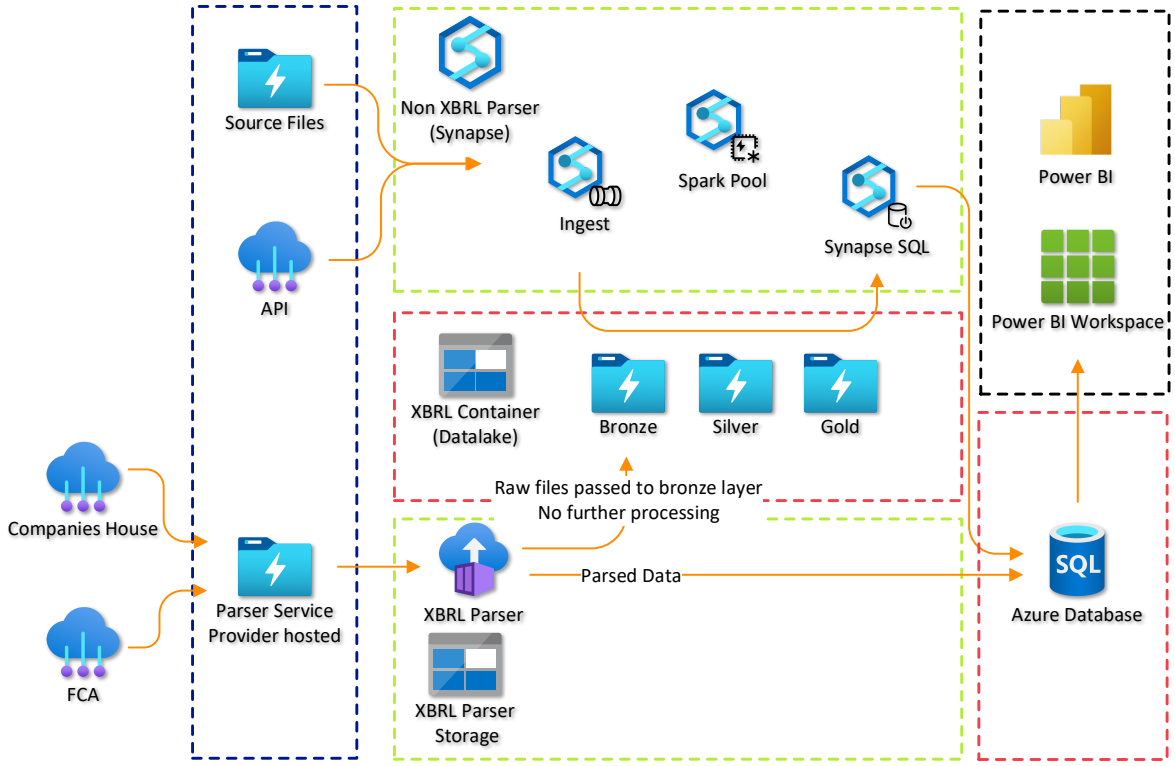
After thorough evaluation, the project team chose to partner with a third-party supplier that had a strong track record of delivering XBRL solutions in regulatory and financial sectors. This supplier provided a production ready parser that was actively maintained, tested across a range of XBRL variants, and capable of rapid integration into the platform's ingestion pipeline. In addition, because the supplier already downloads all of the required filings, they are able to pre-filter the data, and only present the companies of interest.

Outsourcing the parsing task to a service provider enabled the internal team to focus their efforts on presenting and utilising the data, rather than building the foundational tools to extract it. More time could be spent on data modelling, quality assurance, user access, and dashboard development ultimately contributing to a better user experience and faster time to delivery. The parser service provider were also willing to offer some bespoke capabilities, so that not only was the data parsed and stored in a database, they also copy the raw XBRL files to the data lake.

Design Principles and Foundational Architecture

At the heart of the CODEx platform lies a modular and scalable architecture, developed with extensibility and operational resilience as guiding principles. As the FRC are a Microsoft house, the decision was made to build the solution in a Microsoft Azure tenant. The foundation is built upon Microsoft's Azure ecosystem, leveraging a series of PaaS components to reduce operational overhead while maintaining strong control over data security, governance, and integration.

The overall architecture of the system is shown below:



The design offers a minimal set of components, which are scalable in their nature and cost effective in their implementation.

A Microsoft Datalake Gen 2 was used to store all of the files. This provides a fast, reliable and massively scalable solution while remaining very cost effective for the amount of data. The datalake is organised using the medallion architecture to store data in it's various processed states. This will be discussed later in the document.

Azure Synapse was chosen as the data processor. The choice here was to offer a complete eco system that was both capable of ingesting data from remote locations and APIs, as well as offering a rich set of processing tools underpinned by Spark pools. Synapse is a PaaS service, ensuring good availability and reliability.

Azure SQL Server was the chosen database of choice. This again provides the reliability, scalability, performance and cost effectiveness that was desired. By implementing a serverless version of Azure SQL, and because the processing only happens in short, intense bursts of activity with a very low frequency, then the overheads are minimal. If the usage pattern were to change, then it is likely that a dedicated instance would be introduced, but this was not necessary.

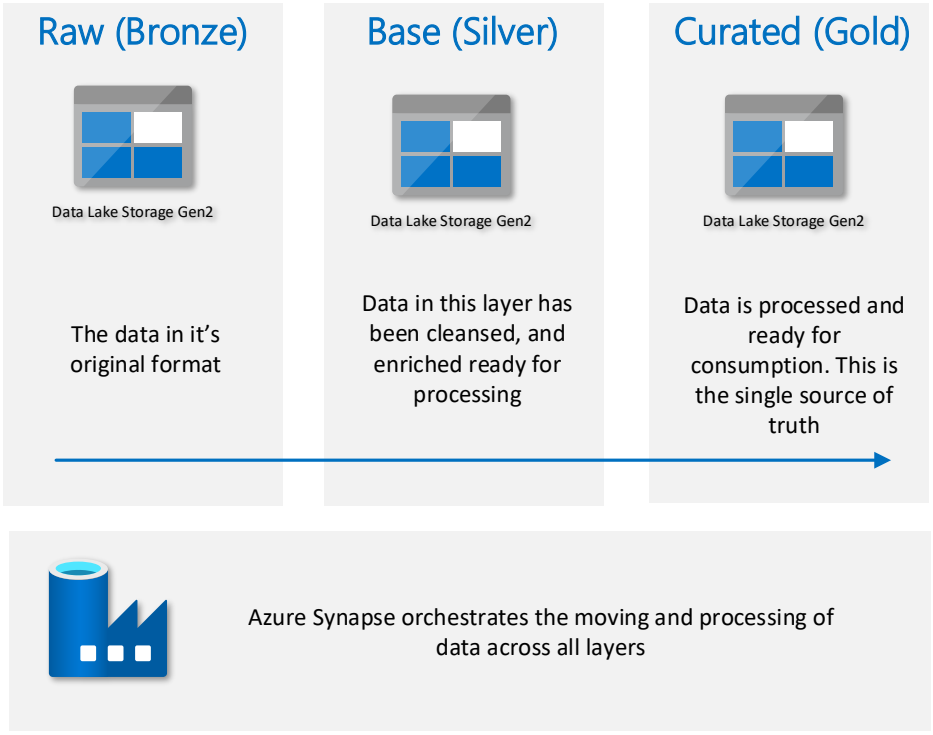
Finally, the parser service provider architecture dictated the need for a Kubernetes cluster to deploy their containers into. The containers sit inside of the network and call out to retrieve the filings from a centralised server. The Kubernetes cluster is small and is limited to prevent costs

from escalating, however in practise, there has been no need for any scaling, and so again this meets all of the criteria.

Power BI was the chosen tool to provide reporting capabilities. By implementing a shared workspace with a carefully crafted semantic layer, then all of the data can be presented to the users in a very digestible format. Again, by maintaining the data inside of the Power BI Workspace offloads a lot of processing away from the core architecture and ensures a responsive and reliable experience for users.

Data Ingestion and Processing Pipelines

The solution was designed around the concept of the medallion architecture, which structures the data lake into three progressive layers: Bronze, Silver, and Gold. Each layer serves a distinct function within the data lifecycle. The Bronze layer captures raw ingested data in its native format, ensuring that no fidelity is lost. The Silver layer introduces cleansing and validation rules, resulting in a curated dataset that is ready for transformation. Finally, the Gold layer contains the business-ready data models, structured to support reporting and advanced analytics.



This layered approach not only promotes data quality and traceability but also allows differentiated access for various user groups, enhancing both security and usability. The architecture supports structured and semi-structured file types, such as CSV, JSON, and XBRL, and is designed to be extensible to future data formats, including unstructured sources.

The ingestion of data into CODEx is orchestrated through Azure Synapse Pipelines, which are configured with scheduled triggers to collect data from remote providers. Each ingestion pipeline handles the staging of incoming files within the data lake's raw zone, using metadata-driven logic to support scalable onboarding of new datasets.

Once ingested, Apache Spark pools within Synapse are used to run validation routines, applying schema checks, integrity constraints, and business rules. These Spark jobs are embedded within notebooks, allowing for reusable, version controlled logic. Cleansed data is then moved to the Silver zone of the data lake, where additional enrichment steps take place. This includes joining datasets, normalising formats, and aggregating figures according to regulatory or analytical needs.

After processing, the data is transformed into dimensional models suitable for consumption. These models are loaded into the Gold zone. This data is then extracted into the SQL Server instance to enrich the XBRL data. Further processing is executed in SQL to provide the final dataset, which is exposed to the Power BI workspace for consumption. A mixture of raw and curated tables and a set of views are exposed. These can take some time to build due to the complex calculations. By curating this data in advance, and by leveraging the Power BI Workspace, we can ensure that the users are able to report on their data quickly and efficiently and are abstracted from any of these time consuming complexities.

The structure of the pipelines and notebook within synapse are very well defined to ensure that it is intuitive for any developer to work with. The ingestion and processing of each data source consists of:

- An ingestion notebook to download from the API
- A Bronze to Silver pipeline to copy and cleanse the data
- A Silver to Gold pipeline to present the data
- An orchestration pipeline which ensures each of these are ran in sequence, and to manage any failures.

Infrastructure and Environment Management

Infrastructure is provisioned using a tri-environment setup: Development, UAT, and Production. Each environment operates within its own Azure subscription, maintaining clear separation of concerns and reducing risk during deployments. Network security is enforced through the use of Azure VNets and private endpoints, ensuring that data flows exclusively within a controlled perimeter.

Authentication and authorisation are managed using Microsoft Entra ID (formerly Azure AD), with role-based access control (RBAC) applied at all resource levels. Managed Identities are leveraged to simplify secure access to services like Key Vault and Storage, reducing the reliance on embedded credentials.

To assist with the setup of the environments, and managing authorisation to the various components and datasets, a user access matrix was developed. This outlines all of the components and their capabilities for each environment, cross referenced against a set of user groups. The creation of this matrix can be difficult at first, but this was an essential tool to ensure the security and integrity of the systems are maintained across environments. It was also essential for assurance purposes to verify that everything was setup correctly.

Monitoring, Governance, and Operational Continuity

To support operational excellence, the platform includes a robust monitoring and alerting framework. Azure Log Analytics captures telemetry from all key services, which is then

analysed by Microsoft Sentinel to provide real-time alerting on anomalies, security breaches, and performance issues. Dashboards are used to track ingestion success rates, processing durations, and usage patterns.

Data governance is implemented via a combination of metadata catalogs, audit trails, and access control policies. Every data movement is logged, and transformations are versioned to ensure lineage can be reconstructed at any point.

Disaster recovery measures include geo-redundant storage, automated backups of metadata and code artifacts, and documented runbooks for platform recovery. These controls ensure that CODEx can continue to function reliably in the face of operational disruptions.

Integration with Delivery Partners

The implementation was delivered through a close partnership model, where third-party vendors were aligned with internal teams through a shared delivery plan and joint ownership of outcomes. Workstreams were managed using agile methodologies, with two-week sprints focused on achieving iterative milestones across ingestion, modelling, and reporting.

Clear separation of responsibilities was documented helping to coordinate activities across design, development, testing, and release. Knowledge transfer sessions were conducted throughout the build, ensuring that internal teams are well-positioned to maintain and evolve the platform post-implementation.

Together, these design and delivery practices enabled the successful implementation of a modern, cloud-native platform that is both resilient and future-proof.

Customisation and Extensibility: Empowering Data Consumers and Developers

Philosophy of Adaptability and Modularity

One of the defining characteristics of the CODEx platform is its inherent flexibility. From the earliest stages of the design process, the system was conceived not as a fixed end-state, but as a framework that could be extended, customised, and adapted to suit future use cases. This design philosophy enables organisations to respond to evolving regulatory demands, integrate new data sources, and support bespoke analytical needs without overhauling the underlying infrastructure.

The medallion architecture, in particular, supports this philosophy through its clear separation of concerns across data layers. Each layer represents a discrete stage of data maturity, and each stage has well-defined interfaces and processing logic. This modular approach ensures that customisation can occur at the appropriate level of the pipeline, without introducing unnecessary complexity across the platform.

Integration of New Data Sources

As financial and regulatory ecosystems evolve, so too does the variety and format of data that must be ingested and analysed. CODEx has been architected to facilitate the straightforward integration of new data sources, whether these are additional XBRL feeds, CSV extracts, JSON APIs, or unstructured textual reports.

Data onboarding is governed by metadata-driven pipelines, which define ingestion rules, file schemas, and validation logic. These definitions are stored in configuration files, enabling new sources to be incorporated without the need to redeploy code. By separating configuration from execution, CODEx empowers developers and data engineers to rapidly iterate on data integration while preserving platform stability.

Moreover, this metadata-first approach supports data versioning and rollback, ensuring that failed ingestion attempts do not compromise data quality downstream. Validation rules and transformation mappings can be tailored per source, allowing each data provider's output to be treated uniquely, yet processed consistently within the broader architecture.

Extending the Semantic Layer

The semantic layer of the CODEx platform plays a crucial role in translating processed data into business-relevant models. This is where business rules, calculation logic, dimensional hierarchies, and metric definitions are formalised. Built primarily within Synapse SQL pools and exposed through tools like Power BI, this layer is the primary interface for analysts and non-technical users.

Customising the semantic model involves creating new views or extending existing models to include new dimensions or measures. Furthermore, semantic enrichment enables the platform to support multi-tenant analytical environments. Different user groups or regulatory partners can receive curated views of the same underlying data, tailored to their specific informational

needs. This capability ensures that the platform can be adopted by a broader ecosystem of stakeholders, without duplication of effort or loss of data fidelity.

Developer Enablement and Self-Service Capabilities

To support long-term sustainability and innovation, CODEx has been designed with developer enablement in mind. A suite of APIs, notebooks, configuration templates, and orchestration components are exposed to internal developers, allowing them to build on top of the core platform.

Developers can define new ingestion workflows, build additional Spark-based transformation steps, or integrate third-party libraries for natural language processing, forecasting, or anomaly detection. Each of these additions is governed by platform-wide policies for testing, documentation, and deployment, ensuring that the platform remains coherent even as it evolves.

Beyond technical users, self-service capabilities have also been prioritised. Analysts can use Power BI to explore datasets, build custom dashboards, and perform ad-hoc analysis, all without needing to understand the underlying pipeline logic. Role-based permissions control access to datasets and tools, ensuring that sensitive information remains protected while empowering users with meaningful insights.

Use Case Evolution and Future Innovation

Perhaps the most important element of customisation is the platform's ability to evolve alongside business needs. CODEx is not a static product but a dynamic capability. As new financial instruments emerge, as regulatory requirements shift, or as internal business priorities change, the platform provides a ready foundation upon which new use cases can be built.

This includes the potential for integrating machine learning models for predictive analytics, incorporating natural language processing to extract data from unstructured regulatory filings, or enabling automated anomaly detection in financial submissions. The modularity and openness of the platform ensure that these innovations can be layered onto existing pipelines, rather than requiring wholesale redesigns.

By embracing extensibility and encouraging participation from both developers and business users, CODEx is positioned not only to meet today's requirements but to become the foundation for tomorrow's data innovation.

Operational Considerations: Security, Maintenance, and Lifecycle Governance

Security and Access Control

Security underpins every facet of the CODEx platform, from the ingestion of external data to the delivery of analytical insights. The platform has been architected with a zero-trust mindset, ensuring that only authorised users can access the data they are permitted to see, and only under controlled and auditable conditions.

Identity and access management are enforced through Microsoft Entra ID (formerly Azure Active Directory), with fine-grained role-based access control (RBAC) applied across all resources. Users are grouped according to organisational roles, and these groups are mapped to data access layers (e.g., Bronze, Silver, Gold) and functional privileges (e.g., read-only, contributor, administrator).

All access to data is logged and monitored, with periodic reviews conducted to identify dormant accounts or role mismatches. Sensitive credentials, such as connection strings and API keys, are managed centrally through Azure Key Vault, ensuring that secrets are not hard-coded or exposed within the platform.

Network security is achieved through the implementation of Azure Virtual Networks, private endpoints, and firewall rules. Data transmission occurs strictly within private network boundaries, eliminating the risk of data leakage via public endpoints. End-to-end encryption is enforced both in transit and at rest, adhering to industry standards for cryptographic protection.

Platform Maintenance and Support

Once the platform was delivered and stabilised, attention turned to the processes and responsibilities required for ongoing support. CODEx is a living system, and its effective operation depends on routine maintenance, performance optimisation, and continuous monitoring.

Testing, Validation, and Promotion Through Environments

To ensure confidence in the data and the platform's capabilities, rigorous testing protocols have been embedded into the development lifecycle. Every data source integrated into the platform is subject to validation checks at multiple points: during ingestion, during transformation, and prior to consumption.

For significant changes or newly onboarded datasets, User Acceptance Testing (UAT) is conducted in an isolated environment, allowing business stakeholders to evaluate correctness and usability before promoting to production.

Promotion through environments (Development to UAT to Production) follows a manual release process. Deployment pipelines were considered, and are recommended for future use, but were deemed unnecessary, and an additional cost overhead, that was not needed for the MVP.

Platform Selection and Technology Decisions

The choice to build CODEx on Azure was driven by multiple considerations: alignment with the broader enterprise cloud strategy, availability of advanced PaaS services, scalability, and strong compliance credentials.

Azure Synapse Analytics was selected for its unified approach to data integration, warehousing, and big data analytics. It supports both serverless and provisioned models, offering flexibility in cost and performance. Azure Data Lake Storage Gen2 was chosen for its hierarchical namespace and seamless integration with Synapse, supporting medallion architecture at scale.

The use of Apache Spark within Synapse allowed for high-performance, parallel processing of large datasets, and facilitated complex data transformations. Meanwhile, Power BI provided a mature and widely adopted analytics layer, enabling business users to interact with the data in intuitive and dynamic ways.

The decision to use an XBRL parser as part of the ingestion workflow was a strategic one, given the format's prevalence in financial reporting. Parsing XBRL accurately and efficiently required specialist expertise, and the platform's modular architecture made it possible to integrate this functionality without disrupting core components.

Together, these decisions have resulted in a robust, extensible, and secure platform that meets both current and future demands of financial data processing.

All chosen components are quite generic and replaceable in their nature. Should the blueprint be used to build a system in a different cloud eco system, then similar components ought to be available which are direct replacements for the architecture built for the FRC. There is nothing here which could not be replicated or changed by experienced architects.

Other Considerations

Collaboration and Third-Party Engagement

Engaging with third-party suppliers can be a highly effective strategy to accelerate project delivery, particularly when internal capacity is constrained or certain specialist skillsets are not available in-house. Whether due to competing priorities, lack of domain expertise, or the complexity of the project at hand, leveraging external support enables organisations to maintain momentum without compromising quality.

Before seeking external partners, it is critical to perform a realistic assessment of internal capabilities, resource availability, and the desired project outcomes. This assessment forms the foundation of a make-versus-buy decision, helping to determine whether third-party involvement is necessary, and if so, in what capacity.

While it is entirely feasible for an organisation to develop and deliver an XBRL platform using internal resources, this project has demonstrated the tangible benefits of engaging external professionals. Third-party suppliers with proven track records in similar implementations can bring immediate value through deep subject matter expertise, established delivery frameworks, and familiarity with relevant technologies and compliance standards.

Furthermore, selecting partners who not only possess the technical capabilities but also have a collaborative mindset, working effectively alongside internal teams and other suppliers, can significantly reduce delivery risk. This collaborative dynamic helps to shorten timelines, streamline integration, and ultimately lower the overall cost of delivery. When well-managed, third-party engagement acts not merely as an outsourced function, but as a strategic extension of the internal team.

Working with both Aiimi and the parser service provider was a perfect partnership. Aiimi has a long history of delivering data platforms particularly within an Azure environment, while the prime function of parser service provider is to parse and deliver XBRL data.

Environment Mangement

At the outset, the project was in an exploratory phase, requiring rapid iteration and flexibility. To support this, a sandbox environment was established where both developers and users could work collaboratively within a single shared space. This setup eliminated the need for complex deployment processes, as all code was accessible and operational within the environment. It also helped keep infrastructure costs to a minimum. With appropriate oversight and a clear understanding that the environment was non-production, this approach worked effectively for a small team operating under a defined quality assurance process.

As the project matured and required a transition toward a production-ready platform, the architecture was expanded to include three distinct environments: development, test (UAT), and production. Given the low frequency of data refreshes, typically no more than once per quarter, a pragmatic data flow strategy was adopted to minimise cost and operational complexity.

In this setup, a single instance of live data is loaded into the UAT environment, where it is parsed and validated. This process ensures any issues, whether related to data integrity or processing logic, are identified before promotion to the production environment. Once data passes quality checks, it is then propagated to production, with the option to replicate it into the development environment if required. It is important to note that this promotion process involves only data, not code.

This strategy has proven effective for maintaining clean, reliable data in production while keeping operational overheads low. However, it introduces limitations when new code is being tested. In such cases, testers often require the ability to load specific test cases that may not align with the current live data. To address this challenge and improve future scalability or if the CODEx platform continues to evolve, two enhancements are recommended:

1. **Introduce a Pre-Production Environment**
   This dedicated environment would act as an intermediary stage for loading and validating live data before it reaches production. It would allow UAT and test environments to be populated with synthetic or manipulated data needed for targeted testing, without risking contamination of the production dataset.
2. **Enhance Ingestion Pipelines with Automated Data Quality Checks**
   Incorporating automated validation within the production ingestion routines would allow the system to identify anomalies and trigger rollback procedures if issues are

detected. This would increase confidence in production data integrity and further reduce the risk of polluting the production data and introducing errors.

These improvements would support a more rigorous and flexible approach to data management, particularly as the platform scales or takes on more frequent update cycles.

Restated Accounts

Financial data submitted to regulatory bodies such as Companies House can be updated or corrected post-submission. Systems must account for this by identifying and reporting against the most appropriate version of a company's accounts. This requires a reporting structure capable of differentiating between original and amended filings and selecting the most accurate representation for analytical use.

The CODEx platform was made aware of restated accounts from the outset, but anyone working with this data must also be aware, as it could severely impact the output.

Building an Effective Semantic Layer

The semantic layer bridges technical implementation and business understanding. It is critical to involve end-users early to ensure that data is modeled in a way that reflects business needs. A well-designed semantic layer allows for intuitive reporting and helps avoid the proliferation of inconsistent or duplicative metrics across the organisation.

Platform Efficiency and Cost Management

Cost efficiency was a central consideration in the design and implementation of the CODEx platform. The initial build was executed entirely within a sandbox environment, with additional environments introduced only when appropriate. This staged approach helped reduce infrastructure expenditure significantly over the course of the development lifecycle.

Every component selected for the platform was deliberately configured with the smallest viable capacity. Wherever possible, serverless technologies were chosen to ensure that resource usage, and therefore cost, was incurred only when the system was actively in use. This strategy made full use of cloud-native pricing models, and is recommended as a best practice for similar projects. As a result, the ongoing monthly cost of maintaining the Azure tenancy remained exceptionally low.

An early architectural decision involved evaluating whether to implement a CI/CD pipeline for managing code deployment across environments, or to adopt a manual approach. Given that most of the development activity took place within a single environment, and did not require frequent or complex promotion, the manual method was selected. The cost and effort associated with establishing a full CI/CD pipeline, including integration within a secure network boundary, were considered disproportionate to the needs of the MVP phase.

In retrospect, this was the correct decision for the scope and duration of the project. Only one round of deployment was required across development, UAT, and production environments, and all SQL and Synapse code was accurately and efficiently replicated using scripted methods. However, if the CODEx platform is extended in future, and development activity becomes more frequent or distributed, it would be highly advisable to implement automated tooling.

In such cases, solutions such as Terraform, Azure DevOps Pipelines, or GitHub Actions can offer significant benefits, enabling consistent environment setup, automated deployments, and tighter control over release management. This approach is especially valuable as platform complexity increases and the need for structured change governance becomes more critical.

Version Control

All development work should be maintained in a source control system, supporting collaboration, rollback, and auditability. This also supports smooth transitions across environments, aids collaboration between teams and simplifies the development lifecycle.

Resilience and Disaster Recovery

The CODEx platform has been designed with a foundational level of resilience, targeting availability levels of 99 percent or higher. This level of resilience is entirely appropriate for the current scope of the platform, which serves as a Minimum Viable Product (MVP) and is not yet classified as a business-critical system. The primary objective during this phase was to validate core concepts and demonstrate value, while keeping complexity and operational overhead to a minimum.

One of the key considerations in maintaining this simplified resilience model was the nature of the platform's components and data. All data ingested into CODEx is sourced from publicly available repositories, meaning it can be redownloaded at any time without incurring loss or breach. In parallel, all code used to orchestrate and operate the platform is maintained in a centralised source control repository. This includes all scripts, pipelines, configurations, and SQL logic. As a result, in the event of a total system failure or loss, the entire platform can be rebuilt and restored with relative ease.

This approach ensures that, while the platform does not include enterprise-grade disaster recovery features such as geo-replication or automated failover, it still offers a high degree of recoverability and operational continuity. For the current use case, this balance between cost, complexity, and resilience is appropriate and strategically sound.

However, should the platform evolve into a production system with more frequent usage, mission-critical dependencies, or real-time reporting demands, a higher level of resilience would be necessary. In such scenarios, implementing geo-redundancy for key components, such as Azure Storage Accounts and SQL Server instances, would be advisable. This would safeguard the platform against regional outages and support business continuity.

It is important to note that enhancing resilience through geo-replication does introduce additional architectural considerations. These include increased infrastructure costs, more complex virtual network topologies, and the need for careful orchestration of automated failover mechanisms. The decision to implement such measures should be based on a clear risk assessment, aligned with business impact and continuity requirements.

Parser Selection

A critical component of the CODEx platform is its ability to process and interpret financial data submitted in XBRL (eXtensible Business Reporting Language) format. The XBRL parser plays a foundational role in this process, acting as the translation layer that converts structured XBRL filings into usable data models for analysis and reporting. Without a reliable parser, the

platform would be unable to ingest, standardise, or analyse one of its primary data sources, making this component essential to the project's success.

A range of XBRL parsing solutions is available, spanning both open-source projects and commercial offerings. Early in the project, a thorough evaluation was conducted to determine the most suitable approach. This included consideration of the following options:

- Adopting an Open-Source Parser, such as **Arelle**, which is widely recognised in the industry and supports a broad range of XBRL standards.
- Licensing a Commercial Parser, which might provide enterprise-grade features, dedicated support, and integration tooling.
- Building a Custom Parser in-house, offering full control over features and alignment with project-specific needs.

Each approach brought its own advantages and risks.

Open-source tools, particularly Arelle, were strongly favoured from the outset due to their flexibility, active user communities, and the absence of licensing costs. Open-source solutions align well with the strategic goals of transparency, customisation, and community-supported development. However, these benefits come with challenges. Projects like Arelle, while powerful, often require a steep learning curve and non-trivial effort to configure, test, and integrate into existing pipelines. The time required to gain a deep understanding of the parser's architecture, ensure compatibility with UK-specific taxonomies, and build the necessary orchestration layers introduced considerable risk to the project timeline. For a project operating within the constraints of a Minimum Viable Product (MVP), these factors could severely impact not only delivery schedules but also overall feasibility.

Commercial solutions offered greater ease of integration and typically came with service-level support, but often at a cost that was not justified for an MVP with limited scope and budget. Moreover, vendor lock-in and reduced flexibility were important concerns, particularly given the regulatory and structural complexity of XBRL data in the UK.

The third option, developing a custom parser, was also explored. While this route offered full control and potential performance benefits, it represented the highest development effort, with long lead times and significant maintenance implications. It was deemed unsuitable for the project's timeline and resource model.

In the end, the decision was made to proceed with a pragmatic and risk-aware approach. Although open-source remained the preferred model, the project acknowledged the overheads associated with configuring and operationalising such a solution in the available timeframe. As such, a working implementation that could meet immediate project needs with minimal risk was prioritised, with a view to revisiting the parser selection in future phases when the platform matures beyond MVP status.

Going forward, open-source solutions such as Arelle remain the most attractive long-term option, especially if time and resources allow for their effective adoption. Their flexibility, openness, and community-driven development model align closely with the goals of the CODEx platform and its commitment to interoperability, reusability, and public value.

The CODEx platform as a whole is reasonably modular in its design. The parsing of the XBRL data is not intrinsic to the output, and any suitable XBRL parser could be used in place of the selected parser if desired, as long as the data stored in the SQL database were to match.

Appendix A – Suggested Implementation Plan

The following is a suggested high level project plan to follow a similar approach to that of the CODEx project

Phase 1: Discovery and Planning

Objective: Define project objectives, assess internal capabilities, and prepare for delivery.

Key Activities:

- Align stakeholders on project vision and expected outcomes
- Define MVP scope and strategic priorities
- Audit available data sources and define regulatory requirements
- Conduct internal capability and capacity assessment
- Agree on environment strategy and platform governance

Key Decision Points:

- Internal delivery vs third-party support
- Priority use cases for MVP and initial exposure
- Technology baseline (e.g., Azure, open-source components)
- Metrics for evaluating success of MVP

Phase 2: Architecture Design

Objective: Design logical, conceptual, and physical architecture.

Key Activities:

- Define medallion architecture (Bronze, Silver, Gold layers)
- Design data flows, pipelines, and storage structure
- Select ingestion and transformation tools (e.g., Azure Data Factory, Synapse)
- Define environment strategy (Dev, Test, Pre-Prod, Prod)
- Decide on identity, access, and security models (e.g., RBAC, Entra ID)
- Build User Access Matrix

Key Decision Points:

- Selection of storage format (e.g., CSV, Parquet, Delta)
- Serverless vs provisioned compute
- Data latency and refresh model
- Parser selection (e.g., Arelle or alternative)

Phase 3: Foundation Build and Sandbox Development

Objective: Establish the initial sandbox environment and expose key datasets.

Key Activities:

- Build the sandbox environment with required data sources
- Implement basic ingestion, storage, and processing pipelines
- Parse and load a core set of XBRL filings
- Expose raw and enriched data via SQL endpoints, views, or simple dashboards
- Introduce version control and simple data lineage

Key Decision Points:

- Datasets to prioritise for initial exposure
- Data refresh approach

Phase 4: Data Modelling, Transformation and Reporting

Objective: Cleanse, enrich, and model data for reporting.

Key Activities:

- Apply data quality rules to Bronze data
- Create transformation logic for Silver and Gold layers
- Build validated views in SQL for consumption
- Connect Power BI or similar tools to SQL layer and / or Datalake
- Develop core dashboards and semantic layer

Key Decision Points:

- Degree of automation in data quality workflows
- Handling of restated filings
- Degree of semantic modelling required for early user engagement
- Access model for internal vs external users

- Method for semantic model development and approval

Phase 5: Data Sprints (User Exploration and Feedback)

Objective: Validate early data exposure and understand user needs through collaborative exploration.

Key Activities:

- Organise focused, short-term workshops (data sprints) with regulatory analysts, domain experts, and business users
- Present available data and demonstrate access through SQL, dashboards, or data extracts
- Gather feedback on data usability, clarity, completeness, and gaps
- Capture analytical questions, feature requests, and reporting expectations
- Identify common pain points or misunderstandings in data presentation

Key Decision Points:

- Which user groups and roles to engage in sprints
- Feedback mechanisms and capture (e.g., sprint boards, sessions notes)
- Decision on quick wins vs backlog for broader changes
- Updates to semantic layer and data models based on sprint outcomes

Sprint Outcomes:

- Improved understanding of user priorities and analytical needs
- Validated or revised data structures and reporting models
- A prioritised backlog of enhancements for future iterations
- Enhanced buy-in and engagement across stakeholders

Phase 6: MVP Refinement

Objective: Refine and finalise platform architecture and dashboards based on validated requirements from the data sprints.

Key Activities:

- Expand the medallion architecture across full dataset coverage
- Strengthen data quality controls and lineage tracking
- Refine semantic models and calculated metrics based on user feedback
- Define environment promotion workflows and prepare UAT environment

Key Decision Points:

- Adjustments to modelling strategy or schema
- Tools and frameworks for semantic layer (e.g., Power BI datasets, dbt, etc.)
- Level of governance required for user-generated content

Phase 7: Testing, Validation, and Promotion

Objective: Validate functionality, ensure data integrity, and prepare for go-live.

Key Activities:

- Create and execute test plans for each environment
- Perform performance and load testing
- Conduct UAT with business users
- Review security, compliance, and data quality checks
- Promote from sandbox to Dev, UAT, and Production as needed

Key Decision Points:

- Use of synthetic vs live data in UAT
- Criteria for go/no-go decision
- Final validation of parser and reporting outputs

Phase 8: Operationalisation and Support

Objective: Transition platform into live use, with monitoring and support in place.

Key Activities:

- Establish support model (internal or third-party)
- Deploy monitoring tools (e.g., Azure Monitor, Sentinel)
- Configure logging, alerting, and incident response
- Finalise documentation and runbooks
- Plan for continuous improvement and backlog management

Key Decision Points:

- Internal vs outsourced support
- Resilience and backup strategy (e.g., geo-redundancy)

- Rollout and onboarding plan for users


aiimi