APPENDIX B

Technology Infrastructure Summary

Platform Stack — Schema Engine Architecture — Security — Data Infrastructure

Confidential — For Informational Purposes Only — Patent Pending

This appendix provides a technology infrastructure summary of the DCXchange.net platform for licensing candidates, platform buyers, and technical reviewers conducting due diligence. It describes the platform's technology stack, the architecture of the Patent Pending Configurable Instrument Schema Engine, the security and data infrastructure, and the ongoing maintenance profile. Full technical detail — including component-level architecture diagrams, data model documentation, API specification, and build complexity analysis — is contained in the DCXchange.net Platform Technology White Paper, available separately upon request.

SECTION 1 — TECHNOLOGY STACK

The DCXchange.net platform is built entirely on the Microsoft .NET technology stack using industry-standard languages, frameworks, and data infrastructure. Every component of the platform uses widely adopted, well-documented, commercially supported technology with deep talent pools and long-term vendor support commitments. There are no proprietary dependencies, no exotic frameworks, and no technology choices that create vendor lock-in or restrict the platform to a narrow set of qualified developers.

Layer	Technology	Role and Rationale
Database	Microsoft SQL Server	Enterprise-grade relational database engine. Primary data persistence layer for all platform data — listings, participants, bids, messages, audit logs, schema definitions, and analytics aggregations. Industry standard for financial, governmental, and enterprise applications requiring ACID compliance, row-level security, and documented disaster recovery.
Web Application Framework	ASP.NET	Microsoft's mature, high-performance web application framework. Handles HTTP request routing, session management, authentication, authorization, and server-side rendering. Runs on Windows Server with IIS or on Linux with Kestrel. Extensive documentation, large developer community, long-term Microsoft support commitment.
Server-Side Language	C# / Visual Basic	C# is the primary server-side language for all platform business logic, API layer, data access layer, and Schema Engine implementation. Visual Basic is available for legacy-compatible components and rapid form logic development. Both compile to the same .NET runtime and interoperate natively.
Front-End Markup and Logic	HTML and JavaScript	Standard HTML5 markup for all platform interfaces. JavaScript handles all client-side behavior including conditional field visibility in the listing form (the Schema Engine's client-side manifestation), real-time bid chart updates via WebSocket connection, market

		activity ticker rendering, and form validation before server submission.
Real-Time Communication	WebSocket (ASP.NET SignalR)	Persistent bidirectional connection between client browsers and the server for real-time event delivery. Powers the live auction bid history chart, the market activity ticker, and push notification delivery to web clients. Native socket implementations in the iOS and Android apps connect to the same SignalR hub.
Mobile — iOS	Swift (Native)	Purpose-built native iOS application communicating with the platform API. Not a web wrapper. Renders using native UIKit components for performance and full integration with iOS device capabilities including Face ID, Touch ID, APNs push notifications, and native file system access.
Mobile — Android	Kotlin (Native)	Purpose-built native Android application communicating with the platform API. Renders using native Android UI components with full integration with Android device capabilities including fingerprint authentication, Firebase Cloud Messaging push notifications, and native file system access.
API Layer	ASP.NET Web API (RESTful)	All data access by web clients, mobile clients, and third-party API consumers goes through a single RESTful API layer. The API enforces all role and tier permissions before any database query executes. One data layer serves all access points simultaneously with no synchronization delay between web, iOS, and Android.
Presentation	CSS	Cascading Style Sheets for all visual presentation across the web platform and PWA. The DCXchange.net visual identity — Chase blue #117ACA, dark navy #1A2B4A, Arial typography, white backgrounds — is implemented entirely in CSS with no dependency on commercial design frameworks.

DEVELOPER AVAILABILITY

Every technology in the DCXchange.net stack is taught in undergraduate computer science programs, supported by major cloud platforms, and available from a large pool of experienced developers in every US metropolitan market. A licensee or acquirer who needs to staff a development team to maintain or extend the platform is not constrained to a specialized talent pool. Standard .NET web developers with SQL Server experience can operate and extend the platform from day one.

SECTION 2 — THE PATENT PENDING CONFIGURABLE INSTRUMENT SCHEMA ENGINE

The Schema Engine is the architectural innovation that makes DCXchange.net's universal instrument coverage model technically possible. Its core insight is that the field set required to describe a financial instrument is data, not code. Rather than hard-coding a separate form, a separate database table, and a separate API response structure for each instrument type, the Schema Engine stores instrument-type definitions as configuration records in the database. Every platform component — the listing form, the

search index, the API layer, the analytics engine — responds dynamically to those configuration records.

The commercial consequence of this architecture is what Amazon achieved by treating product attributes as configuration rather than as hard-coded product types: the ability to add any new instrument type to the platform in hours rather than months, without code deployment, without database schema migration, and without any disruption to existing platform operations. This is the subject of the platform's patent applications.

Schema Engine Components

Component	Function
Instrument Type Registry	Master table of all supported instrument types across all 15 categories. Each instrument type has a unique identifier, category assignment, display name, and active status. Adding a new instrument type is a database insert, not a code deployment.
Field Definition Table	For each instrument type, a set of field definition records specifying field name, data type, validation rules, display label, help text, required or optional status, display order, and conditional visibility rules. This table is the Schema Engine's authoritative source of truth.
Dynamic Form Renderer	Server-side and client-side logic that reads the field definition table for the selected instrument type and renders the appropriate input fields in the listing form. The form is constructed at runtime from configuration data, not from hard-coded HTML. A new instrument type with new fields requires zero code changes to the form renderer.
Conditional Field Visibility	JavaScript logic that reads field dependency rules from the field definition table and shows or hides form sections based on prior field selections. A seller who selects "Auction" as the sale method sees auction-specific fields. A seller who selects "Fixed Price" does not. All of this behavior is driven by the configuration table, not by if-statements in application code.
Search Index Integration	Field definition records include a search index flag that determines which fields are included in the platform's search and filter index. Instrument-specific fields marked as searchable are automatically included in the search infrastructure when a new instrument type is activated. No search code changes required.
API Response Generation	The API layer reads the field definition table to construct instrument-specific listing response objects for web and mobile clients. A client requesting a listing detail view receives only the fields defined for that instrument type — not a generic record with 170 nullable columns.

WHAT THE SCHEMA ENGINE REPLACES

A platform that hard-codes each instrument type would require a separate database table, a separate form, a separate set of validation rules, a separate API endpoint, and a separate search configuration for every instrument type it supports. Across 150+ instrument types in 15 categories, that approach would require thousands of lines of redundant code, hundreds of database tables, and months of development time for each new instrument category added. The Schema Engine replaces all of that with a single configurable architecture. Adding a new instrument category is a database operation, not a development project.

SECTION 3 — SECURITY AND DATA ARCHITECTURE

Data Transmission

All data transmitted between any client — web browser, iOS application, or Android application — and the platform API layer is encrypted in transit using Transport Layer Security (TLS). No unencrypted data transmission occurs on any access point under any circumstance. HTTP requests are permanently redirected to HTTPS at the infrastructure level before reaching the application.

Authentication and Session Management

Participant credentials are authenticated against the database at the API layer. Passwords are processed through a cryptographic one-way hashing algorithm — bcrypt or Argon2 — before storage. Passwords are never stored in any readable form anywhere in the system. Session tokens are cryptographically signed JSON Web Tokens (JWTs) with defined expiration windows. Tokens are transmitted only over TLS and stored in secure storage on the client — the iOS Keychain for iOS clients, the Android Keystore for Android clients, and HttpOnly secure cookies for web clients.

Access Control

All permission evaluation occurs at the API layer before any database query executes. The permission framework evaluates the authenticated participant's role classification and subscription tier against the requested resource and operation. A participant cannot access data or execute operations above their tier entitlement regardless of what the client-side interface presents. Client-side permission hiding is a UX convenience, not a security mechanism. Server-side enforcement is the only enforcement that matters and the only enforcement the platform relies on.

Data Encryption at Rest

All personally identifiable information stored in the SQL Server database is encrypted at rest using Transparent Data Encryption (TDE) at the database engine level. This protects participant data in the event of physical storage media access. Due diligence documents stored in blob storage are encrypted at rest using storage-level encryption with platform-managed keys.

Audit Infrastructure

The platform maintains comprehensive, tamper-evident audit logs recording all significant events — authentication, listing activity, document access, message transmission, offer submission, bid placement, and administrative actions. Audit log records are written with timestamps and cannot be modified or deleted through normal application operations. The audit infrastructure supports the Internal Messaging System's immutability commitment and the Auction Engine's bid record integrity.

SECTION 4 — ONGOING MAINTENANCE PROFILE

The platform's technology choices were made with long-term maintainability as an explicit design constraint. The result is a platform that a competent .NET development team can maintain, extend, and operate without specialized knowledge of exotic frameworks, proprietary APIs, or unusual architecture patterns.

Routine maintenance activities include SQL Server patch management on the standard Microsoft update cycle, ASP.NET framework updates on the standard .NET release schedule, SSL certificate renewal on the standard annual cycle, and application-level bug fixes and feature additions as required. None of these activities require platform-specific expertise beyond standard .NET web development and SQL Server database administration.

The Schema Engine architecture specifically reduces ongoing maintenance cost by eliminating the code changes that would otherwise be required when adding new instrument types or modifying existing field sets. Instrument taxonomy changes — the most common platform evolution activity — are database operations performed through the admin interface, not code deployments requiring developer involvement.

Full technical architecture documentation, data model diagrams, API specification, deployment procedures, and administrator reference documentation are included in the platform delivery package under all three commercial arrangements.

Complete technical infrastructure detail is contained in the DCXchange.net Platform Technology White Paper, available separately upon request from licensing@dcxchange.net.

Standard Technology. Extraordinary Architecture. Built to Last and Easy to Own.