

Architecture Review: [Application Name]

Prepared for HedgeServ by [Architect Name]

*[Enter Date]*

# Analysis and Risk-Rate Level

The grading levels used in the detailed sections below use the following maturity model. Inversely they also set the priority level in need to address or track a particular implementation.

|  |  |
| --- | --- |
| Low | Technical implementation aligns to recommended best practices and appears to be a sound overall with little to no technical debt through implementation or approach. This is considered a mature & acceptable implementation. |
| Medium | Technical implementation approaches best practices but there are some key areas in need of improvement and/or will be tracked by architecture team. Minor technical debt. |
| High | Technical implementation is has critical areas in need of improvement. The issue identified may require major refactoring or significant re-architecture. This is major technical debt and should be addressed very soon. |
| Critical | Technical implementation is not recommended for production use and should be addressed as soon as possible. This is considered proof-of-concept, unproven or experimental. Significant technical debt or security violation. |
| N/A | Not applicable to product or application. |

# Executive Summary

*{Insert a summary of the platform in 250 words or less and include the team members responsible for platform}*

## Product Features

|  |  |
| --- | --- |
| Core Features | *Risk-Rating* |
| Customers/Consumers | *Risk-Rating* |
| Front End UI/UX Frameworks | *Risk-Rating* |
| Mobility/Responsive/Accessibility | *Risk-Rating* |
| Authentication/Authorization | *Risk-Rating* |
| Entitlements | *Risk-Rating* |

## Software Development Lifecycle

|  |  |
| --- | --- |
| Development Methodology | *Risk-Rating* |
| Deployment Methodology | *Risk-Rating* |
| Testing Methodology | *Risk-Rating* |
| Production Support | *Risk-Rating* |
| Code Analysis | *Risk-Rating* |

## Architecture

|  |  |
| --- | --- |
| New Technology | *Risk-Rating* |
| Hosting Facility | *Risk-Rating* |
| Disaster Recovery Plan | *Risk-Rating* |
| High Availability & Failover | *Risk-Rating* |
| Load Balancing | *Risk-Rating* |
| Routing | *Risk-Rating* |
| Performance | *Risk-Rating* |
| Scalability | *Risk-Rating* |
| Service Design | *Risk-Rating* |
| Application Security | *Risk-Rating* |
| Network Security | *Risk-Rating* |
| Data Security | *Risk-Rating* |
| Database | *Risk-Rating* |
| Caching | *Risk-Rating* |
| Logging | *Risk-Rating* |
| Analytics | *Risk-Rating* |
| 3rd Party Integrations (**internal** to product/platform) | *Risk-Rating* |
| 3rd Party Integrations (**external** to product/platform) | *Risk-Rating* |

# Criteria for Evaluation

## Product Focus

|  |  |
| --- | --- |
| Core Features | *Low* |
| An ALL NEW product that delivers a dynamically user configurable dashboards:   * + A market differentiating product in line with the expectations of being the technology leader in the hedge fund administration space   + Supporting the needs of all client portfolio types and user groups   + Supporting portfolio views, investor views and operational views   + ALL NEW capability to quickly respond to requests for new dashboards and views   + The framework will allow for any data set to be supported   + In demo scenarios, could quickly build preferred views based on prospect requirements   + Data sets from supported sources can be added without a release   + New data sets can be exposed in days or weeks (rather than expensive custom builds we do now) | |
| Customers/Consumers | *Low* |
| *Flexbase will be available to both internal Hedgeserv Users and external clients. This will number in the few thousands in aggregate.*  Usage will vary between internal and external users. As greater functionality is rolled out (dynamic queries/near real-time notifications) the system should be designed to scale. | |
| Front-end UI/UX Frameworks | *Medium* |
| The flexbase UI/UX will be built on HTML5, Angular 2.0 and D3.js.  ***Suggestion:***   * *The Tavant Infinity framework is being refactored in Angular 2.0. They have no production clients using their framework in Angular 2.0. This needs to be validated.* * *Furthermore once we are able to assess the code-base we can determine how extensible and re-usable the framework can be for future projects.* * *We must ensure that there is no tight binding/coupling to specific technologies (DBs, Logging, Caching, Protocols, Message formats, etc).* | |
| Mobility/Responsive/Accessibility Design | *Medium* |
| Flexbase is web-based user dashboard built to be responsive and run on all browsers in a PC, Laptop or tablet. Mobile is not expected to be supported currently. Browser versions should be validated across IE, FF, Safari on Windows and Mac.  ***Suggestion:***   * *We should be looking to create mobile standards if the use-case for our customer’s-customers use becomes necessary. Most users may want to digest this data on tablets and mobile form-factors.* * *Validating the quality of our application against real mobile devices could be achieved using AWS Device Farm.* | |
| Authentication/Authorization Method(s) | *low* |
| *Should cover definition of application entitlements and how well they conform to existing standards (compliance) or gap to emerging standards (risk).*  The Flexbase architecture will leverage a combination of webAuth (JSON Web Token – JWT) and ForgeRock integration for 2FA and authentication as well as application access. Fine grained entitlements is yet TBD and can reside in a separate FB architecture or centralized on ForgeRock (Directory Services LDAP DB).  ***Suggestion:*** *(insert text here).* | |
| Entitlements | *low* |
| *Should cover definition of application entitlements and how well they conform to existing standards (compliance) or gap to emerging standards (risk). In general tools should be migrating towards using WebAuth-FR (WAFR).*  Applications entitlements are well known and will align to existing levels of entitlements to data. Flexbase will persist user-level preferences related to the UI/UX settings in a separate SQL Server database.  ***Suggestion:*** *None.* | |

## Software Development Lifecycle (SDLC)

|  |  |
| --- | --- |
| Development Methodology | *low* |
| *Capture if the product team is following established best-practices for developing software. Delivery methodology (agile, extreme, waterfall, etc). Are they capturing and maintaining project boards (e.g. JIRA)? Daily stand-ups? Code reviews? Commenting pull-requests? Commenting code? Unit testing code, etc?*  The team will develop software using the Agile-Scrum methodology. All user stories will be managed and maintained in JIRA. Sprints will be 2-3 weeks and the team will conduct daily stand-ups.  Code will be source controlled using github.com. Code reviews will be conducted by tech lead for each pull request. Code will be commented and unit-tested.  ***Suggestion:*** *None.* | |
| Deployment Methodology | *low* |
| *Is the team following best-practices for achieving CI/CD with deploying and testing of software. Deployment and Resource managers – Docker, Mesos*  The goal of the project is to have complete automation of deployment of software changes from development to production using a combination of Chef and Jenkins. Automation of testing will be done to the fullest extent possible.  ***Suggestion:*** *None.* | |
| Change Control Methodology | *low* |
| The product will adhere to current HedgeServ change control methodology.  ***Suggestion:*** *None* | |
| Testing Methodology (Unit, Integration, Load) | *TBD* |
| *TBD*  ***Suggestion:*** *(insert text here).* | |
| Production Support Methodology | *TBD* |
| *How is the product supported once deployed to production (monitoring, run-books, escalation, service-level agreements, etc)*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Code Analysis | *TBD* |
| *Capture location of source code in which repository (e.g. SVN or Git)? What best-practices should be observed by reviewer? Architect should schedule time to sit-in on a code review. Tools like Sonar with findbugs for Static Analysis. TestNG like tools for Dynamic Analysis. Also security code analysis with 3pp tools like BlackDuck for 3pp exposure and mitigation.*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |

## Architecture

The following diagram depicts the high-level diagram of the **[Product Name]** Architecture.

**[Insert Architecture Logical Design Here]**

|  |  |
| --- | --- |
| Architecture Logical Design | *Medium* |
| Flexbase is a configurable dashboards that leverage ForgeRock to provide user authentication and authorization to Flexbase. Once authenticated & authorized the user is issued a JSON-Web Token (JWT) with a pre-defined time-to-live (ttl). The user is then routed via NGINX+ to the web server hosting the dashboard/widgets. The flexbase API is called to retrieve persisted user custom configurations (templates, widget configuration, widget types, user preferences, etc). The system logs activities throughout to the ELK stack (logins, identified actions, feature usage, etc). Once the widgets and dashboard is loaded the widgets call the GraphQL Data API with (JWT) to request data they are authorized to view. GQL uses this to understand which subset of data a user has access to and makes a request to 1 of 3 data sources;   1. An OLAP data repository (e.g. Hadoop, Grid DB) 2. Dynamic Data Service (DDS) 3. Cube Services   The data is then returned to the widget and rendered.  (Future Capability – Timeframe TBD)  To fulfill the need for near real-time notification an agent installed on each client HAS will listen for updates as they occur within the underlying data stores posted messages via JMS (e.g. SonicMQ) and publishes them via AMQP (e.g. RabbitMQ) at which point the centralized subscribed multi-tenant Event Aggregation Listener (AGL) component will bundle all client updates (with entitlements provided) and publish that information to the Update Relay. The UR manages the UX socket connections and pushes updates to the correct browsers.  ***Suggestion:*** *The overall architecture design introduces some new logical designs that will be a first for Hedgeserv, namely: Infinity Framework (on Angualar 2) and GraphQL. This architecture will need to be validated that it will work as intended. In future phases need to ensure the near-realtime components and socket management is secure and scalable.* | |
| New Technology | *Risk-Rating* |
| The first phase of the architecture deployed will not introduce any new technology.  ***Suggestion:*** *Future phases of the project will likely introduce a few new technologies currently not supported in production (e.g. RabbitMQ, HazelCast caching, etc) still TBD.* | |
| Hosting Provider | *low* |
| *Where is the application/product hosted? Any description of hosting facility that is worth mentioning should be done here (SOX1, 2 certification, etc)*  The platform will be hosted in our current production data-center.  ***Suggestion:*** *None.* | |
| Disaster Recovery | *medium* |
| *Is there a disaster recovery plan in place? What is it? Does it conform to established best-practices?*  DR will be in AWS.  ***Suggestion:*** *None.* | |
| High Availability/Failover/Business Continuity | *Risk-Rating* |
| *Is the solution architected to provide adequate high-availability (e.g. is redundancy built in)?*  The solution architecture is designed with high availability in mind. The solution will run at a minimum of 2 instances of services across the stack. Technologies chosen (e.g. Tomcat) are mature technologies. With redundancy built in any single service failure will be logged and traffic can be supported by remaining services with the ability to failover and scale out horizontally.  ***Suggestion:*** *HA/Failover should be tested under normal and heavy load.* | |
| Load Balancing (ELB) | *Risk-Rating* |
| *Is the solution take adequate balancing where scale-out is required or expected?*  NGINX+ has been chosen as the network load-balancer.  ***Suggestion:*** *None.* | |
| Application Routing (ALB) | *low* |
| *Are there unique routing logic necessary for this application/product anywhere in the architecture?*  There is no distinct routing requirements outside of the normal logical flows of data requests to the correct HAS servers which will be managed by the NGINX+ servers.  ***Suggestion:*** *None.* | |
| Performance | *Risk-Rating* |
| *Are there unique/specific performance requirements for this product/application? Has the product been measured (or plans in place to measure) for performance under anticipated load?*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Service Design (should be part of code analysis) | *Risk-Rating* |
| *If the architecture uses any REST/SOAP based endpoints are they well documented and understood? Are they designed to scale and be re-used beyond the product/application requirements?*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Application Security (OWASP, Security Scan results | *Risk-Rating* |
| *Are applications routinely scanned for vulnerabilities during SDLC (Static/Dynamic testing)? Explicit OWASP top 10 mitigations?*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Network Security (VPC, NACL, Security Groups) | *Risk-Rating* |
| *Should address what cyber defenses are in place for the network (IPS/IDS, DDOS suppression, Firewalls, Packet inspection, etc)? External communications (data egress or requests) should be over secure channels.*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Data Security (PII, PCI, FERPA, HIPAA) | *Risk-Rating* |
| *Should include data access controls, audits? Are data loss prevention systems in place? Backups, encryption (at row, table or db level)? How are encryption keys managed?*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Database | *Risk-Rating* |
| (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Caching | *Risk-Rating* |
| *Should capture all the data caching points from front-end down to DB and include the types of data being cached and level of sensitivity of data and any security precautions taken in caching the data (e.g. is cached data encrypted – symmetric/assymetric)?*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Logging | *Risk-Rating* |
| *Should capture any logging standards and how detailed/effective are the logs in providing RCA? When are logs reviewed? How are they used? How long are logs kept and are they backed up?*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| Analytics (Big Data) | *Risk-Rating* |
| *Are basic analytic data points been defined for this platform? (# of logins, time on platform, features used, etc)*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| 3rd Party Integrations (internal) | *Risk-Rating* |
| *Capture any 3rd party plug-ins or tools (hosted privately or publicly but installed and monitored by us) used to fulfill the architecture and why they were chosen? Licensed/Open-Sourced? Supported and is a support agreement in place?*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |
| 3rd Party Integrations (external) | *Risk-Rating* |
| *Capture any 3rd party plug-ins or tools (hosted by a 3rd party provider (e.g. Okta)) used to fulfill the architecture and why they were chosen? Licensed/Open-Sourced? Supported and is a support agreement in place?*  (Insert descriptive text here)  ***Suggestion:*** *(insert text here).* | |