Sept 19th, 2025

Architecture Options

OptionTrax Migration Project

Globant | EMPOWER

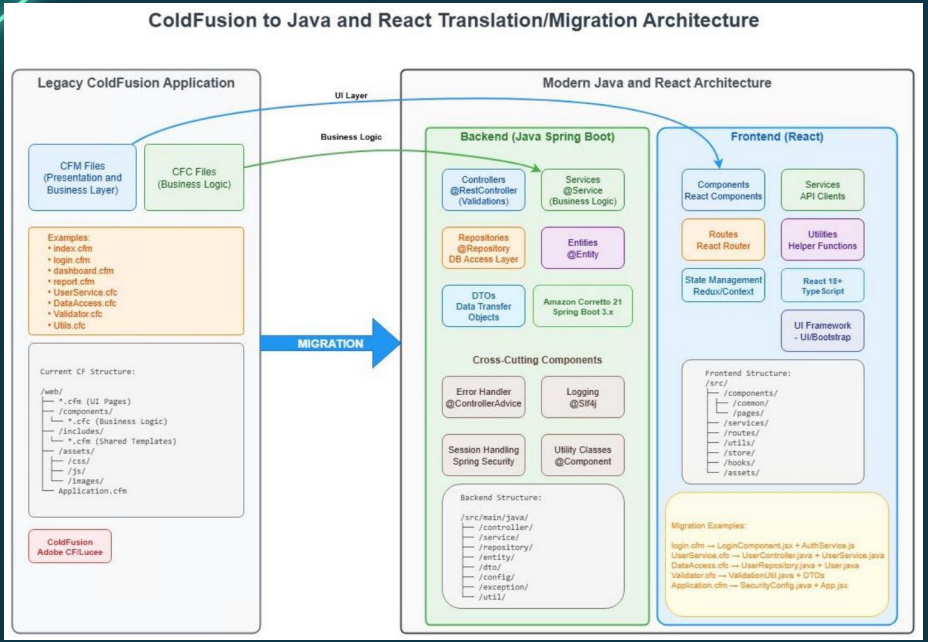
Agenda

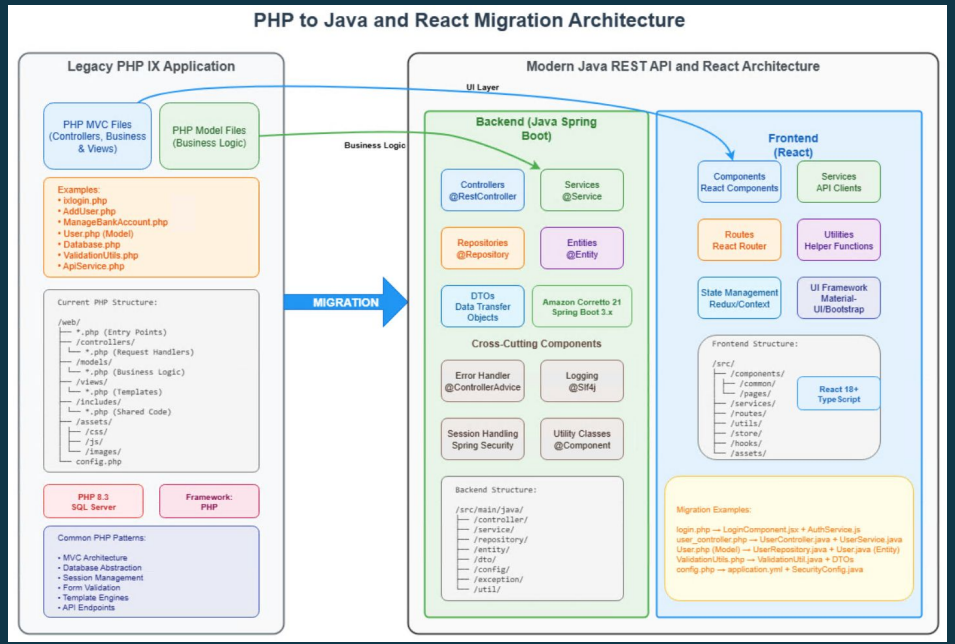
1. Architectural Options
    - AS-IS 1:1 Translation
    - 4-6 Modules Services (4 OT & 2 ITX)
    - Services by Menu
    - Microservices
2. Comparative
    - Options comparison
    - Comparative risk matrix
3. Impact
4. Appendix

1 Architectural Options

OptionTrax Migration Project

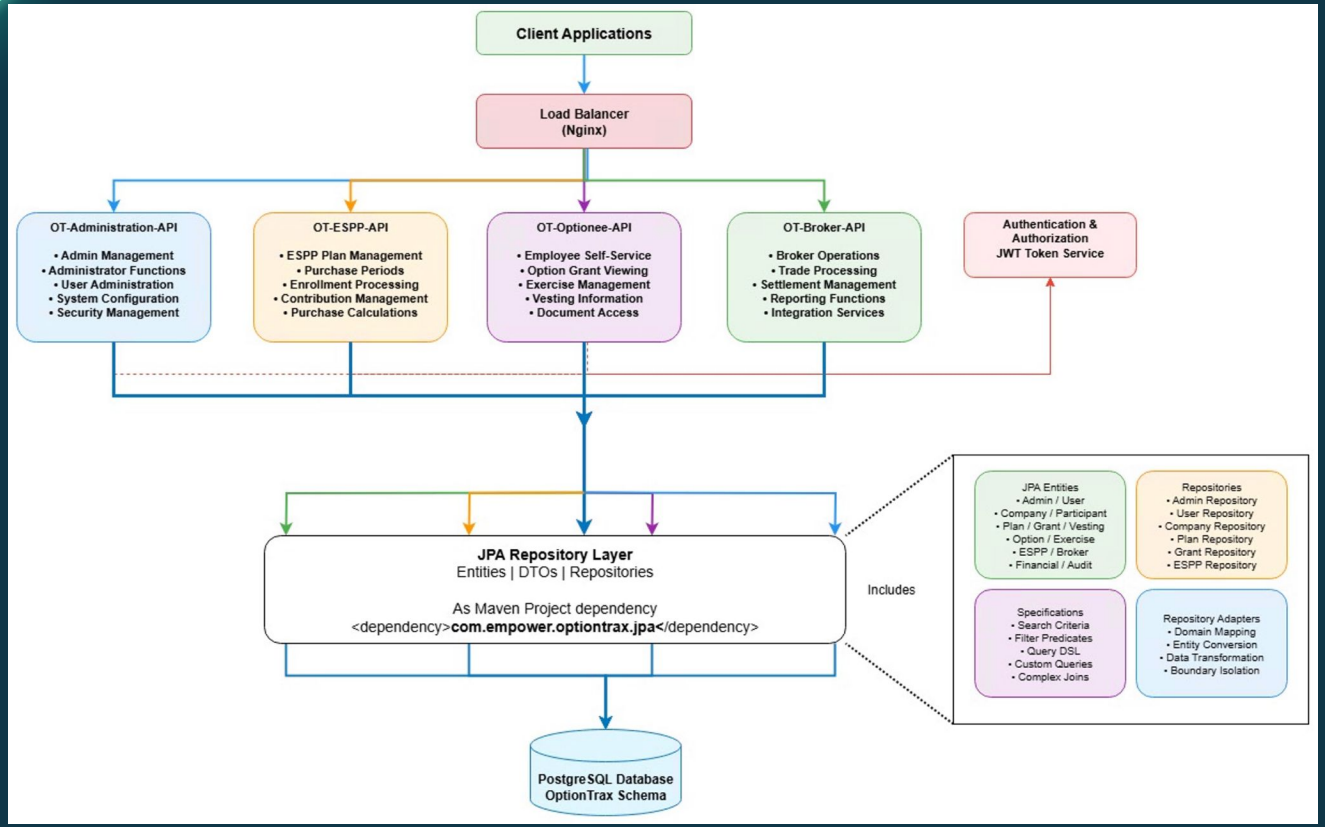
Architectural Options - AS-IS 1:1 Translation





- A direct 1:1 translation was the agreed approach to reduce complexity, improve efficiency, allowing us to dedicate the efforts into code extraction and translation to ensure feature parity between both legacy and new applications.
- The legacy application implements a monolithic architecture.

Architectural Options - 4-6 Module Services



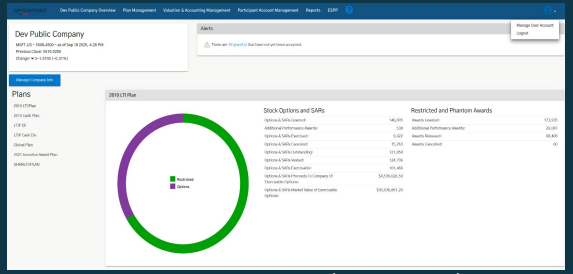
- **6 Module API/Rest Services** (4 for OptionTrax, 2 for InvestmenTrax).
- JPA Common DB project dependency including:
  - Entities
  - DTOs
  - Repositories
  - Mappers

OptionTrax Migration Project

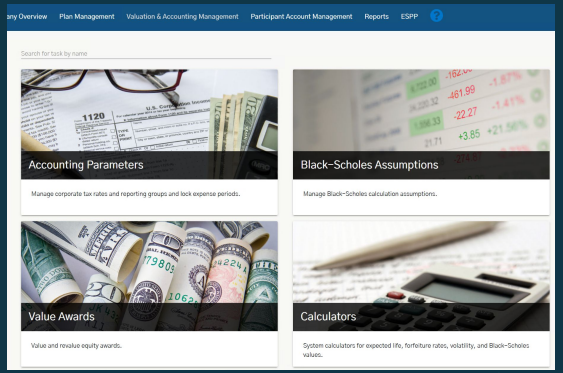
Architectural Options - Services by Menu

One new monolithic service per top-level + first child menu for each application (OT + IT)

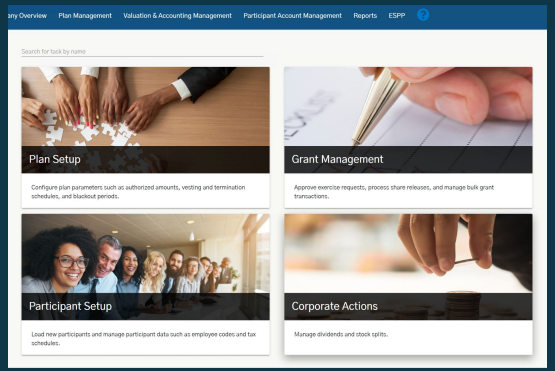
- **28 API/REST services**, each with the corresponding repository, Spring Boot application, deployment pipeline, and infrastructure.
- Overhead to analyze menus and tables involved in each service.



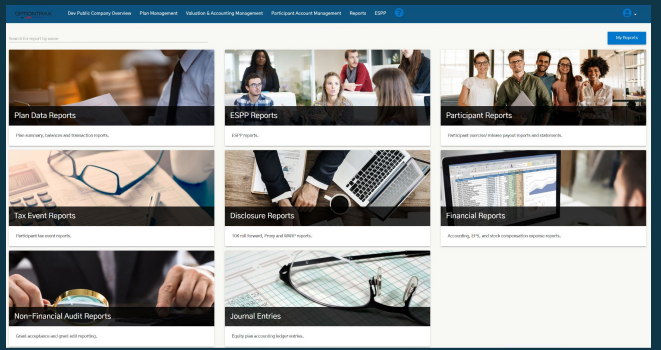
Company Management (1 service)



Valuation & Account Management (4 services)

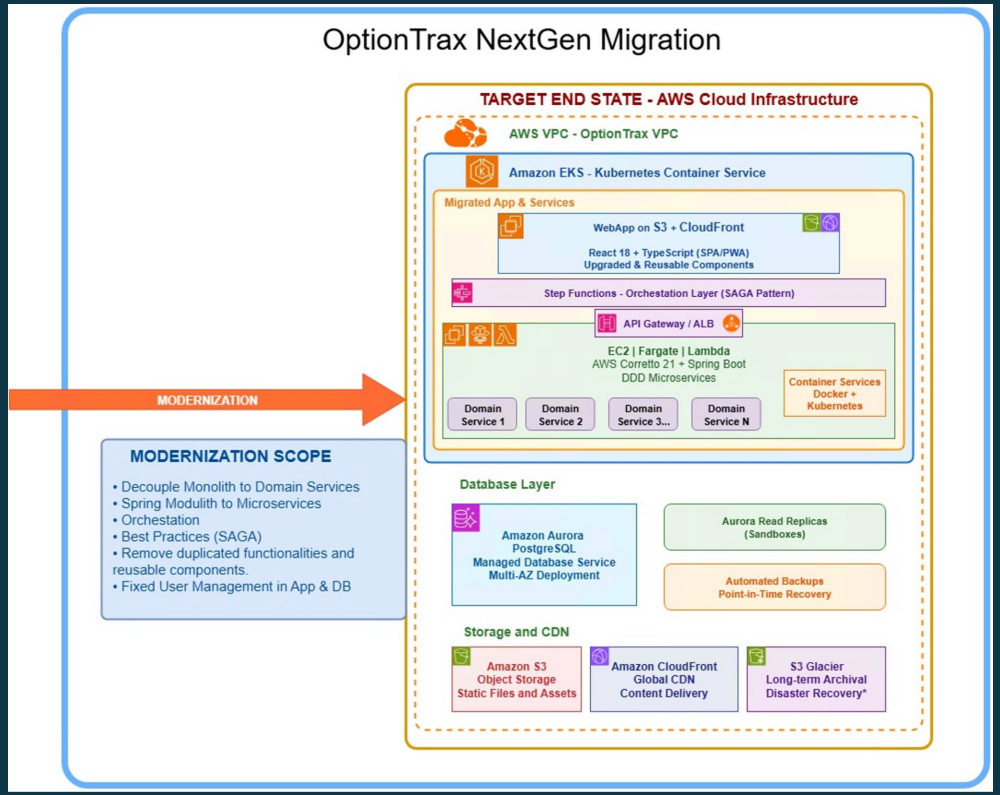


Plan Management (4 services)



Reports Management (8 services)

Architectural Options - Microservices



- **Unknown number of services**

Implementation guidelines

- Domain service approach
- Orchestration
- Each service to have its own database
- Required deep analysis to define al domain services and its operations.

2 Comparative

Options comparison

Architectural Approach	AS-IS 1:1 Translation Monolith	API per Module (Modular Monolith)	API per Menu (UI-Driven Decomposition)	Microservices
Cost	Low initial cost; manageable long-term maintenance cost.	Low initial cost; manageable long-term cost.	Costs are phased and spread out over time.	High initial investment; can be cost-effective at extreme scale.
Complexity	Simpler development, testing, maintenance and troubleshooting.	Simpler development, testing, maintenance and troubleshooting Adds lower complexity for the migration	Moderate complexity; requires a revision of the database access layer. Service count is higher	High development, testing, and management complexity. Largest number of services
Scalability & Resilience	Vertical scaling only; a single point of failure can bring down the entire system. Using Modulith would prepare for an enhancement in the next iteration	Vertical scaling only; single point of failure. Using Modulith would prepare for an enhancement in the future	Incremental scalability as new application sections are built.	Superior horizontal and granular scaling; isolated failures.
Data Management	Single, shared database; simple ACID transactions.	Single, shared database; simple ACID transactions.	Single, shared database; simple ACID transactions.	Dedicated private database per service; complex distributed transactions. Eventual data consistency.
Team Organization	Centralized, small to medium-sized teams.	Centralized, small to medium-sized teams.	Can be adopted by teams of any size; teams can work on different components.	Decentralized, autonomous, cross-functional teams.
Deployment Velocity	Slower, single-unit deployments.	Slower, single-unit deployments.	Continuous, low-risk deployments of new components.	High; each service can be deployed independently.

OptionTrax Migration Project

Comparative Risk Matrix

Decision tree

Is this a new project?
├── Yes → Start with Monolithic
│ └── Team > 2-4 teams → Modular Monolithic
└── No → Existing system?
├── Monolithic with scaling issues?
│ ├── Clear domain boundaries? → Modular Monolithic
│ └── Complex domains + large teams? → Microservices
└── Already Modular Monolithic?
└── Independent scaling needed? → Microservices

Risk Factor	Monolithic	Modular Monolithic	Microservices
Over-engineering	Low	Medium	**High**
Technical Debt	**High**	Medium	Low
Deployment Failures	Medium	Medium	**High**
Performance Issues	Low	Low	**High**
Team Coordination	Low	Medium	**High**
Skill Gap	Low	Medium	**High**
Dependencies	Medium	Medium	**High**

3 Impact

Options Impact

Architectural Approach	AS-IS 1:1 Translation Monolith	API per Module (Modular Monolith)	API per Menu (UI-Driven Decomposition)	Microservices
Time Impact	Delayed 1 week due to backend services pending Empower architecture approval to be deployed and tested.	Delayed 1 week due to backend services pending Empower architecture approval to be deployed and tested. **+ 1 Week** to make a more detailed analysis for the	Delayed 1 week due to backend services pending Empower architecture approval to be deployed and tested. **+ 2-3 Weeks** for analysis and implementation of approximately XX UI-Driven Services.	Delayed 1 week due to backend services pending Empower architecture approval to be deployed and tested. **+ 2-3 Months** for analysis and implementation of Domain Services, Transactions, Orchestration, Tables involved by services.
Activities	- Continue with current plan timeline - Continue getting the Development Cycle with the first endpoints deployed in DEV.	- Create the 4-6 repos/services - Reorganize Services, packaging by Business Modules - Continue getting the Development Cycle with the first endpoints deployed in DEV.	- Analyze in deep how many services will be created by Menu. - Analyze the DB Tables involved in each service. - Create N repos/services from de previous analysis. - Continue getting the Development Cycle with the first endpoints deployed in DEV.	- Conduct an in-depth analysis of how many domain services will be created. - Analyze the database tables involved in each domain service. - Analyze complex flows that require transactions from more than one database entity. - Create N repositories/services based on the above analysis. - Continue the development cycle with the first endpoints implemented in DEV.

4 Appendix.

OptionTrax Migration Project

Project Scope by SOW

Scope of Work. Functional Requirements:

Scope of Work. Functional Requirements:

1. Database Migration

- Re-architect the DB layer by implementing a multi-tenant strategy
  - Logical data separation design: Each tenant's data must be isolated and inaccessible to other tenants. Achieved through schema separation, or shared database with tenant-specific views.
- Migrate current data from approximately 450 databases (not to exceed 500 databases) into the new architecture. Around 150 are sandbox databases that will be evaluated for migration or discarded.
- Migrate database type from Microsoft SQL Server to AWS Aurora Postgres

2. Application Code Migration: The project will focus on translating ColdFusion and PHP code to Java and React.

- ColdFusion: Primary module for equity plan administration
  - Approximately 2,297 files across 4 external Portals (not to exceed 2,600 files).
  - Used by Plan Administrators, Stock Shareholders and Equity Participants.
- PHP: Cap Table modules (share issuance, transfer and configure distributions, equity subscription, enrollments & redemption events)
  - Approximately 510 files
- The entire tool currently runs on just 2 Windows Servers. Estimated total:
  - 450 UI screens
  - 230 reports
  - 12 batch jobs
  - 450 databases (each client has their own database, includes around 150 sandbox databases, table structures are the same across databases)
  - 280 of the databases are prod databases
- Leverage GenAI in the translation of existing code to the target codebase
  - GitHub Copilot will be the primary tool used for code translation.
  - The use of GenAI is limited to generating code snippets and assisting in refactoring; manual review will be required for accuracy.

Out of Scope

Out of Scope:

- Implementing Empower Monitoring Standards
- Any ADA or WCAG Compliance
- Enhancements or new features that are not mutually agreed upon during the Assessment Phase for inclusion in the implementation roadmap.
- Optimizations and refactoring of code other than arising due to database consolidation that is a deviation from the baseline in ColdFusion to be migrated to Java & React
- Language reviews & testing of language translations for upper environments (Dev, UAT, Prod)
- Full-end to-end DevOps pipeline; Globant will require Empower FTE to partner on the implementation due to credentials restrictions on vendor accounts
- Issues arising from third-party integrations or services that are outside Globant's scope or control

Application Code Migration:

1. Coldfusion Accelerator may be installed in the environment and utilized.
2. Empower will provide Github Copilot and it may be the AI-powered digital assistant to be used because it is the current Empower standard.
3. Quality of AI Code Translations will be iterative and ensuring accuracy is paramount and priority over speed.
4. The application will be replicated according to the legacy initial snapshot. A separate environment will be provided by Empower at the project's outset to serve as a basis for comparison.
5. Code modifications and changes to the baseline will not take place during the migration phase.
6. During the Execution phase, only migration efforts are considered, excluding code cleanup, improvements, etc.
7. The applications will be developed and tested on desktop web as per the original legacy applications, without any design changes or adaptations for new devices or resolutions.

Options comparison

Sources

https://www.ibm.com/think/topics/monolithic-architecture

https://aws.amazon.com/es/blogs/apn/migrating-applications-from-monolithic-to-microservice-on-aws/

https://developer.android.com/topic/modularization/patterns

https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

https://cloud.google.com/learn/what-is-microservices-architecture

https://developers.redhat.com/articles/2022/01/11/5-design-principles-microservices

https://learn.microsoft.com/en-us/azure/architecture/microservices/design/data-considerations

https://learn.microsoft.com/en-us/azure/architecture/example-scenario/devops/automated-api-deployments-apiops

https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

https://cloud.google.com/kubernetes-engine/docs/learn/cymbal-books/lp1/modular