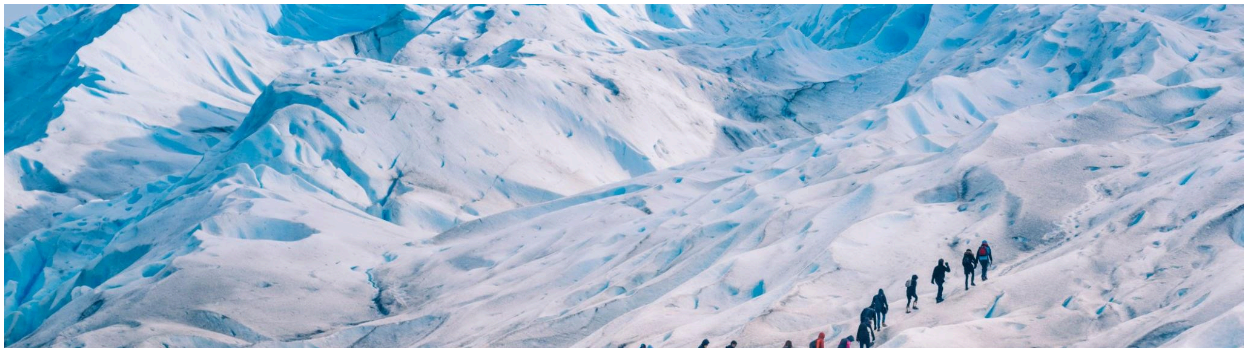

Enterprise Legacy Modernization Project

Introduction

When you enter the professional software development world, you won't be building applications from scratch on greenfield projects. Instead, you'll inherit massive, complex legacy codebases that have been "duct taped together" over decades—systems with millions of lines of code written in older versions of Java, COBOL, Python, or C that somehow keep entire businesses running.

These enterprise systems represent the reality of software development: proven business logic wrapped in outdated technology stacks, sprawling architectures that evolved organically, and critical functionality buried in code that nobody fully understands anymore. Your ability to quickly comprehend these systems and modernize them effectively will determine your career trajectory.

This project simulates that exact scenario. You'll take a real open-source codebase with over 1,000,000 lines of legacy code and use AI-assisted development tools—like Cursor and Claude Code—to not just understand it, but to completely modernize and relaunch it for a new target market.

The skills you develop here translate directly to enterprise value: companies desperately need developers who can leverage AI to breathe new life into their legacy systems while preserving the valuable business logic that took years to perfect. This isn't about replacing legacy systems—it's about evolving them.

Core Challenge

Select a large-scale legacy open-source codebase (1M+ lines), identify a specific target user segment that could benefit from a modernized version, then use AI-assisted development to create a completely relaunch-ready application that preserves the core business logic while delivering a modern user experience and architecture.

Success means: Shipping a working, modernized application that demonstrates both deep understanding of the legacy system and the ability to transform complex codebases using AI assistance.

Project Pathways

Choose one pathway based on the type of legacy modernization challenge you want to tackle:

Path 1: Enterprise CRM Modernization

Legacy Codebase: SugarCRM CE (2.3M lines, PHP 7.4) or SuiteCRM (1.8M lines, PHP)
Why These Systems: Contain sophisticated customer relationship management business logic with complex workflows, custom field systems, and integration patterns that took years to refine

Modernization Opportunities:

- Mobile-first architecture for field sales teams
- Industry-specific customizations (healthcare, real estate, manufacturing)
- Modern API-first design with microservices architecture
- AI-powered lead scoring and customer insights

Path 2: Office Suite Platform Transformation

Legacy Codebase: Apache OpenOffice (8M+ lines, C++) or LibreOffice (10M+ lines, C++)
Why These Systems: Represent decades of document processing expertise with complex file format handling, cross-platform compatibility, and feature-rich editing capabilities

Modernization Opportunities:

- Cloud-native collaborative editing platform
- Mobile document editing for specific professional workflows
- Industry-specific document templates and automation
- Integration with modern cloud storage and workflow systems

Path 3: Legacy Python Web Platform Evolution

Legacy Codebase: Plone CMS (1.1M lines, Python 2.7/3.6) or legacy Django apps (Django 1.x/2.x)
Why These Systems: Contain years of web application patterns, user management systems, content workflows, and integration logic that represent typical enterprise Python deployments

Modernization Opportunities:

- Python 3.11+ migration with async/await patterns
- Modern container deployment with Docker/Kubernetes
- API-first architecture replacing monolithic structure
- Modern frontend frameworks replacing server-side templates

Path 4: Deprecated Python Framework Revival

Legacy Codebase: Pylons-based applications, TurboGears 1.x projects, or Zope 2.x applications

Why These Systems: Representative of enterprise Python applications built 10-15 years ago that companies still depend on but struggle to maintain

Modernization Opportunities:

- Framework migration to modern Python (Flask/FastAPI/Django)
- Database ORM modernization (SQLAlchemy 2.x patterns)
- Authentication system updates for modern security standards
- Cloud-native deployment replacing legacy server configurations

Target User Selection

After choosing your pathway, identify a specific target user segment that represents a modernization opportunity:

Examples of Target Users:

1. **SugarCRM → Real Estate CRM**: Agents who need mobile-first property management with modern UX
2. **OpenOffice → Educational Document Platform**: Teachers needing collaborative lesson planning with cloud integration
3. **Trac → Startup Project Management**: Small tech teams wanting simple, fast issue tracking without enterprise complexity
4. **Alfresco → Small Business Workflow**: SMBs needing document approval workflows without enterprise overhead

Target User Requirements:

- Must represent a legitimate market opportunity that could benefit from the legacy system's core functionality
- Should highlight specific pain points that modern technology can solve

- Needs to be narrow enough to complete in 7 days but broad enough to demonstrate meaningful modernization

Grading Criteria

Your project will be evaluated on the following criteria, each weighted equally:

1. Legacy System Understanding (20 points)

Demonstrates deep comprehension of the original codebase

- **Excellent (18-20)**: Complete architecture mapping, identifies all critical business logic, demonstrates understanding of data flows and integration points
- **Good (14-17)**: Solid understanding of core functionality, identifies most business rules, maps primary system components
- **Satisfactory (10-13)**: Basic understanding of system structure, identifies main features but misses some complexity
- **Needs Improvement (0-9)**: Superficial understanding, fails to grasp core business logic or system architecture

2. Six New Features Implementation (50 points - 10 points each)

Must add exactly 6 meaningful features that enhance the original application

- **Feature Requirements**: Each feature must be functional, add genuine business value, and integrate properly with existing code
- **Examples of Qualifying Features**:
  - Modern authentication (OAuth, 2FA, SSO)
  - Real-time notifications/messaging
  - Advanced search and filtering
  - Mobile-responsive design improvements
  - API endpoints for external integration
  - Dashboard/analytics functionality
  - Automated workflows or business rules
  - File upload/management systems
  - User role management enhancements
  - Integration with third-party services

Scoring per Feature (5 points each):

- **Excellent (10)**: Feature is fully functional, well-integrated, adds significant business value
- **Good (8)**: Feature works properly, integrates well, provides clear value

- **Satisfactory (6)**: Feature functions but may have minor integration issues or limited value
- **Needs Improvement (2-4)**: Feature partially works or poorly integrated
- **Missing (0)**: Feature not implemented or non-functional

3. Technical Implementation Quality (20 points)

Code quality, architecture decisions, and technical execution

- **Excellent (17-20)**: Clean, well-structured code following best practices, proper error handling, efficient database queries, good security practices
- **Good (12-16)**: Generally well-written code with minor issues, adequate error handling, reasonable performance
- **Satisfactory (8-11)**: Functional code with some technical debt, basic error handling, acceptable performance
- **Needs Improvement (4-7)**: Poorly structured code, minimal error handling, performance issues
- **Unacceptable (0-3)**: Broken code, security vulnerabilities, major architectural problems

4. AI Utilization Documentation (10 points)

Evidence of effective AI-assisted development throughout the project

- **Excellent (9-10)**: Comprehensive documentation of AI prompts and techniques, innovative use of AI tools, clear methodology for AI-assisted learning and development
- **Good (7-8)**: Good documentation of AI usage, effective prompts, clear integration of AI in development process
- **Satisfactory (5-6)**: Basic documentation of AI assistance, some evidence of AI tool usage
- **Needs Improvement (2-4)**: Minimal documentation, unclear AI utilization
- **Missing (0-1)**: No evidence of AI assistance or documentation

7-Day Project Timeline

Days 1-2: Legacy System Mastery

- **Select codebase and analyze architecture** using AI-assisted code exploration
- **Identify target user segment** and define their specific needs/pain points
- **Map core business logic** that must be preserved in modernization
- **Set up development environment** and reproduce current functionality

Days 3-4: Modernization Design & Foundation

- **Design modern architecture** that preserves business logic while improving performance/UX
- **Begin core implementation** focusing on most critical user workflows
- **Implement modern deployment pipeline** (containerization, CI/CD, cloud deployment)
- **Create user interface foundation** with modern design patterns

Days 5-6: Feature Implementation & Integration

- **Complete core feature modernization** ensuring all essential functionality works
- **Implement target user-specific enhancements** that differentiate from legacy version
- **Add modern integrations** (authentication, third-party services, APIs)
- **Performance optimization and testing** to achieve quantified improvement goals

Day 7: Polish & Launch Preparation

- **Final testing and bug fixes** ensuring production-ready quality
- **Create deployment documentation** and user migration guides
- **Prepare demonstration** showing before/after comparisons and quantified improvements
- **Document modernization methodology** and AI-assisted development approach

Final Thoughts

This project bridges the gap between academic learning and professional reality. In the enterprise world, you'll rarely build from scratch—instead, you'll modernize proven systems for new opportunities. The combination of legacy system comprehension and AI-assisted modernization represents the most valuable skill set for modern developers.

Success here demonstrates that you can take any complex legacy system, understand its business value, and transform it into something modern and competitive. This ability to leverage AI for legacy modernization is exactly what enterprises need most, making this project directly relevant to your career growth and immediate professional value.

Remember: the goal isn't to judge or criticize legacy code—it's to recognize the business value embedded in these systems and use modern tools to unlock that value for new users and use cases.

YOU CAN DO THIS! - JUST SUBMIT 💕