Heuristic Evaluation Report

Card Memo - Flashcard Learning Application

Repository: https://github.com/Kreetrix/Ohjelmistotuotanto-1

Evaluation Date: December 2, 2025

Evaluation Type: Software Engineering Heuristic Analysis

Technology Stack: Java 17+, JavaFX, MariaDB, Maven, Jenkins, Docker

Executive Summary

Card Memo is a flashcard-based learning application developed by a team of four developers using Java, JavaFX, and MariaDB. The application demonstrates several strong architectural decisions, particularly in its internationalization strategy and database design. However, critical security considerations and performance optimization strategies require immediate attention before production deployment.

1. Architecture & Design Patterns

Strengths

- Clear separation of concerns with distinct Backend (Java/Maven), Frontend (JavaFX), and Database (MariaDB) layers
- Database-driven localization system using normalized translation tables rather than bloated columns
- Internationalization support through ResourceBundle pattern with dedicated I18n utility class

Concerns

- No mention of architectural pattern (MVC, MVVM, Clean Architecture) - this could lead to coupling issues
- Lack of information about API layer or service architecture
- The LanguageController directly managing UI suggests potential tight coupling between presentation and business logic

2. Database Design

Strengths

- Normalized translation system using separate tables (card_translations, deck_translations, languages)
- Scalable approach - adding new languages doesn't require schema changes
- Proper foreign key relationships implied by the structure
- Timestamps for tracking updates (updated_at)

Database Schema

Table	Purpose	Key Fields

languages	Stores supported languages	code, name, native_name
card_translations	Card content in multiple languages	card_id, language_code, front_text, back_text
deck_translations	Deck metadata in multiple languages	deck_id, language_code, deck_name, description

Concerns

- No mention of database migration strategy
- Missing information about indexing strategy for performance
- No discussion of connection pooling or database transaction management
- The translation fallback mechanism isn't clearly defined in the schema

3. Code Quality & Testing

Strengths

- Unit testing with JUnit 5
- Integration testing mentioned
- UI testing with TestFX
- Code coverage tracking with Jacoco
- CI/CD pipeline with Jenkins

Concerns

- No specific test coverage targets mentioned
- Missing information about testing strategy for internationalization features
- No mention of mocking frameworks for database testing
- Test organization structure not documented

4. DevOps & Deployment

Strengths

- Docker containerization for consistent deployment
- Jenkins CI/CD automation
- Backup and staging environments
- Version control with GitHub

Concerns

- No mention of deployment strategy (blue-green, rolling, canary)
- Database migration automation not discussed
- No information about monitoring, logging, or error tracking
- Rollback strategy not mentioned

5. Internationalization (i18n) Implementation

Strengths

- Runtime language switching capability
- Clean separation of translation resources
- Support for multiple languages (English, Russian, Japanese)
- Dual-system approach: UI translations (ResourceBundle) + data translations (database)

Concerns

- Potential performance issues loading translations from database on every request
- No caching strategy mentioned for database translations

- Risk of inconsistent translations between UI and data layers
- No mention of translation validation or completeness checking

6. Security Considerations

Critical Gaps

- No mention of authentication/authorization implementation despite "login and registration support"
- Database credentials management not discussed
- No information about password hashing strategy
- API security not mentioned (if REST endpoints exist)
- SQL injection prevention not documented
- No discussion of session management

7. Error Handling & Resilience

Missing Information

- No error handling strategy documented
- Database connection failure handling unclear
- Translation fallback behavior needs clarification
- No mention of logging framework or strategy

8. Performance & Scalability

Concerns

- No caching layer mentioned despite database-heavy translation system
- No discussion of query optimization
- Connection pooling strategy not documented
- No mention of pagination for large deck/card collections
- JavaFX performance considerations for large datasets not addressed

9. Code Maintainability

Positive Indicators

- Clear documentation of language support
- Team collaboration evident (4 developers: Axel Nokireki, Vladimir Karpenko, Georgii Afanasev, Patrik Skogberg)
- Agile sprint methodology

Concerns

- No coding standards or style guide mentioned
- Documentation strategy unclear
- API documentation not mentioned
- No information about code review process

10. Dependencies & Technical Debt

Risks

- Java 17+ requirement - compatibility considerations
- MariaDB-specific features may limit database portability
- JavaFX has limited community support compared to web frameworks
- "Training for new tools (Jenkins)" suggests learning curve overhead

Priority Recommendations

High Priority

1. Document and implement security measures (authentication, password hashing, SQL injection prevention)
2. Implement caching for database translations to avoid performance bottlenecks
3. Define and document architectural pattern (suggest MVVM for JavaFX)
4. Create database migration strategy and scripts
5. Add comprehensive error handling and logging framework

Medium Priority

6. Define test coverage targets and improve testing documentation
7. Implement connection pooling for database
8. Add monitoring and alerting for production environment
9. Document API contracts if REST endpoints exist
10. Create coding standards and establish code review process

Low Priority

11. Consider pagination for large datasets
12. Optimize database queries with proper indexing
13. Implement automated translation validation
14. Add deployment rollback procedures

Conclusion

Card Memo demonstrates solid foundational architecture with a particularly well-designed internationalization and database translation system. The development team has adopted modern DevOps practices with CI/CD pipelines and automated testing. However, before the application can be considered production-ready, critical security measures must be implemented, and performance optimization strategies should be established to handle the database-intensive translation lookups.

The team should prioritize addressing the security gaps and implementing caching mechanisms. With these improvements, the application has strong potential to serve as an effective learning tool for students and language learners.

Heuristic Evaluation Report | Card Memo Application

Generated on December 2, 2025