Analysis and Improvement of a *Dynamic Web Project's Dependability*

Presented by:

- *Dominykas Kruminis*
- *Stefano Nicolò Zito*
- *Alisher Khaireden*

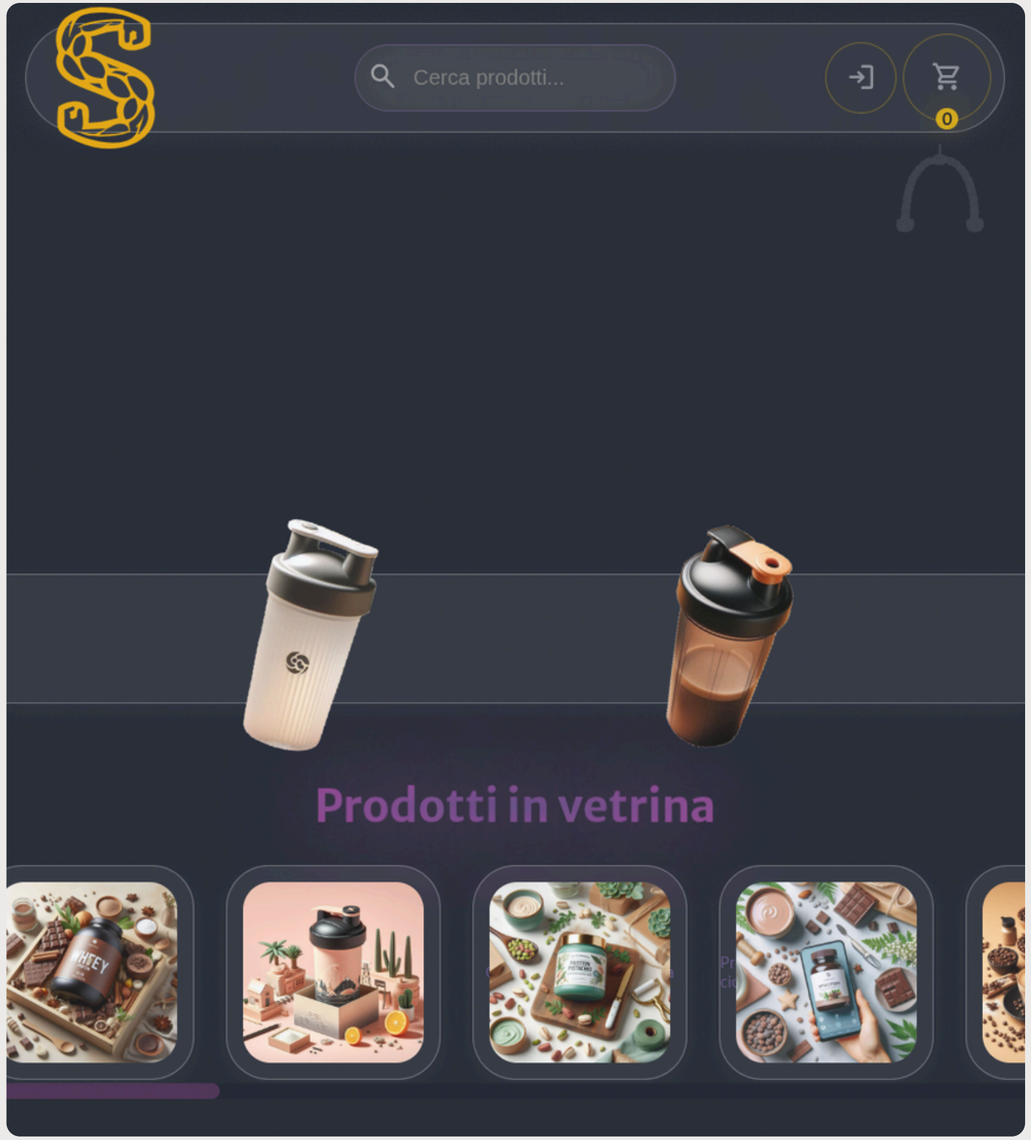
UNIVERSITÀ DI SALERNO, DIPARTIMENTO DI INFORMATICA

JANUARY, 2026

Introduction: *Shapedibles*

- **Pattern:** Classic Servlet-based architecture.
- **Logic:** Java Servlets handle HTTP/HTTPS requests and dynamically generate HTML responses to build the UI.
- **Data Tier:** Uses a relational SQLite database.
- **Data Access:** Implements DAO pattern to encapsulate CRUD operations, using JavaBeans for data transfer.

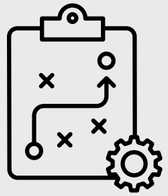
- **CI:** For automated build and integration cycles.
- **Formal Verification:** Using JML to ensure code correctness.
- **Deployment:** Containerized via Docker.
- **Testing & Analysis:** Includes automated testing, performance benchmarking, and comprehensive security assessments.



https://github.com/stefzito8/SwD-Shapedibles

Formal Verification Strategy

Ensuring architectural integrity by verifying the ‘bottom’ layer of the application.

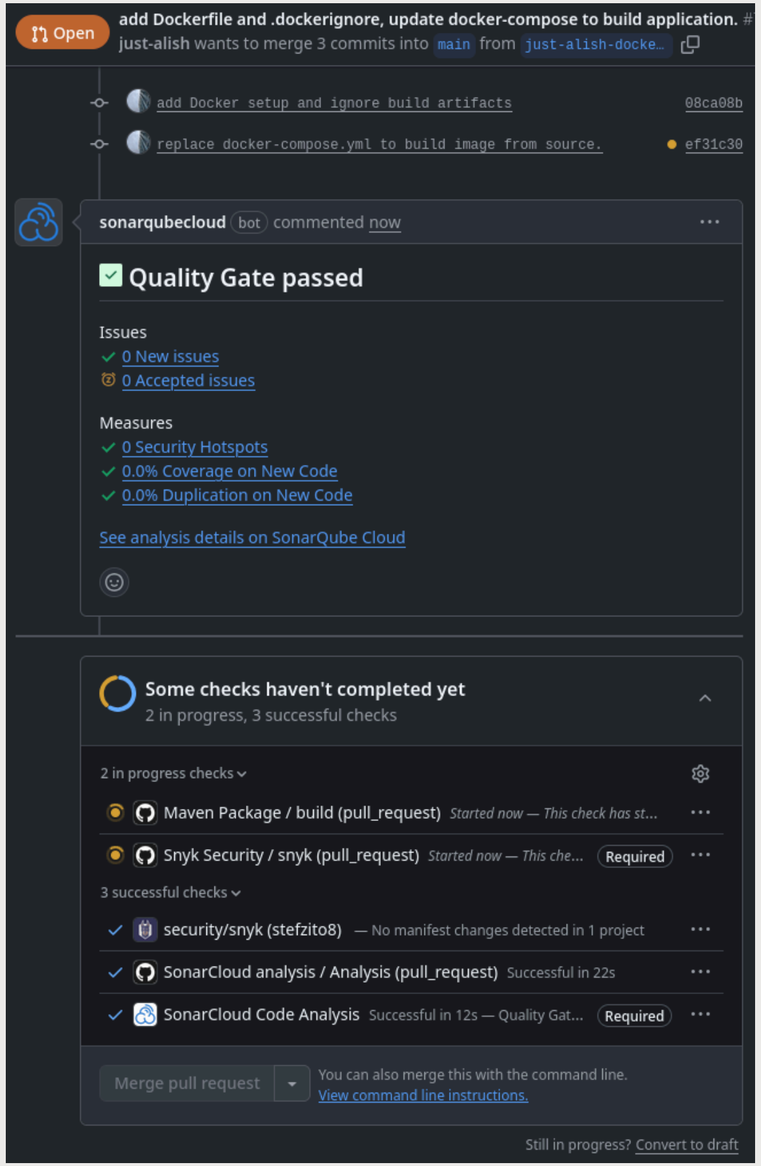


- **Tooling:** Used OpenJML for formal specification and static analysis.
- **Targeted Components:**
  - *DAO Classes:* Verified at the base level to prevent "contract errors" from propagating upward.
  - *Bean Classes:* Basic specifications (getters/setters) to support DAO calls.
  - *Cart Class:* Verified logic for adding/deleting items in user sessions.
- **Why this matters:** If the data layer is mathematically proven correct, the rest of the app rests on a stable foundation.

Implementation & CI Results

Integrating formal methods into the modern development pipeline.

- **Modular Specification:**
  - Interfaces: Defined pre-conditions and post-conditions (inherited by implementations).
  - Implementations: Added loop invariants, assertions, and nullability checks.
- **CI/CD Integration:**
  - Automated via Maven Package GitHub Action.
  - JML checks run automatically on every code push.
- **Outcome:**
  - 100% Success: No contradictions found between specifications and code.
  - Verified "Pure" getters and "Assignable" setter constraints.



Dockerization & Security Pivot

Portable, accessible, secure-ish and headache-free.

Discovery 1

Research 2

Planning 3

Execution 4

Optimization 5

Test cases

~1000 test cases - result of applying distinct testing methodologies tailored to the specific responsibilities of each architectural layer.

Model Layer: Black-Box approach

Equivalence Partitioning
Boundary Value Analysis

Control Layer: White-Box strategy

Statement Coverage
Branch Coverage

Filter layer: a hybrid strategy

Both Black-Box and White-Box
Mockito

Integration strategy: The Modified Sandwich

H2 test database
Simulation of upper layer

Code coverage

JaCoCo - primary indicator of our White-Box testing effectiveness.

The analysis prioritized two specific metrics:

1. TER1 (Instruction Coverage) - target of 80%
2. TER2 (Branch Coverage) - target of 70%

Outcome

Instruction coverage of 95%

Branch coverage of 80%

Analysis revealed areas where coverage metrics must be interpreted with context.

Mutation testing

Final Mutation Coverage score of 83% and a Test Strength of 89%

Notable portion of surviving mutants were identified as "equivalent".

Campaign uncovered valid gaps, particularly within the Controller layer's exception-handling logic.

JMH micro-benchmarks

~100 benchmark tests

Utilized an **H2** in-memory database as a test double.

Targeted specific components based on their execution frequency and computational complexity:

1. Authentication Filter
2. Cart Operations
3. Database Access (DAO)

Throughput was the primary metric for the Authentication Filter.

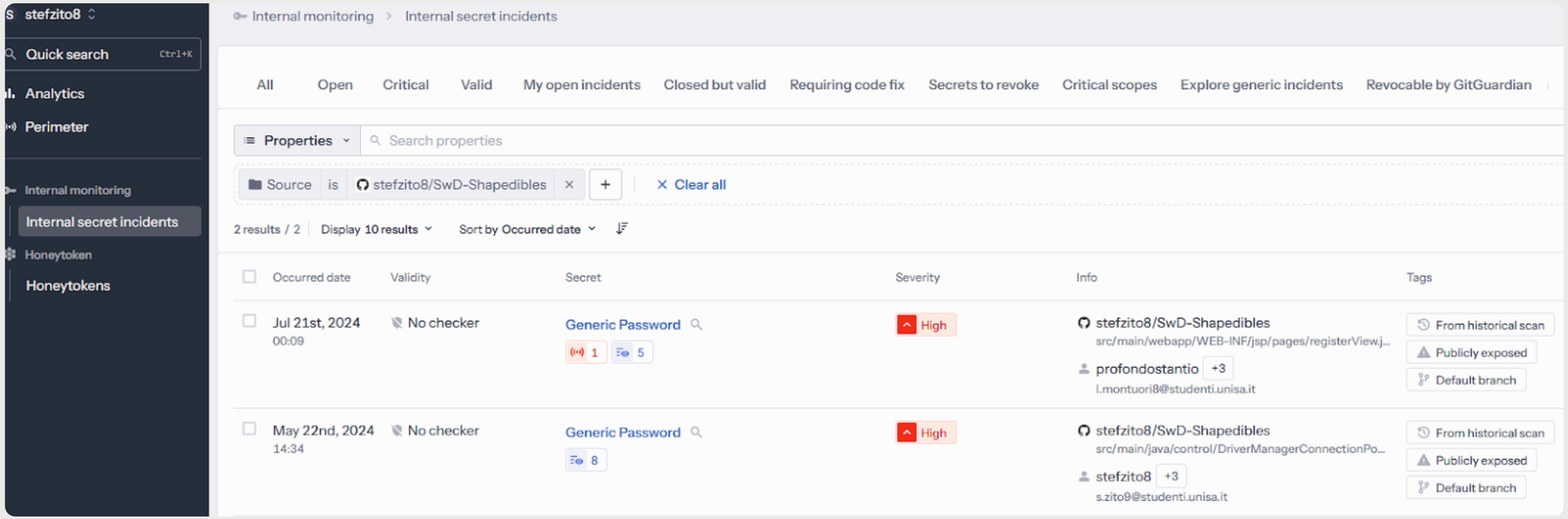
For Cart Operations and Database Searches, we focused on **Average Time** (latency).

Benchmarks successfully **highlighted code-level inefficiencies**.

Security Assessment & DevSecOps

Continuous vulnerability scanning and secret protection.

- **Integrated Toolset:**
  - Snyk & SonarQube Cloud: Fully integrated into the CI/CD pipeline via GitHub Actions for automated scanning on every push.
  - GitGuardian: Utilized for secret detection and leaked credential monitoring.
- **Key Findings:**
  - Secret Management: Initial scan flagged two hardcoded secrets; these were successfully remediated (Current count: 0).
  - Code Quality & Security: Vulnerabilities identified by Snyk and SonarQube were tracked and resolved (detailed in the technical appendix).
- **The Result:** A transparent security posture where all scan results are visible directly within the GitHub Actions dashboard.



Vulnerability Purging

Resolving found code Secrets and Vulnerabilities.

- **The Secret Codes:**
  - The first code: revealed itself to be a false alarm. The secret was simply ignored.
  - The second code: contained hardcoded credentials to the original database connection. was eliminated, resolving the secret.
- **The Vulnerabilities:**
  - False alarms: Out of 71 vulnerabilities found by Snyk, 70 revealed themself to be false alarms. All were “Use of Hardcoded Credential”
  - The real one: The only real vulnerability found by Snyk was a “Trust Boundary Violation”. The issue was solved by simply implementing the correct validation and cleaning mechanism for the user data inside of the filter.java class.

Inclusion of LLMs in the study

Artificial assistants and their role in learning

- **Conceptual understanding:**
  - LLMs were used as an explanatory aid to accelerate learning and reduce onboarding time, rather than as an authoritative source. In particular, they were employed to clarify the syntax and semantics of Docker Compose .yml files, the behaviour of Docker images and tags, and common Git workflows, including error recovery and retrying failed operations.
- **Artefact generation:**
  - LLMs were also used to assist in the generation of auxiliary artefacts that support development and analysis. These included initial drafts of unit tests, Java Modeling Language (JML) specifications for selected methods, and configuration files related to testing and verification workflows.
- **Assistants, not creators:**
  - LMs were not used to generate core application logic or architectural components. Their role in artefact generation was limited to reducing repetitive effort and providing structured templates, while correctness, completeness, and suitability remained the responsibility of the authors. The information provided by them was cross-checked against official documentation and validated through experimentation.

The End

We thank you for your time.