DIGITAL BILL PROCESSING SYSTEM:

A Role-Based Workflow, Version-Control and Audit System for Tracking Bills from Drafting to Publication at the National Assembly of Zambia

By

Chali Gerald Chibwana (2017011498)

Prince Daka (2021418987)

Mitchell Ndona (2021395111)

Supervisors:

Dr Lighton Phiri[1]

Mr Paul Chola[2]

Mr Alinani Sichone[2]

Mr Daniel Sichanga[2]

A report submitted to the Department of Computing and Informatics, The University of Zambia, in partial fulfilment of the requirements of the degree of Bachelor of Science in Computer Science (Software Engineering)

THE UNIVERSITY OF ZAMBIA

LUSAKA

2026

[1] Department of Computing and Informatics, School of Natural and Applied Sciences, University of Zambia

[2] National Assembly of Zambia

DECLARATION

[GUIDANCE: The student should declare that the report is their own work, identify any material drawn from other sources, and acknowledge collaboration where applicable.]

I declare that this final year project report is my own work and has been prepared in accordance with the academic requirements of the University of Zambia. All sources of information, ideas, data, software, images, and other material that are not my own have been appropriately acknowledged and referenced.

Where this project was undertaken as part of a group, I have clearly identified my individual contributions and have not presented the work of other group members as my own.

Student Name:

_____

_____

Signature:

_____

_____

Computer Number:

_____

_____

Date:

_____

_____

APPROVAL

[GUIDANCE: To be completed and signed after the report has been reviewed and accepted by the supervisor and, where required, the department.]

This final year project report has been submitted for examination with the approval of the project supervisor.

Supervisor Name:

_____

_____

Signature:

_____

_____

Date:

_____

_____

Departmental/Examiner Approval (if applicable):

_____

_____

ABSTRACT

Bills at the National Assembly of Zambia (NAZ) pass through three institutions and eight stages, from drafting at the Ministry of Justice, through First, Second, Committee, Report and Third Reading in the House, to presidential assent and publication by the Government Printer. Hand-offs between these offices are largely manual, so it is difficult to establish where a Bill is, which text is current, who changed it and whether statutory time limits apply. This project designed, implemented and evaluated a web-based Digital Bill Processing System that gives every Bill a single, shared and auditable record. Following a design science approach and an Agile Scrum process, the team built a three-tier system: a Next.js 14 client hosted on Vercel, a stateless Spring Boot 4 REST API running in Docker on Amazon EC2, and a PostgreSQL database on Amazon RDS whose schema is versioned through ten Flyway migrations. The Bill lifecycle is enforced as an eight-state finite state machine with role-based access control, so only the office that owns a stage can advance a Bill, save a version or decide an amendment. The system also provides append-only version control with a line-based comparison view, amendment management, full-text search, in-app notifications, statutory-period tracking with legal citations, reports exported to PDF and Excel, and an audit trail protected by a database trigger. Verification used 125 JUnit test methods in 15 test classes (158 executed test cases, all passing), run by a GitHub Actions pipeline with a Docker smoke test. Validation used a user acceptance test on the live deployment in which 25 participants carried out three tasks covering search, version comparison and stage advancement, and completed the System Usability Scale (SUS) and Technology Acceptance Model (TAM) items. Participants completed 90.7% of task attempts, gave a mean SUS score of 70.8, rated as good and above the benchmark of 68, and rated perceived usefulness, ease of use and intention to use between 4.22 and 4.35 out of 5. The results show that a role-based workflow system can make Bill processing at NAZ traceable and transparent; the main difficulty reported was unfamiliarity with legislative language rather than the interface, and the system is ready for a supervised pilot with National Assembly staff.

Keywords: legislative information systems; Bill tracking; workflow state machine; role-based access control; version control; audit trail; Spring Boot; usability evaluation

ACKNOWLEDGEMENTS

We thank God for the strength and opportunity to complete this project.

We are sincerely grateful to our supervisor, Dr Lighton Phiri of the Department of Computing and Informatics, for his guidance, his candid feedback at each review, and his insistence on a software engineering approach that is rigorous and evidence-based.

We are equally grateful to our co-supervisors at the National Assembly of Zambia, Mr Paul Chola, Mr Alinani Sichone and Mr Daniel Sichanga, for explaining how Bills move through the House, for reviewing our requirements and for their feedback on the system.

We thank the Department of Computing and Informatics at the University of Zambia for the academic environment that made this work possible, the students who gave their time to evaluate the system, and our families and friends for their support and encouragement.

TABLE OF CONTENTS

DECLARATION ii
APPROVAL iii
ABSTRACT iv
ACKNOWLEDGEMENTS v
LIST OF TABLES x
LIST OF FIGURES xi
LIST OF ACRONYMS AND ABBREVIATIONS xiii
CHAPTER ONE 1
INTRODUCTION 1
1.1 Study Background 1
1.2 Problem Statement 2
1.3 Study Objectives 2
1.3.1 Broad Objective 2
1.3.2 Specific Objectives 3
1.3.3 Research Questions 3
1.4 Study Rationale 3
1.5 Scope and Limitations 4
1.5.1 Specific Scope 4
1.5.2 Limitations 4
1.5.3 Addressing Limitations 5
1.6 Ethical Considerations 5
1.7 Report Organisation 6
CHAPTER TWO. 7
RELATED WORK 7
2.1 The Legislative Process as a Regulated Workflow 7
2.2 Digital Parliaments and Existing Systems 7
2.3 Workflow Modelling and Finite State Machines 8
2.4 Role-Based Access Control 9
2.5 Version Comparison 9
2.6 Audit Trails, Web Architecture and Security 9
2.7 Usability and Technology Acceptance 10
2.8 Summary and Research Gap 10

CHAPTER THREE 12
METHODOLOGY 12
3.1 Introduction 12
3.2 Research Approach 12
3.3 Development Methodology 13
3.3.1 Agile Scrum 13
3.3.2 Branching, Review and Continuous Integration 13
3.4 Requirements Elicitation 14
3.5 Design Principles 14
3.6 Tools and Technologies 15
3.7 Data Strategy 16
3.8 Project Management and Version Control 16
3.9 Evaluation Design 17
3.9.1 Verification 17
3.9.2 Validation: User Acceptance Test 17
CHAPTER FOUR 19
SYSTEM ANALYSIS AND DESIGN 19
4.1 Analysis of the Current Process 19
4.2 Actors and Roles 19
4.3 Functional Requirements 20
4.4 Non-Functional Requirements 21
4.5 Use Case Model 21
4.6 Bill Lifecycle Design 23
4.6.1 The Bill Lifecycle State Machine 23
4.6.2 Statutory Periods 24
4.7 System Architecture 25
4.8 Data Design 27
4.9 Interface (API) Design 29
4.10 Security Design 30
4.11 User Interface Design 30
4.12 Deployment Design 31
CHAPTER FIVE 32
IMPLEMENTATION 32
5.1 Codebase Structure 32

5.2 Authentication and Authorisation 33
5.3 Bill Workflow and Stage Transitions 34
5.4 Events, Audit and Notifications 34
5.5 Version Control and Comparison 35
5.6 Amendments and Official Documents 36
5.7 Search, Reports and Statutory Periods 36
5.8 Frontend Implementation 37
5.9 Integration, Deployment and Operations 42
5.10 Implementation Challenges and Solutions 43
5.11 Team Contributions 44
CHAPTER SIX 45
TESTING, RESULTS AND EVALUATION 45
6.1 Introduction 45
6.2Testing Strategy 45
6.3 Test Cases and Results 46
6.3.1 Automated Test Suite 46
6.3.2 Representative System Test Cases 47
6.3.3 Functional Completeness Against Requirements 48
6.4 Performance and Operational Results 48
6.5 User Evaluation 49
6.5.1 Participants 49
6.5.2 Task Completion 49
6.5.3 Usability (SUS) 50
6.5.4 Acceptance (TAM) 52
6.5.5 Qualitative Feedback 52
6.6 Analysis of Results 53
6.7 Objective-by-Objective Evaluation 54
6.8 Individual Results 54
6.8.1 Chali Gerald Chibwana (Backend Lead) 55
6.8.2 Prince Daka (DevOps and Testing Lead) 55
6.8.3 Mitchell Ndona (Frontend Lead) 55
6.9 Chapter Summary 56

CHAPTER SEVEN 57
DISCUSSION 57
7.1 Introduction 57
7.2 Interpretation of Findings 57
7.2.1 A Single, Enforced Record of Each Bill (RQ1, RQ2; O1, O2) 57
7.2.2 A Tamper-Evident History and Accountability (RQ3; O3, O4) 58
7.2.3 Usability and Acceptance (RQ4; O6) 59
7.2.4 Delivery and Verification (O5, O6) 60
7.3 Comparison with Existing Work 61
7.4 Contributions of the Project 61
7.5 Limitations and Implications 63
7.6 Chapter Summary 65
CHAPTER EIGHT 66
CONCLUSION AND RECOMMENDATIONS 66
8.1 Conclusion 66
8.2 Recommendations for the NAZ Handover 67
8.3 Future Work 67
REFERENCES 69
APPENDICES 73
APPENDIX A: INDIVIDUAL CONTRIBUTION STATEMENTS 73
APPENDIX B: USER MANUAL / INSTALLATION GUIDE 85
APPENDIX C: ADDITIONAL SYSTEM DESIGNS / DIAGRAMS 87
APPENDIX D: ADDITIONAL TEST CASES AND RESULTS 89
APPENDIX E: DATA COLLECTION INSTRUMENTS 90
APPENDIX F: ADDITIONAL CODE / CONFIGURATION 93
APPENDIX G: OTHER SUPPORTING MATERIAL 94

LIST OF TABLES

Table 1: Comparison of existing legislative systems with the Digital Bill Processing System 8
Table 2: Research questions, objectives, methods and evidence 12
Table 3: Tools and technologies used, with justification 15
Table 4: Team members, roles and main responsibilities 16
Table 5: Sprint plan and delivered increments (from the repository history) 17
Table 6: User roles, stage ownership and responsibilities 20
Table 7: Functional requirements by module and implementation status 20
Table 8: Non-functional requirements and design measures 21
Table 9: Flyway migrations and the features they introduced 29
Table 10: Backend packages and REST endpoints 29
Table 11: Development and runtime environment 32
Table 12: Continuous integration and deployment workflows 43
Table 13: Challenges encountered and how they were solved 43
Table 14: Principal contributions of each team member 44
Table 15: Test environment 46
Table 16: Automated test suite by class 46
Table 17: Representative system test cases and results 47
Table 18: Task completion by task (n = 25) 50
Table 19: Summary of SUS scores (n = 25) 51
Table 20: Mean response to each SUS item (1 = strongly disagree, 5 = strongly agree; n = 25) 51
Table 21: Technology acceptance items (1 = strongly disagree, 5 = strongly agree; n = 23) 52
Table 22: Evaluation of each objective 54
Table 23: Common problems and solutions 86
Table 24: Bill stages, owners and statutory periods 88
Table 25: All automated test methods by class 89
Table 26: Consent and device questions 90
Table 27: Task questions 90
Table 28: Usability and acceptance items 90
Table 29: Scored results of each participant (n = 25) 91
Table 30: Supervisor feedback and how it was addressed 94

LIST OF FIGURES

Figure 1: Design science research process followed in this project (adapted from Peffers et al. [37]) 11
Figure 2: Scrum workflow used for each backlog item, from product backlog to sprint review 13
Figure 3: Use case diagram of the Digital Bill Processing System 22
Figure 4: Sequence diagram: advancing a Bill to the next stage (UC6) 23
Figure 5: State machine of the Bill lifecycle, with stage owners and statutory periods 24
Figure 6: Three-tier architecture of the Digital Bill Processing System 26
Figure 7: Entity-relationship diagram of the database (crow's foot notation) 28
Figure 8: Deployment diagram: Vercel, GitHub Actions and AWS (eu-west-1) 31
Figure 9: Class diagram of the Bill module 33
Figure 10: Comparison of version 1 (Bill as introduced) with version 2 (Act as enacted) (live deployment) 35
Figure 11: Sign-in page (demonstration data) 37
Figure 12: Dashboard on the live deployment, including practice Bills created for the user evaluation 38
Figure 13: Bills list with search and stage filter (demonstration data) 38
Figure 14: Bill detail page for a published Bill: stage pipeline, details, period panel and action panel (live deployment) 39
Figure 15: Versions panel: the Bill as introduced (v1) and as enacted (v2), with links to text, changes and PDFs (live deployment) 39
Figure 16: Stage history of the Agricultural Marketing Bill, with officer, office, time and notes for each transition (live deployment) 40
Figure 17: Amendments queue with two amendments pending decision (live deployment) 41
Figure 18: Reports page: Bills by legislative stage, with PDF and Excel export (live deployment) 41
Figure 19: Processing times by stage, with the period in law and its citation beside each stage (live deployment) 42
Figure 20: Audit trail with sign-ins, failed sign-ins, stage changes and version uploads, visible to the Administrator only (live deployment) 42

Figure 21: Backend CI run of 16 October 2026 on GitHub Actions: 158 tests run, 0 failures, 0 errors, BUILD SUCCESS 47
Figure 22: Self-reported task completion by task (n = 25) 49
Figure 23: SUS score of each participant, in ascending order, with the mean and the benchmark of 68 51
Figure 24: Sequence diagram: signing in and calling a protected endpoint (UC1) 87
Figure 25: Sequence diagram: proposing and deciding an amendment (UC7, UC8) 88

LIST OF ACRONYMS AND ABBREVIATIONS

Abbreviation	Meaning
API	Application Programming Interface
AWS	Amazon Web Services
BCrypt	Adaptive password-hashing function based on the Blowfish cipher
CI/CD	Continuous Integration / Continuous Delivery
CORS	Cross-Origin Resource Sharing
CRUD	Create, Read, Update and Delete
CSV	Comma-Separated Values
DSR	Design Science Research
DTO	Data Transfer Object
EC2	Amazon Elastic Compute Cloud
ECR	Amazon Elastic Container Registry
ERD	Entity-Relationship Diagram
FR	Functional Requirement
GIN	Generalised Inverted Index (PostgreSQL)
HTTP(S)	Hypertext Transfer Protocol (Secure)
JPA	Jakarta Persistence API
JSON	JavaScript Object Notation
JWT	JSON Web Token
LCS	Longest Common Subsequence
MoJ	Ministry of Justice
NA	National Assembly
NAZ	National Assembly of Zambia
NFR	Non-Functional Requirement
OIDC	OpenID Connect
PDF	Portable Document Format
PR	Pull Request
RBAC	Role-Based Access Control
RDS	Amazon Relational Database Service
REST	Representational State Transfer
SHA-256	Secure Hash Algorithm, 256-bit
SO	Standing Order (National Assembly of Zambia Standing Orders, 2024)
SQL	Structured Query Language
SRS	Software Requirements Specification
SSM	AWS Systems Manager

Abbreviation	Meaning
SUMI	Software Usability Measurement Inventory
SUS	System Usability Scale
TAM	Technology Acceptance Model
UAT	User Acceptance Testing
UI	User Interface
UMUX-Lite	Usability Metric for User Experience, two-item version
UUID	Universally Unique Identifier

CHAPTER ONE

INTRODUCTION

The National Assembly of Zambia (NAZ) is the country's legislature. Under the Constitution it makes laws, oversees the Executive and approves public expenditure [1]. Law-making is its central function: a proposed law, a *Bill*, is drafted, debated, amended, passed by the House, assented to by the President and published before it becomes an Act of Parliament [1], [2]. This report presents the design, implementation and evaluation of the Digital Bill Processing System, a web application that tracks every Bill through that process in a single, shared and auditable record.

This chapter gives the background to the study (Section 1.1), states the problem (Section 1.2), sets out the objectives and research questions (Section 1.3) and the rationale (Section 1.4), defines the scope and limitations (Section 1.5), discusses ethical considerations (Section 1.6), and outlines the organisation of the report (Section 1.7).

1.1 Study Background

In Zambia, Government Bills are prepared by the Ministry of Justice, which drafts the text on behalf of the sponsoring ministry. Once a Bill is published and presented to the House, the National Assembly takes it through a sequence of stages defined by the Standing Orders: First Reading, Second Reading, Committee Stage, Report Stage and Third Reading [2]. After the House passes the Bill, it is presented to the President for assent, and the assented Act is then published by the Government Printer. Two of these stages carry periods prescribed by law. Standing Order 113(1) provides that a committee may not report a Bill back to the House before twenty working days have passed, and Article 66(1) of the Constitution, read with Standing Order 127(1) and (6), gives the President twenty-one days to assent, after which the Bill is deemed to have been assented to [1], [2].

Parliaments around the world are moving such processes onto digital platforms. The Inter-Parliamentary Union (IPU) reports steady investment by parliaments in document management, legislative tracking and open parliamentary data [3], and supports this through its Centre for Innovation in Parliament [4]. Mature examples include the UK Parliament's Bills website, which shows each Bill's stages, versions and amendments [5], and Congress.gov,

which tracks United States legislation from introduction to enactment [6]. In Africa, the Bungeni system developed under the United Nations Africa i-Parliaments Action Plan showed that configurable legislative workflow software is feasible for African legislatures [7].

Zambia's own policy direction points the same way. The Electronic Government Act No. 41 of 2021 promotes the use of information and communication technology in public institutions [8], while the Data Protection Act No. 3 of 2021 and the Cyber Security and Cyber Crimes Act No. 2 of 2021 set the legal frame for handling government information securely [9], [10]. Within NAZ, a parallel final year project is digitising the management of parliamentary questions and motions [11]; the present project addresses the Bill process.

1.2 Problem Statement

The movement of a Bill between the Ministry of Justice, the National Assembly and the Government Printer depends largely on paper files, physical registers and email. Each office holds its own copy of the Bill and its own record of what has happened to it. As a result, there is no single place where an authorised officer can see the current stage of every Bill, the version of the text that is current, the amendments that have been proposed and decided, or who took each action and when. Officers risk working from an outdated copy, progress across offices is difficult to monitor and report on, and the record of decisions is weak when it is needed for accountability.

The National Assembly therefore needs a secure, centralised system that records every Bill from drafting to publication, enforces the order of the legislative stages and the responsibility of each office, keeps every version and amendment decision, informs the next office when it must act, applies the time limits that the law prescribes, and leaves a complete audit trail.

1.3 Study Objectives

1.3.1 Broad Objective

To design, implement and evaluate a secure, web-based system that digitises the processing of Bills at the National Assembly of Zambia, giving each Bill a single, traceable record from drafting to publication.

1.3.2 Specific Objectives

The specific objectives were:

1. To analyse the Bill process at NAZ and specify the functional and non-functional requirements of a Bill processing system in a Software Requirements Specification (SRS).
2. To design and implement an eight-stage Bill workflow in which each stage is owned by one office and every stage change is authorised, recorded and visible to the next office.
3. To implement version control for Bill text, including the ability to compare any two versions, and an amendment process in which proposals are decided by the owning office with recorded reasons.
4. To provide search, notifications, statutory-period tracking, reporting and an append-only audit trail that support monitoring and accountability.
5. To deploy the system to a publicly accessible cloud environment through an automated build, test and deployment pipeline.
6. To verify the system through automated testing and validate it through a user acceptance evaluation of its usability and acceptability.

1.3.3 Research Questions

1. What stages, actors, rules and data govern the movement of a Bill at NAZ, and what requirements follow from them?
2. How can the Bill lifecycle be implemented so that every state change is valid, authorised and auditable?
3. How can a system keep a complete, tamper-evident history of a Bill's text, amendments and decisions?
4. How usable and acceptable do users find a system that implements this design?

1.4 Study Rationale

- **For the National Assembly and the Ministry of Justice:** one live view of every Bill, so that the question "where is this Bill and which text is current?" is answered instantly rather than by telephone or file search.
- **For accountability:** each stage change, version and amendment decision is tied to an authenticated officer and a time, and the audit log cannot be altered, even by someone with database access.

- **For legal compliance:** the system shows statutory periods only where the law prescribes them, with the legal citation, and never presents an internal target as a legal deadline.
- **For Members of Parliament and the public:** faster, more traceable law-making and a foundation for future publication of Bill status as open data.
- **For the University:** a documented application of software engineering practice (requirements engineering, layered design, automated testing, continuous delivery and usability evaluation) to a real public-sector problem.

1.5 Scope and Limitations

1.5.1 Specific Scope

In scope. The project covers Government Bills from the point at which the Ministry of Justice registers a draft to the point at which the Government Printer records publication. It includes authentication with four roles (Ministry of Justice officer, National Assembly officer, Government Printer and Administrator); Bill registration; stage-by-stage advancement with history; version control with comparison; amendment proposal and decision; storage of official PDF documents; full-text search and filtering; in-app notifications; statutory-period tracking; a dashboard and reports with PDF and Excel export; an append-only audit trail; and deployment to Vercel and Amazon Web Services (AWS).

Out of scope. Private Members' Bills; the Order Paper, questions and motions (covered by a separate project [11]); electronic voting, Hansard transcription, typesetting of the official Gazette, e-mail and SMS notifications, a user-management screen, and integration with external systems such as the Government Gazette or the Parliament website.

Temporal scope. Development ran from 8 July 2026, when the repository was initialised, to 16 October 2026, when implementation was frozen for evaluation.

1.5.2 Limitations

- **Access to users.** The user evaluation was carried out with 25 proxy users rather than with Assembly staff, below the target of 30, so it measures usability for new users rather than acceptance in daily work.
- **Data.** The system was demonstrated with Bills and Acts published on the Parliament website and ZambiaLII and with practice Bills; no confidential working drafts were used.

- **Time.** A fourteen-week development window meant that comments on Bills (FR-022), scheduled reminders (FR-028) and the report on amendment activity (FR-042) were deferred.
- **Infrastructure.** The API runs on a single EC2 instance in one AWS region; load testing and high availability were not part of the project.

1.5.3 Addressing Limitations

- **Proxy users.** The tasks were chosen so that people with no parliamentary training could perform them, the results are interpreted as a measure of learnability for new users (Section 7.3), and a pilot with National Assembly staff is the first recommendation for the handover (Section 8.2).
- **Realistic data.** Real Bills and Acts published on the Parliament website and ZambiaLII were loaded with their actual versions, so the version comparison and stage history show genuine legislative changes rather than invented ones.
- **Deferred features.** Deferred requirements were agreed with the supervisor, are recorded against their FR numbers (Section 4.3), and are carried into future work (Section 8.3).
- **Single-instance hosting.** Snapshots of the managed database and a deployment workflow that rebuilds the server from the repository mean the system can be restored or moved; high availability is a recommendation for the handover.

1.6 Ethical Considerations

- **Informed consent and anonymity.** The questionnaire explained the purpose of the study, stated that participation was voluntary and that participants could stop at any time, and collected no names, e-mail addresses or computer numbers. Only the type of device used (phone or computer) was recorded as a participant characteristic.
- **Data protection.** The system processes the names and roles of officers. It follows the principles of the Data Protection Act No. 3 of 2021 [9]: data minimisation, access only by authorised roles, and protection of data in transit (HTTPS) and at rest (hashed passwords, a managed database in a private network).
- **Security.** The design addresses the OWASP Top 10 risks [12]: access control is enforced on the server, passwords are hashed with BCrypt, tokens are signed and expire, and responses use DTOs so that no internal fields such as password hashes can leak. These measures support the Cyber Security and Cyber Crimes Act No. 2 of 2021 [10].

- **Legal accuracy.** The system shows a statutory period only where the Constitution or Standing Orders prescribe one, and always cites the provision; it never presents an administrative target as a legal deadline.
- **Data used.** Demonstrations and the user study used Bills and Acts already published on the Parliament website and ZambiaLII [13], and clearly labelled practice Bills. No confidential draft Bill was entered into the system.
- **Intellectual property.** Open-source libraries were used under their licences, and all sources are acknowledged in the References.

1.7 Report Organisation

Chapter 2 reviews related work on parliamentary information systems, workflow modelling, access control, version comparison, audit trails and usability evaluation. Chapter 3 describes the research and development methodology. Chapter 4 presents the analysis and design of the system, including requirements, use cases, the Bill lifecycle, architecture and data design. Chapter 5 describes the implementation and the contribution of each team member. Chapter 6 covers testing, and Chapter 7 evaluates the system against its requirements and objectives, including the user acceptance test. Chapter 8 concludes and makes recommendations for the handover to NAZ and for future work. The references and appendices (individual contribution statements, user manual, additional designs and test cases, the evaluation instrument and code excerpts) follow.

CHAPTER TWO.

RELATED WORK

This chapter reviews the knowledge on which the system rests: the legislative process (Section 2.1), existing parliamentary systems (Section 2.2), workflow and state machines (Section 2.3), role-based access control (Section 2.4), version comparison (Section 2.5), audit trails and web security (Section 2.6), and usability and technology acceptance (Section 2.7). Section 2.8 summarises the review and identifies the research gap.

2.1 The Legislative Process as a Regulated Workflow

In Westminster-derived parliaments such as NAZ, a Bill moves through a fixed sequence of readings, a committee stage and a report stage before it is passed [2]. Each stage has a responsible office, and the rules about who may act, in what order and within what time are set by the Constitution and the Standing Orders [1], [2]. A Bill process is therefore a *regulated workflow*: an ordered set of activities, each performed by an authorised role, whose correctness can be checked against written rules. Van der Aalst et al. catalogued the control-flow patterns that such workflows use; the Bill process is a strict *sequence* with role-based task assignment, with no parallel branches and no stage that may be skipped [14].

2.2 Digital Parliaments and Existing Systems

- **UK Parliament Bills website.** Publishes every Bill before Parliament with its stages, dates, versions of the text and amendment papers [5]. It is an exemplary public tracking service, but it presents the outcome of internal processes rather than supporting the drafting and stage-management work itself.
- **Congress.gov (United States).** Tracks legislation from introduction to enactment, with full text of each version, actions and committee activity, and supports advanced search [6]. It is a large public information service maintained by the Library of Congress, not a workflow tool for a small legislature.
- **Bungeni.** An open-source parliamentary and legislative information system developed for African parliaments under the Africa i-Parliaments Action Plan. It manages the workflow of Bills, questions and motions with configurable states and uses XML document models [7]. It demonstrated the approach in the region but is no longer actively maintained and predates current web frameworks.

- **Akoma Ntoso.** An XML standard for parliamentary, legislative and judicial documents that enables machine-readable, interoperable legislation [15]. It is a data standard rather than a workflow system and is relevant as a future export format.
- **LEOS (European Commission).** Open-source software for drafting and editing legislation collaboratively, with versioning, built on Akoma Ntoso [16]. It focuses on the drafting of the text rather than on moving a Bill between institutions.
- **NAZ Order Papers System.** A sibling final year project that digitises parliamentary questions and motions at NAZ, with a permission-based lifecycle and duplicate detection [11]. It addresses a different business process but confirms the institution's need for workflow digitisation and offers a reference point for architecture and evaluation.

Table 1 compares these systems on the capabilities that the problem statement requires.

Table 1: Comparison of existing legislative systems with the Digital Bill Processing System

Capability	UK Bills site	Congress.gov	Bungeni	LEOS	This system
Internal stage-by-stage workflow	No (publication)	No (publication)	Yes	Drafting only	Yes (8 states)
Role ownership of each stage	Not public	Not public	Configurable	Partial	Yes, enforced on server
Versions of the Bill text	Yes	Yes	Yes	Yes	Yes, append-only
Side-by-side version comparison	Partial	Partial	No	Yes	Yes (line-based LCS)
Amendment proposal and decision	Published papers	Published actions	Yes	Comments	Yes, with reasons
Statutory periods with citations	No	No	No	No	Yes
Tamper-evident audit trail	Not public	Not public	Partial	Partial	Yes (DB trigger)
Fit for NAZ (cost, hosting, rules)	Not available	Not available	Unmaintained	EU-specific	Built for NAZ

The comparison shows that public tracking services document Bills well but do not support the internal hand-offs, while workflow and drafting tools either are no longer maintained or are tied to another institution's procedures. None encodes Zambian statutory periods or the division of responsibility between the Ministry of Justice, the National Assembly and the Government Printer.

2.3 Workflow Modelling and Finite State Machines

A finite state machine models an entity that is always in exactly one of a finite set of states and moves between them only through defined transitions. Harel's statecharts extended this notation for the design of reactive systems [17]. Modelling a Bill's stage as a state machine makes the legal rules executable: an illegal transition (for example, skipping Committee Stage,

or moving a Bill out of the terminal *Published* state) can be refused by construction rather than detected after the fact. This project uses an eight-state machine with guards based on the user's role (Section 4.6).

2.4 Role-Based Access Control

Role-based access control (RBAC) assigns permissions to roles rather than to individuals, and users acquire permissions through role membership [18]. The NIST RBAC model formalised this approach and is widely used in information systems [19]. RBAC suits the Bill process because responsibility is defined by office: any Ministry of Justice officer may act on a draft, and any National Assembly officer on a Bill between First and Third Reading. Broken access control is the most common critical web application risk in the OWASP Top 10, which is why the checks must be enforced on the server and not only in the user interface [12].

2.5 Version Comparison

Comparing two versions of a text is a classic problem in computing. Hunt and McIlroy's algorithm, the basis of the Unix *diff* utility, reduces it to finding the longest common subsequence (LCS) of the two sequences of lines; lines outside the LCS are reported as added or removed [20]. Myers later gave an O(ND) algorithm that is faster when the two versions differ little [21]. Because Bills are structured clause by clause, a line-based comparison shows exactly which clauses an amendment changed, which is how this project presents differences between versions (Section 5.5).

2.6 Audit Trails, Web Architecture and Security

An audit trail records who did what, to which object, and when. Its value depends on its integrity: a log that can be edited proves little. Common techniques include append-only storage, restricting database privileges, and computing cryptographic hashes so that any later change can be detected. This project combines an append-only audit table enforced by a database trigger with SHA-256 checksums on stored documents, so that integrity does not depend on the application alone.

Representational State Transfer (REST) is an architectural style for networked applications based on resources, standard methods and stateless interaction [22]. Statelessness lets each request be authenticated independently, typically with a bearer token. The JSON Web Token

(JWT) standard defines a compact, signed token that carries claims such as the user's identity and role [23]. Passwords are stored as BCrypt hashes, an adaptive function designed to remain expensive to attack as hardware improves [24]. Layered architectures that separate presentation, application and data tiers, and patterns such as the Data Transfer Object (DTO) [25], the Observer [26] and domain events [27], are standard means of keeping such systems maintainable.

2.7 Usability and Technology Acceptance

Usability is normally measured with standardised questionnaires. The System Usability Scale (SUS) is a ten-item questionnaire that yields a score from 0 to 100 [28]; across hundreds of studies the average SUS score is 68 [29], and Bangor et al. associate scores above about 70 with acceptable usability [30]. The Software Usability Measurement Inventory (SUMI) is a fifty-item instrument scored against a commercial reference database [31]. The Technology Acceptance Model (TAM) explains users' intention to use a system through its perceived usefulness and perceived ease of use [32]. Where questionnaires must be short, the UMUX-Lite uses just two items, one on usefulness and one on ease of use, that map onto TAM's two constructs and can be converted to a SUS-equivalent score [33], [34]. Section 3.9 explains how they were used in this project.

2.8 Summary and Research Gap

The literature and existing systems show that (i) parliaments benefit from digitising legislative workflows [3]; (ii) state machines and RBAC are established means of enforcing ordered, role-based processes [17], [19]; and (iii) version comparison and immutable audit records are mature techniques [20]. However, no documented system combines these into a single tool for the Zambian Bill process: one that enforces the order of stages and the responsibility of each office, applies Zambian statutory periods with their legal basis, preserves every version and amendment decision, and is evaluated with users. This project addresses that gap.

The project is framed by design science research, in which knowledge is produced by building and evaluating an artefact that solves a relevant problem [35]. The artefact is the Bill processing system; its design is grounded in the concepts of Sections 2.3 to 2.6 (state machine, RBAC, immutable history); and its evaluation draws on two complementary lenses: software quality,

assessed through verification against the requirements [36], and user acceptance, assessed through perceived usefulness and ease of use [32], [34]. Figure 1 summarises how these elements relate.

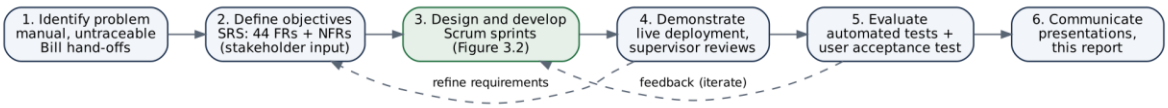


Figure 1: Design science research process followed in this project (adapted from Peffers et al. [37])

Bill processing is a regulated, role-based, sequential workflow. Finite state machines and RBAC make its rules enforceable, line-based comparison and append-only storage make its history trustworthy, and standard usability instruments allow its acceptability to be measured. Existing systems either publish Bill information or support drafting, but none fits the Zambian process. Chapter 3 describes how the project was carried out.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

Methodology is how a project answers its research questions. This chapter describes the research approach (Section 3.2), the development methodology (Section 3.3), requirements elicitation (Section 3.4), the design principles (Section 3.5), the tools and technologies (Section 3.6), the data strategy (Section 3.7), project management and version control (Section 3.8) and the evaluation design (Section 3.9). Table 2 relates each research question to the objective it supports, the method used to answer it, and where the answer appears.

Table 2: Research questions, objectives, methods and evidence

Research question	Objective	Method	Evidence
RQ1: stages, actors, rules and data of the Bill process	O1	Document analysis; meetings with NAZ co-supervisors; SRS (ISO/IEC/IEEE 29148)	Sections 3.4, 4.2–4.5
RQ2: valid, authorised, auditable state changes	O2	State-machine design; RBAC policy; unit tests of every transition	Sections 4.6, 5.2–5.7, 6.3
RQ3: complete, tamper-evident history	O3, O4	Append-only versions; database trigger; SHA-256 checksums; tests	Sections 4.8, 5.2–5.7, 6.3
RQ4: usability and acceptability	O5, O6	Live deployment; user acceptance test with three tasks, SUS and TAM items	Sections 3.9, 7.2

3.2 Research Approach

The project follows design science research (DSR), which produces knowledge by building an artefact that addresses a real organisational problem and evaluating it rigorously [35]. Hevner et al. require that the artefact be relevant to a real problem, grounded in existing knowledge, and evaluated with appropriate methods [35]. The team applied the six activities of Peffers et al. [37] as shown in Figure 1: the problem was identified with the National Assembly (Chapter 1); the objectives of a solution were defined as requirements (Chapter 4); the artefact was designed and built (Chapters 4 and 5); it was demonstrated on a public deployment using real Bills; it was evaluated through testing and a user study (Chapters 6 and 7); and the results are communicated in this report and in the project presentations.

3.3 Development Methodology

3.3.1 Agile Scrum

Within the DSR cycle, the software was built with Scrum, an iterative framework in which a team delivers working increments in fixed time-boxes called sprints [38]. Scrum was chosen over a plan-driven model because the details of the Bill process were clarified progressively through feedback from the supervisors, and because a working system was needed early for review [39]. The team held biweekly Scrum meetings to review the finished increment and plan the next one. These were held in person, moved online during the abrupt closure of schools, and returned to in-person meetings with the supervisors when the university reopened. The product backlog was the set of 44 functional requirements in the SRS, grouped into eleven modules and tracked on a Trello board. Each sprint selected a group of modules, and work was divided by architectural layer: backend, frontend, and DevOps and testing. Figure 2 shows how a backlog item moved from planning to review.



Figure 2: Scrum workflow used for each backlog item, from product backlog to sprint review

3.3.2 Branching, Review and Continuous Integration

Every change was made on a feature branch (for example *feature/version-control*) and merged into the protected main branch only through a pull request that had been reviewed by another member and had passed the continuous integration (CI) checks. Over the project the repository accumulated 95 commits and 31 pull requests. The sprints are summarised in Section 3.8.

Software quality assurance (SQA) was built into this process rather than added at the end: requirements were traceable to code and tests through their FR numbers, every pull request was peer reviewed, the CI pipeline ran linting, type-checking, unit and integration tests and a container smoke test on every change, and database changes were made only through versioned Flyway migrations [40], [41].

3.4 Requirements Elicitation

Requirements were gathered through four complementary elicitation techniques, consistent with ISO/IEC/IEEE 29148 [42]:

- **Document analysis.** The team studied the Constitution of Zambia [1], the National Assembly Standing Orders 2024 [2] and Bills published on the Parliament website [43] to establish the stages of a Bill, the responsible offices and the statutory periods.
- **Meetings with domain experts.** Requirements and the stage model were discussed with the National Assembly co-supervisors, Mr Paul Chola, Mr Alinani Sichone and Mr Daniel Sichanga, who explained how a Bill moves between the Ministry of Justice, the Table Office and the Government Printer and reviewed the team's understanding.
- **Review of comparable systems.** The systems reviewed in Section 2.2 were examined to identify expected capabilities, such as version comparison and stage tracking, and capabilities that do not apply to NAZ.
- **Supervisor reviews.** Progress reviews with Dr Lighton Phiri refined the scope and the evaluation approach, for example by making version control and search the critical features to demonstrate.

The output was a Software Requirements Specification (SRS) containing 44 functional requirements (FR-001 to FR-044) in eleven modules, and non-functional requirements for security, auditability, performance, usability, availability and maintainability. Each requirement was given an identifier that is carried into the code, commit messages and tests, so that every implemented feature can be traced back to the requirement it satisfies. The requirements are presented in Sections 4.3 and 4.4.

3.5 Design Principles

The research workflow followed the DSR cycle in Figure 1, and each build iteration followed the Scrum cycle in Figure 2. The software itself was designed as a three-tier web application: a browser-based presentation tier, a stateless REST application tier and a relational data tier. This structure was chosen because it separates concerns, allows the user interface and the API to be developed and deployed independently by different team members, and keeps all business rules on the server where they cannot be bypassed [12], [22]. Section 4.7 presents the architecture in detail.

Five principles guided the design decisions described in Chapters 4 and 5:

- **Rules live on the server.** Every permission and stage rule is enforced by the API; the interface only hides what a user cannot do.
- **One rule, one place.** Stage ownership and sequence are defined once, in a single policy class, and every action that changes a Bill consults it.
- **History is appended, never rewritten.** Versions, stage changes, amendment decisions and audit entries are only ever added; corrections are new records.
- **The law is quoted, not invented.** Only periods prescribed by the Constitution or the Standing Orders are shown as legal, always with their citation.
- **Everything traces to a requirement.** FR identifiers are carried into code, commit messages and tests.

3.6 Tools and Technologies

Table 3 lists the main technologies and the reason each was selected.

Table 3: Tools and technologies used, with justification

Layer	Technology	Justification
Frontend	Next.js 14 (React 18, TypeScript 5), Tailwind CSS, TanStack Query 5	Component-based UI with file-system routing [44]; static typing catches contract errors early; Query caches and refreshes server data
Backend	Spring Boot 4.1 on Java 21; Spring Security; Spring Data JPA (Hibernate)	Mature, well-documented framework for secure REST APIs [45]; declarative security and transactions
Authentication	JSON Web Tokens (jjwt 0.12), BCrypt	Stateless, signed tokens [23]; adaptive password hashing [24]
Database	PostgreSQL on Amazon RDS; Flyway migrations V1–V10	Relational integrity, built-in full-text and trigram search [46], [47]; versioned, repeatable schema changes [40]
Reports	Apache PDFBox 3, Apache POI 5	Server-side generation of PDF and Excel reports
Containers	Docker, Docker Compose	Identical environments for development, CI and production [48]
Cloud	Vercel (frontend); AWS EC2, ECR, RDS, SSM, IAM OIDC (backend)	Public HTTPS hosting; managed database with snapshots; deployments without stored access keys
CI/CD	GitHub Actions	Lint, type-check, tests and Docker smoke test on every pull request [41]
Testing	JUnit 5, Mockito, Spring MockMvc, Postman	Unit, web-layer and manual API testing
Design	Graphviz, Mermaid, Figma-style mock-ups in code	Version-controlled diagrams that can be regenerated
Collaboration	Git and GitHub (pull requests), Trello, WhatsApp	Branching, review, backlog and sprint tracking
Evaluation	Google Forms; Python (pandas, SciPy, Matplotlib)	Questionnaire delivery; reproducible scoring, statistics and charts

3.7 Data Strategy

The system was developed and evaluated with public and clearly labelled data only. Real Bills and Acts published on the Parliament website [43] and ZambiaLII [13] were entered with their actual texts, so that, for example, the Agricultural Marketing Bill, 2025 is held as the Bill as introduced (version 1) and the Agricultural Marketing Act, 2026 as enacted (version 2), with both official PDFs. Where the public record did not give the date of a stage, the stage history says so rather than inventing a date. For the user evaluation, each participant worked on a practice Bill whose title marks it as "for testing only". Four demonstration accounts, one per role, use invented officer names. No confidential working draft was entered into the system at any point.

3.8 Project Management and Version Control

The team of three divided the work by architectural layer, as shown in Table 4, while requirements, design, documentation and presentations were shared. The product backlog was kept on a Trello board and reviewed in the biweekly Scrum meetings described in Section 3.3.1.

Table 4: Team members, roles and main responsibilities

Member	Role	Main responsibilities
Chali Gerald Chibwana (2017011498)	Backend Lead	REST API, database schema and migrations, authentication, Bill workflow and statutory periods, versions, amendments, documents, notifications and audit; the frontend screens for these features
Prince Daka (2021418987)	DevOps and Testing Lead	AWS infrastructure, CI/CD pipeline and deployment workflow, search and reporting modules, most of the automated test suite, code review
Mitchell Ndona (2021395111)	Frontend Lead	Repository set-up, frontend foundation and reusable components, sign-in and Reports pages, user evaluation design and analysis

All code was kept in one GitHub repository and changed only through reviewed pull requests, as described in Section 3.3.2. Table 5 summarises the sprints, reconstructed from the repository history.

Table 5: Sprint plan and delivered increments (from the repository history)

Sprint (dates, 2026)	Requirement modules	Main increments delivered
Sprint 0 (8–14 Jul)	Set-up	Monorepo, Next.js scaffold with mock API, Docker Compose, CI workflow, Spring Boot scaffold, users table
Sprint 1 (15 Jul–9 Sep)	Authentication FR-001–004; Bill registration FR-005–008	JWT login and /auth/me, seeded role accounts, Bill entities and stage history, first version upload, Docker deployment configuration
Sprint 2 (10–23 Sep)	Workflow FR-009–012; status FR-025–028; search FR-033–036; collaboration FR-021–024; reporting FR-041–044	Role-based transitions, backend CI with Docker smoke test, search indexes, notifications, dashboard, Bill submission and detail pages, reports with PDF/Excel export, amendment API, AWS deployment workflow
Sprint 3 (13–16 Oct)	Audit FR-037–040; amendments FR-017–020; versions FR-013–016; documents FR-029–032	Append-only audit trail, amendments interface, version control with text and comparison, official PDF storage
Evaluation (17–21 Oct)	Validation	Implementation frozen; pilot; user acceptance test; presentations

3.9 Evaluation Design

The evaluation combined **verification**, which asks whether the system was built correctly against its requirements, and **validation**, which asks whether it is the right system for its users [39].

3.9.1 Verification

Verification used automated tests at two levels. Unit tests checked the domain policies, such as the transition rules, statutory periods, search criteria and stage durations, in isolation. Web-layer tests used Spring MockMvc to call each controller with different roles and check status codes, response bodies and access rules. The tests ran on every pull request in the GitHub Actions pipeline, which also built the Docker image and checked that the containerised API started and answered a login request. Manual API tests in Postman covered the happy path, authentication failures and error states.

3.9.2 Validation: User Acceptance Test

Following the supervisor's guidance to use a standard instrument, to test the critical features (version control and search in particular) and to record participant characteristics, the team designed a user acceptance test (UAT) in which participants used the live system and then completed an online questionnaire (Appendix E). The design was as follows:

- **Instrument.** The questionnaire combines the ten-item System Usability Scale [28], the most widely used standardised usability questionnaire, with three items for the constructs of the Technology Acceptance Model: perceived usefulness, perceived ease of use and intention to use [32]. All thirteen items use a five-point agreement scale. Earlier drafts, which also included the full TAM item set and timed tasks, were judged too long, so the final form was designed to take about five minutes.
- **Tasks.** Three tasks exercised the critical features: (T1) find a named Bill using search and report its current stage; (T2) open the Agricultural Marketing Bill, 2025, compare its two versions and write down one change; and (T3) move a practice Bill to its next stage. After each task participants reported whether they completed it easily, with some difficulty, or not at all. The written answers to T1 and T2 allow completion to be checked as well as self-reported.
- **Participants and recruitment.** The target was at least 30 participants. The link to the system and the form was shared with the team's classmates and with junior students in the Department of Computing and Informatics, and with a few students taking ICT in the School of Education, so that some participants came from outside computing. The form recorded the device used (phone or computer); it did not record programme or year of study.
- **Pilot.** Before the form was sent out, three people piloted the tasks and questionnaire on the live system, to check that the instructions were clear and that the form could be completed in about five minutes.
- **Ethics.** The form opened with a consent statement, and no names, e-mail addresses or computer numbers were collected (Section 1.6).
- **Metrics and analysis.** Task completion rate per task; accuracy of the written T2 answers; the SUS score of each participant, computed in the standard way and compared with the average of 68 [29] and the adjective scale of Bangor et al. [49]; and the mean of each TAM item. Differences were tested with a one-sample t-test against 68 and a Mann–Whitney U test between devices, and open-ended comments were grouped into themes.

CHAPTER FOUR

SYSTEM ANALYSIS AND DESIGN

4.1 Analysis of the Current Process

Before the system, a Government Bill moved between three institutions largely on paper. The Ministry of Justice drafted the Bill and sent it to the National Assembly; the Table Office recorded each reading, committees worked from their own copies, and the Bill as passed was sent for presidential assent and then to the Government Printer for publication. Each office kept its own register and its own copy of the text, and hand-offs were made by delivering documents or by e-mail.

Three weaknesses follow from this process and shaped the requirements. First, there is no shared view of status: finding where a Bill is requires asking the office that last handled it. Second, there is no authoritative history of the text: amendments made at Committee or Report Stage are applied to separate copies, and comparing versions is manual. Third, accountability depends on the completeness of separate paper registers, so the question of who changed what, and when, cannot always be answered. The design that follows addresses each weakness with, respectively, one shared record per Bill, append-only versions with comparison, and an audit trail that cannot be edited.

4.2 Actors and Roles

Four user roles were identified from the division of responsibility in the Bill process. Table 6 shows each role, the stages it owns and its main tasks. Ownership is exclusive: a Bill in a given stage can be advanced, versioned or have its amendments decided only by the role that owns that stage, or by an Administrator.

Table 6: User roles, stage ownership and responsibilities

Role	Organisation	Stages owned	Main tasks
MoJ officer (MOJ_OFFICER)	Ministry of Justice	Draft	Register Bills, save versions of the draft, upload the official PDF, release the Bill for First Reading
NA officer (NA_OFFICER)	National Assembly (Table Office, Committees)	First Reading to Third Reading	Advance Bills through the House stages, record amendments and decide them, save versions as amended
Government Printer (GOV_PRINTER)	Government Printer	Presidential Assent	Record assent and publication
Administrator (ADMIN)	NAZ ICT	Any (cannot skip stages)	Oversight, reports, audit trail and export

Secondary stakeholders are the sponsoring ministries, Members of Parliament, the Clerk of the National Assembly, who needs reports on the legislative programme, and the public, who benefit from faster and more transparent law-making.

4.3 Functional Requirements

The SRS groups the 44 functional requirements into eleven modules. Table 7 summarises each module, with representative requirements and their implementation status at the end of the project.

Table 7: Functional requirements by module and implementation status

Module (FRs)	Representative requirements	Status
Authentication (FR-001–004)	Sign in with username and password; role-based access; session expiry; failed sign-ins recorded	Implemented
Bill registration (FR-005–008)	MoJ officer registers a Bill with its long and short titles; unique reference number (e.g. NAZ-2026-007)	Implemented
Workflow management (FR-009–012)	Advance a Bill one stage at a time; only the stage owner may act; stage history with actor, time and notes	Implemented
Version control (FR-013–016)	Save a new version with a change summary; list all versions; compare any two versions	Implemented
Amendment management (FR-017–020)	Propose an amendment to a clause; accept or reject with recorded reasons; amendment history	Implemented
Collaboration (FR-021–024)	Notify the next owner when a Bill arrives (FR-023); comments on a Bill (FR-022)	Partly (FR-022 deferred)
Status tracking (FR-025–028)	Current stage, time in stage, statutory period with citation; scheduled reminders (FR-028)	Partly (FR-028 deferred)
Document management (FR-029–032)	Upload and download the official PDF of a Bill; integrity checksum	Implemented
Search and retrieval (FR-033–036)	Keyword search on reference and titles; filter by stage, submitter and date range; paging	Implemented
Audit trail (FR-037–040)	Record every action; who, what, when; prevent modification; export (Administrator)	Implemented
Reporting (FR-041–044)	Bills per stage; amendment activity (FR-042); processing time per stage; export to PDF and Excel	Partly (FR-042 deferred)

4.4 Non-Functional Requirements

The non-functional requirements in the SRS are summarised in Table 8 against the quality characteristics of ISO/IEC 25010 [36], together with the design measure that addresses each.

Table 8: Non-functional requirements and design measures

Quality characteristic	Requirement	Design measure
Security	Only authenticated users may access the system; each role may perform only its own actions	JWT bearer tokens; Spring Security filter chain; server-side RBAC in BillTransitionPolicy; BCrypt hashing; DTOs
Integrity / accountability	Every action is recorded; records cannot be altered	Audit row written in the same transaction as the action; database trigger rejects UPDATE and DELETE; SHA-256 checksums
Performance	Search and list pages respond within about two seconds for normal use	GIN full-text and trigram indexes; server-side paging; list queries never load document bodies
Usability	New users can complete core tasks without training	Consistent layout, stage pipeline bar, plain-language messages; measured in the UAT
Availability	Accessible over the internet during working hours	Managed hosting on Vercel and AWS; automatic container restart
Maintainability	Changes can be made safely by future developers	Layered packages by feature; typed API contract; versioned migrations; automated tests and CI
Portability	Runs in any environment with Docker	Containerised API; configuration by environment variables
Legal compliance	Statutory periods follow the Constitution and Standing Orders	Periods held with citation in the BillStage enumeration; none invented

4.5 Use Case Model

Figure 3 shows the twelve main use cases and the actors that participate in them. Every actor signs in (UC1), searches and views Bills (UC3) and receives notifications (UC10). The Ministry of Justice officer submits Bills (UC2). The officers who own a stage save versions (UC4), advance the Bill (UC6) and decide amendments (UC8). Only the Administrator views and exports the full audit trail (UC12).

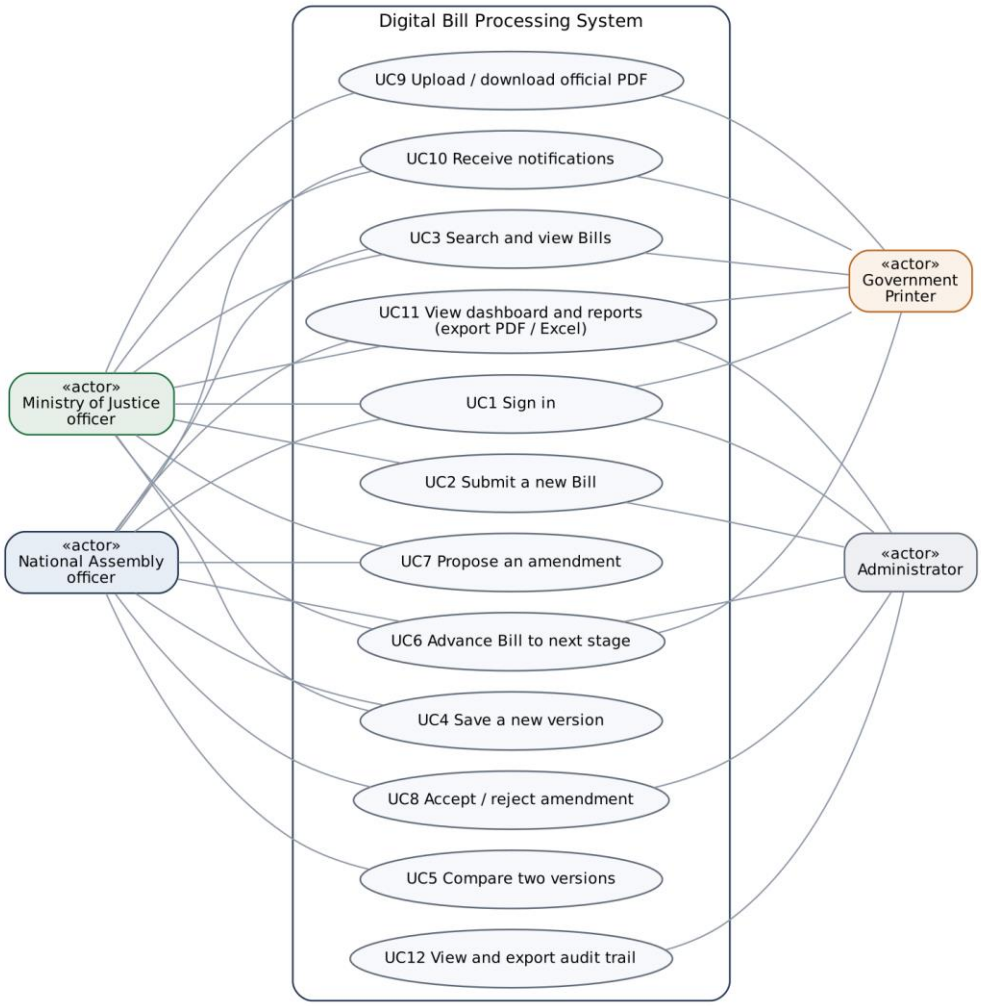


Figure 3: Use case diagram of the Digital Bill Processing System

Representative user stories, written in the product backlog form, are:

- As a **Ministry of Justice officer**, I want to register a new Bill and upload its official PDF, so that the National Assembly receives a complete, authoritative copy.
- As a **National Assembly officer**, I want to advance a Bill to the next stage, so that the Bill's status is always current and the next office is notified.
- As a **National Assembly officer**, I want to compare the version passed at Committee Stage with the version first presented, so that I can see exactly which clauses changed.
- As a **Government Printer**, I want to see the Bills awaiting assent and how long they have been waiting, so that I can act within the 21-day period.

- As an **Administrator**, I want to export the audit trail, so that I can answer questions about who changed a Bill and when.

Figure 4 shows the sequence of interactions when an officer advances a Bill, the most important use case. The request is authenticated by the JWT filter, checked by the transition policy, saved together with its stage history and audit entry in a single transaction, and only after the commit is a notification sent to the next owner.

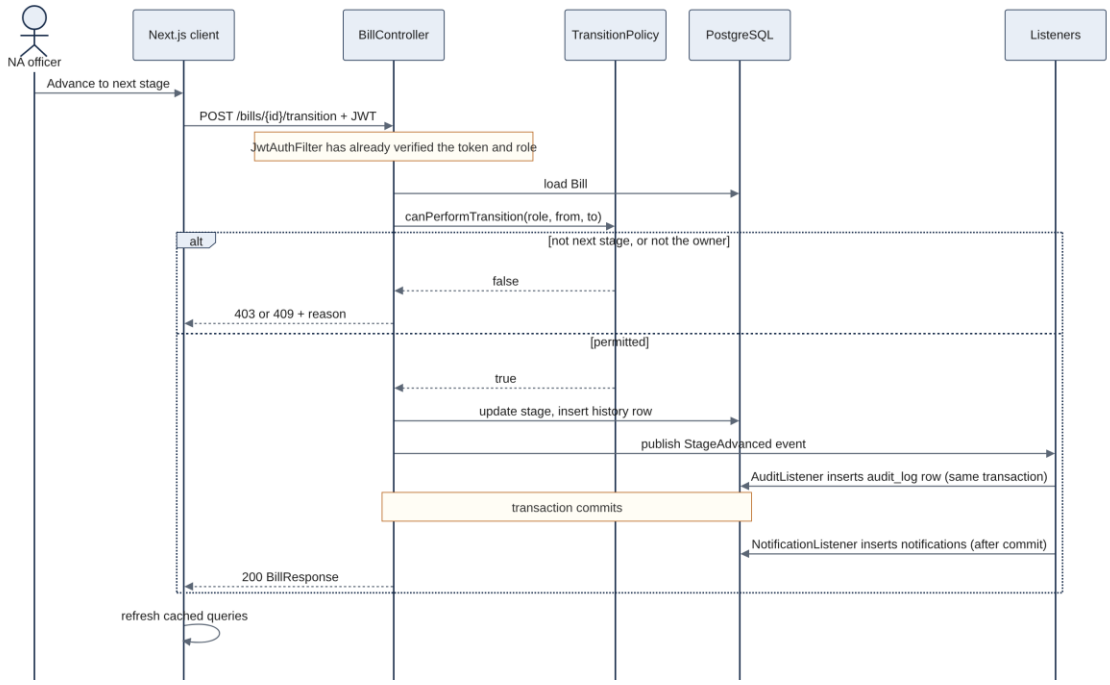


Figure 4: Sequence diagram: advancing a Bill to the next stage (UC6)

4.6 Bill Lifecycle Design

4.6.1 The Bill Lifecycle State Machine

The lifecycle of a Bill is modelled as a finite state machine with eight states (Figure 5). The transition function allows exactly one transition from each non-terminal state, to the next state in sequence, and only when the user's role owns the current state; the Administrator may act on any stage but, like everyone else, cannot skip one. *Published* is terminal. Two states carry a statutory period: *Committee Stage* has a minimum of twenty working days before the committee may report (SO 113(1)), and *Presidential Assent* has a maximum of twenty-one calendar days, after which assent is deemed given (Article 66(1); SO 127(1) and (6)).

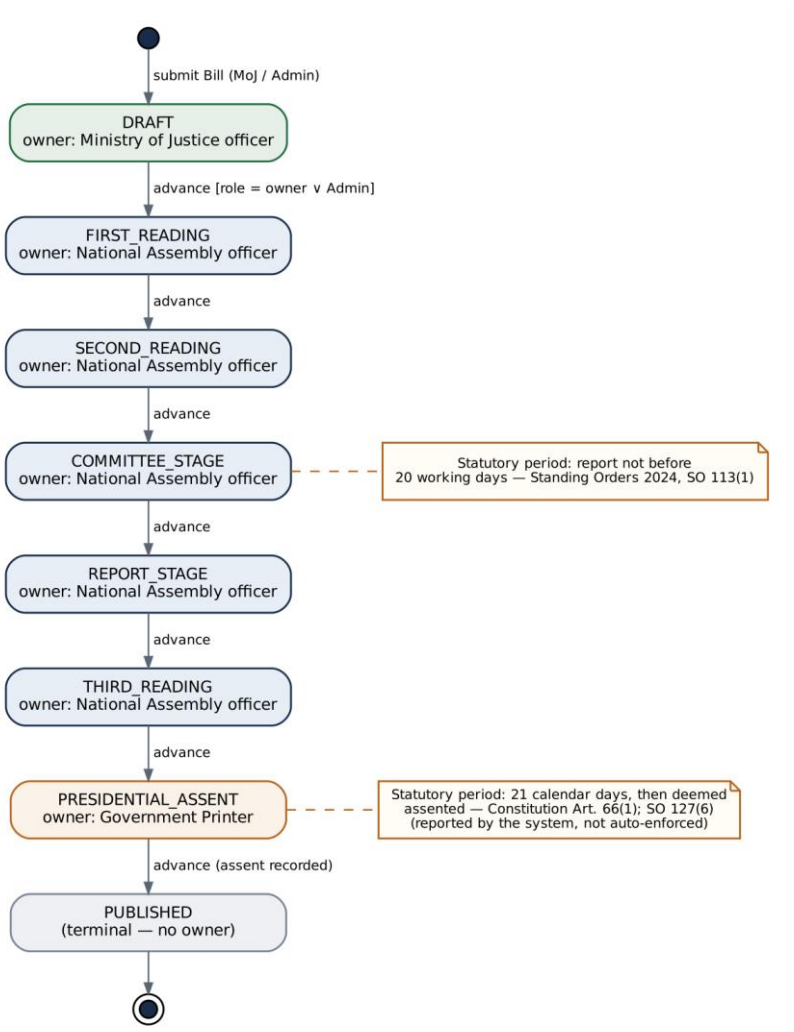


Figure 5: State machine of the Bill lifecycle, with stage owners and statutory periods

Formally, let S = {DRAFT, FIRST_READING, SECOND_READING, COMMITTEE_STAGE, REPORT_STAGE, THIRD_READING, PRESIDENTIAL_ASSENT, PUBLISHED} be the states in order, *owner(s)* the role that owns state *s*, and *r* the role of the user. A request to move a Bill from state $*s*_i$ to state *t* is permitted if and only if $t = *s*_{i+1}$ and (r = owner($*s*_i$) or r = ADMIN). Because the ordering and ownership are held in code as an enumeration and a map, the rule is checked in constant time and cannot be bypassed by a crafted request.

4.6.2 Statutory Periods

BillDeadlinePolicy resolves the date associated with a stage in a fixed order of precedence: a period prescribed by law, if any; otherwise an administrative target configured by the National

Assembly, if any; otherwise none. Working days are counted by skipping Saturdays and Sundays. The effect of a period follows the law: a Committee Stage period is a minimum and can never make a Bill "overdue", and an elapsed assent period means that assent is deemed given, not that someone is late. Only an administrative target can be overdue. Public holidays are not yet excluded, so the computed SO 113(1) date is labelled as indicative.

4.7 System Architecture

Figure 6 shows the architecture of the system. It has three tiers:

- **Presentation tier.** A Next.js 14 application served by Vercel. App Router pages (dashboard, Bills, Bill detail, amendments, reports and audit) are composed from reusable components such as the search bar, the stage pipeline bar and the versions panel. A typed API client wraps every endpoint, and TanStack Query caches server data and refreshes it after a change. The token and current user are kept in the browser session store.
- **Application tier.** A Spring Boot 4 REST API on Java 21, packaged as a Docker container on an Amazon EC2 instance. Each request passes through the Spring Security filter chain, where a JWT filter authenticates it, before reaching a controller. Controllers delegate decisions to domain policies (BillTransitionPolicy for who may move a Bill and to which stage, and BillDeadlinePolicy for statutory periods) and persist changes through Spring Data JPA repositories. After a change, the controller publishes a domain event (for example *BillStageAdvanced*). The AuditListener records the event in the same transaction, while the NotificationListener notifies users only after the transaction commits. Report writers produce PDF and Excel files.
- **Data tier.** A PostgreSQL database on Amazon RDS whose eight tables are created and changed only by Flyway migrations V1 to V10. Hibernate validates the entity mappings against this schema at start-up, rather than generating it.

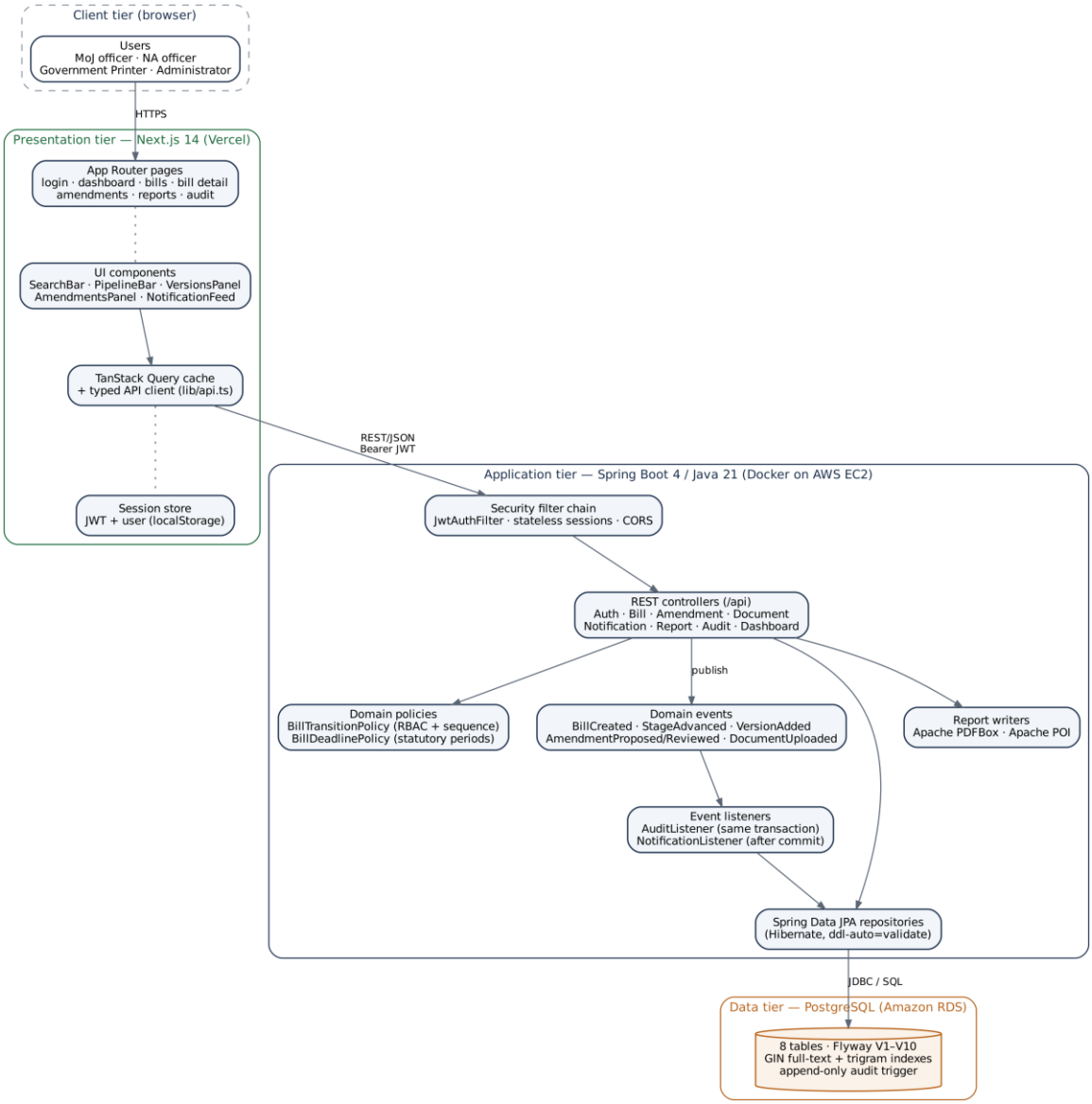


Figure 6: Three-tier architecture of the Digital Bill Processing System

Two architectural decisions deserve explanation. First, the API is **stateless**: no session is stored on the server, and every request carries a signed token [22], [23]. This lets the API be restarted or replaced without signing users out and simplifies deployment. Second, the system uses **domain events** with the Observer pattern [26], [27], so that controllers do not call the audit and notification services directly. New reactions to an event can be added without changing the code that raises it, and the transactional phase of each listener expresses a business rule: an audit record is part of the action and must succeed with it, whereas a notification is a courtesy and must never block the action.

4.8 Data Design

Figure 7 shows the entity-relationship diagram (ERD) of the database. The central table, *bills*, holds one row per Bill with its unique reference number, titles, current stage, current version number and the time it entered its stage. The other tables record what happens to a Bill:

- *bill_stage_history* records every stage change with the stages before and after, the officer and the time (FR-011).
- *bill_versions* holds every version of the text, numbered per Bill, with a change summary and the stage at which it was saved. Versions are append-only: a correction is a new version, never an edit.
- *amendments* records each proposed amendment against a clause and, optionally, the version it amends, with its status (PENDING, ACCEPTED or REJECTED), the reviewer and the reasons.
- *bill_documents* stores official PDF files with their size and SHA-256 checksum.
- *notifications* holds in-app notifications for each recipient, with read status.
- *audit_log* holds one row per action. The actor's username and role are copied into the row, so that a later change to a user cannot rewrite history, and a trigger rejects every UPDATE and DELETE.
- *users* holds accounts, BCrypt password hashes and roles.

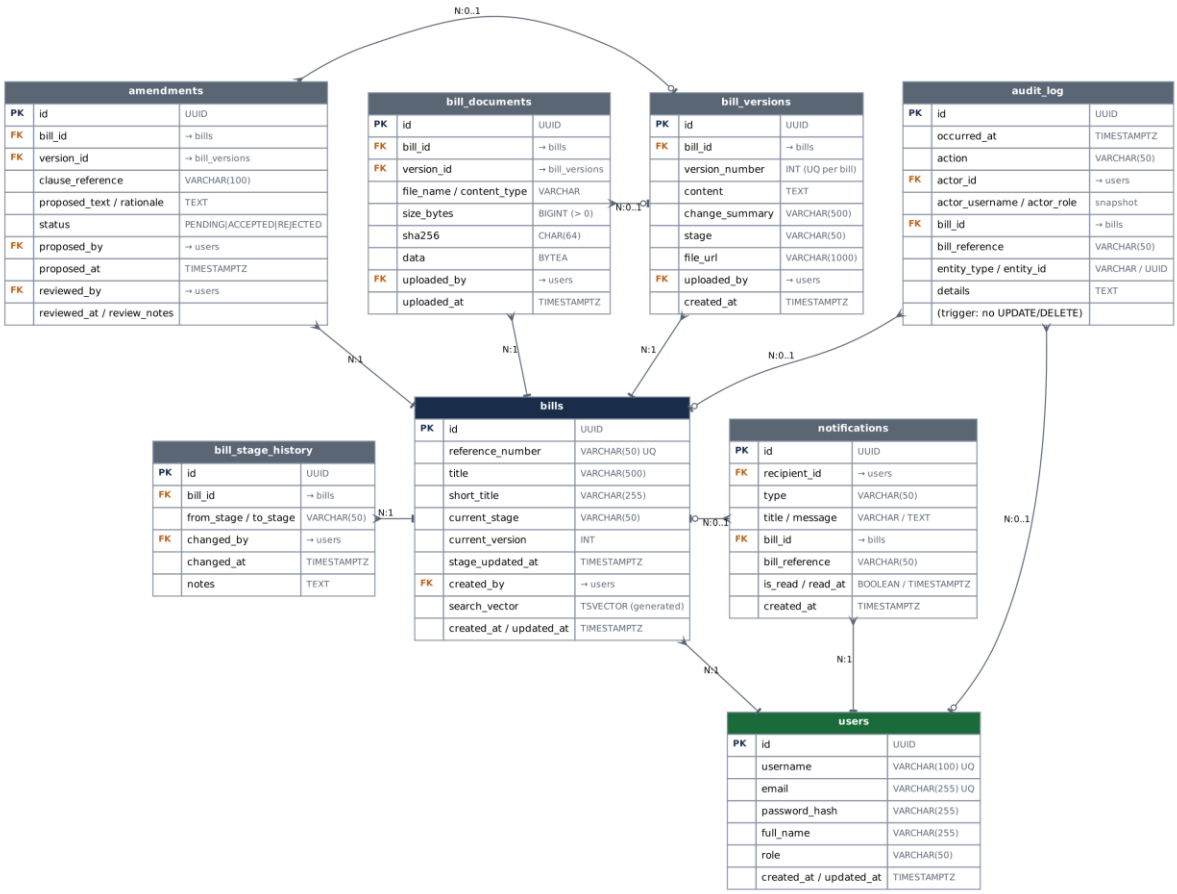


Figure 7: Entity-relationship diagram of the database (crow's foot notation)

All primary keys are UUIDs, so identifiers cannot be guessed or enumerated, and all timestamps are stored with time zones. The schema is in third normal form, with two deliberate exceptions for the audit trail and notifications: the Bill reference and the actor's name are copied so that each record remains meaningful on its own, even if related data change later. Table 9 lists the ten migrations that built the schema incrementally, one per feature.

Table 9: Flyway migrations and the features they introduced

Migration	Change	Requirements
V1	users table with roles	FR-001–004
V2	bills, bill_stage_history, bill_versions	FR-005–016
V3	Short title and summary columns on bills	FR-005
V4	Rename stage DRAFTING to DRAFT	FR-009
V5	pg_trgm extension, generated tsvector column, GIN and B-tree search indexes	FR-033–036
V6	notifications	FR-023
V7	amendments	FR-017–020
V8	audit_log with append-only trigger	FR-037–040
V9	Version text (content) and change summary	FR-013–016
V10	bill_documents with SHA-256 checksum	FR-029–032

4.9 Interface (API) Design

The API follows REST conventions with JSON bodies and is served under the */api* context path. Each request other than sign-in carries a bearer token, and every response is a DTO rather than an entity. Table 10 summarises the packages and the endpoints they expose.

Table 10: Backend packages and REST endpoints

Package	Responsibility	Main endpoints
auth	Sign-in, token issue and validation, security configuration	POST /auth/login; GET /auth/me
bill	Bills, stages, transitions, versions, statutory periods, search	GET/POST /bills; GET /bills/{id}; POST /bills/{id}/transition; GET /bills/{id}/workflow, /history; GET/POST /bills/{id}/versions
amendment	Proposal and decision of amendments	POST/GET /bills/{id}/amendments; GET /amendments; PATCH /amendments/{id}/review
document	Official PDF storage	POST/GET /bills/{id}/documents; GET /documents/{id}/file
notification	In-app notifications	GET /notifications; PATCH /notifications/{id}/read, /read-all
dashboard	Counts by stage and recent activity	GET /dashboard/stats
report	Reports and exports	GET /reports/bills-per-stage[/pdf|/xlsx]; GET /reports/processing-times[/pdf|/xlsx]
audit	Audit trail and export	GET /audit; GET /audit/export; GET /bills/{id}/audit
user	Accounts, roles and seeded demonstration users	(internal)

4.10 Security Design

- **Authentication.** Passwords are stored only as BCrypt hashes [24]; sign-in issues a signed JWT that expires after 24 hours by default [23]; sessions are stateless.
- **Authorisation.** Every action that changes a Bill passes through one RBAC policy on the server [19]; a role outside the stage owner receives *403 Forbidden*, and a skipped stage *409 Conflict*.
- **Data exposure.** Responses are DTOs, so internal fields such as password hashes are never serialised; identifiers are UUIDs that cannot be enumerated.
- **Integrity.** Audit rows are written in the same transaction as the action and are protected by a database trigger; documents carry SHA-256 checksums and are accepted only if they are genuine PDFs.
- **Transport and secrets.** All traffic uses HTTPS; secrets are supplied as environment variables; deployments obtain short-lived AWS credentials through OpenID Connect, so no access keys are stored [12].

4.11 User Interface Design

The interface was designed to look and behave like a formal government system: a restrained navy and green palette drawn from the National Assembly identity, a fixed sidebar with the main areas, and consistent page headers. The key interaction design decisions were:

- **One record per Bill.** The Bill detail page (Figure 14) brings together everything about a Bill: the stage pipeline, the statutory period panel, versions and comparison, amendments, documents, stage history and the Bill's own audit trail.
- **Show only permitted actions.** Buttons such as *Advance stage* or *Save new version* appear only for the role that owns the current stage. The server enforces the same rule, so hiding a button is a convenience, not a security measure.
- **Make the stage visible.** A pipeline bar shows all eight stages, with completed stages, the current stage and the remaining stages in distinct styles.
- **Legal periods are labelled as legal.** Where a statutory period applies, the panel shows the date, how it is computed (working or calendar days) and the provision, for example "Standing Orders 2024, SO 113(1)".
- **Search first.** The Bills list (Figure 13) opens with a search box and a stage filter, because finding a Bill is the most frequent task.

Screens of the implemented system are presented in Section 5.8 and Appendix C.

4.12 Deployment Design

Figure 8 shows how the system is deployed. The frontend is built and served by Vercel. The backend image is built by GitHub Actions, pushed to Amazon ECR and started on EC2 through AWS Systems Manager (SSM), using short-lived credentials obtained through OpenID Connect, so that no AWS access keys are stored in the repository. Because Flyway migrations cannot be undone, the deployment workflow refuses to run until the operator confirms that an RDS snapshot has been taken.

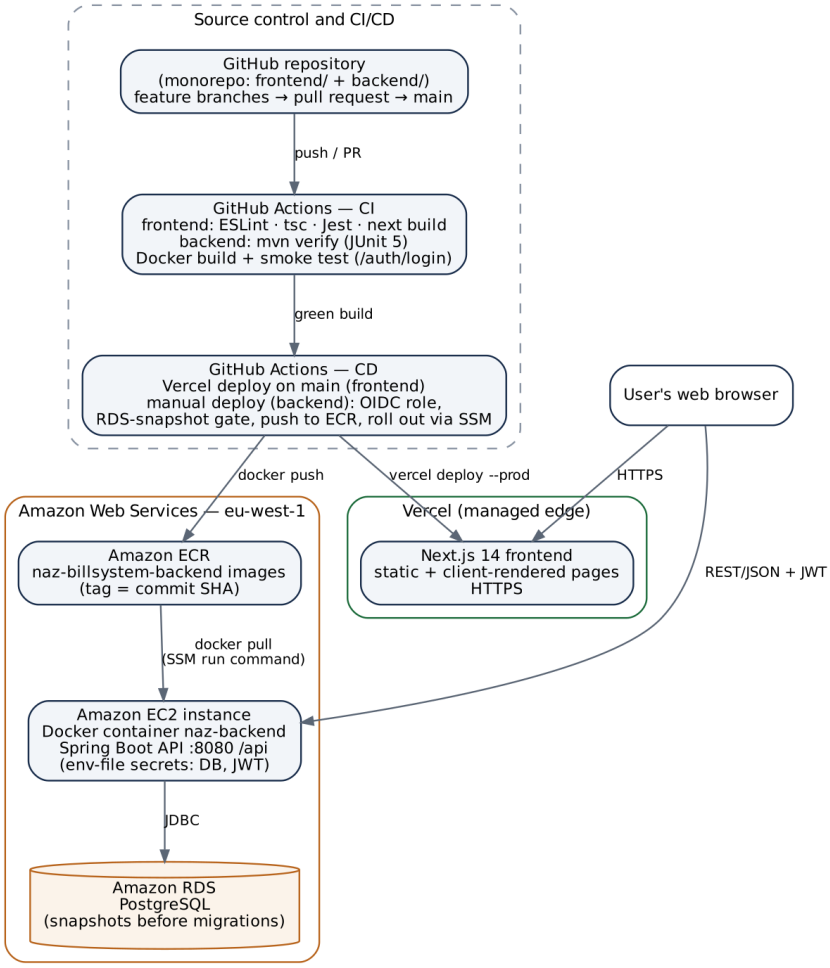


Figure 8: Deployment diagram: Vercel, GitHub Actions and AWS (eu-west-1)

CHAPTER FIVE

IMPLEMENTATION

This chapter describes how the design in Chapter 4 was implemented, from the codebase structure (Section 5.1) through the main modules (Sections 5.2 to 5.7), the frontend (Section 5.8), integration and deployment (Section 5.9) and the challenges met (Section 5.10). Section 5.11 summarises the contribution of each team member.

5.1 Codebase Structure

The members developed locally with Visual Studio Code (with a shared set of recommended extensions), Git, Docker Desktop, Node.js 22 and JDK 21. A single Docker Compose file started PostgreSQL and the API locally, so every member ran the same database version as production. Configuration, such as the database URL, the JWT secret and token lifetime, and the allowed origins, was supplied through environment variables documented in *.env.example*, so no secret was committed to the repository. The code is held in one GitHub monorepo, with *frontend/*, *backend/* and *.github/workflows/* folders.

Table 11: Development and runtime environment

Item	Development	Production
Frontend runtime	Next.js dev server (Node.js 22)	Vercel (naz-bill-system.vercel.app)
Backend runtime	Spring Boot on JDK 21 or Docker	Docker container on Amazon EC2 (eu-west-1)
Database	PostgreSQL in Docker Compose	Amazon RDS for PostgreSQL
Schema	Flyway on start-up	Flyway on start-up, after an RDS snapshot
Build and test	npm, Maven wrapper (./mvnw)	GitHub Actions
Size of code base	Java: 73 source files, about 5,200 lines; 2,800 lines of tests	TypeScript: 29 files, about 3,900 lines; SQL: 10 migrations

The backend is organised by feature rather than by technical layer. Each package holds the entity, repository, controller, DTOs and events of one module, so that a developer working on amendments, for example, finds everything in one place. The packages and their endpoints are listed in Table 10 (Section 4.9).

Figure 9 shows the main classes of the Bill module. Entities (*Bill*, *BillVersion*, *BillStageHistory*) are mapped to tables; controllers expose them only as response DTOs; and the two policy classes are stateless components that hold the business rules, which makes them simple to unit-test.

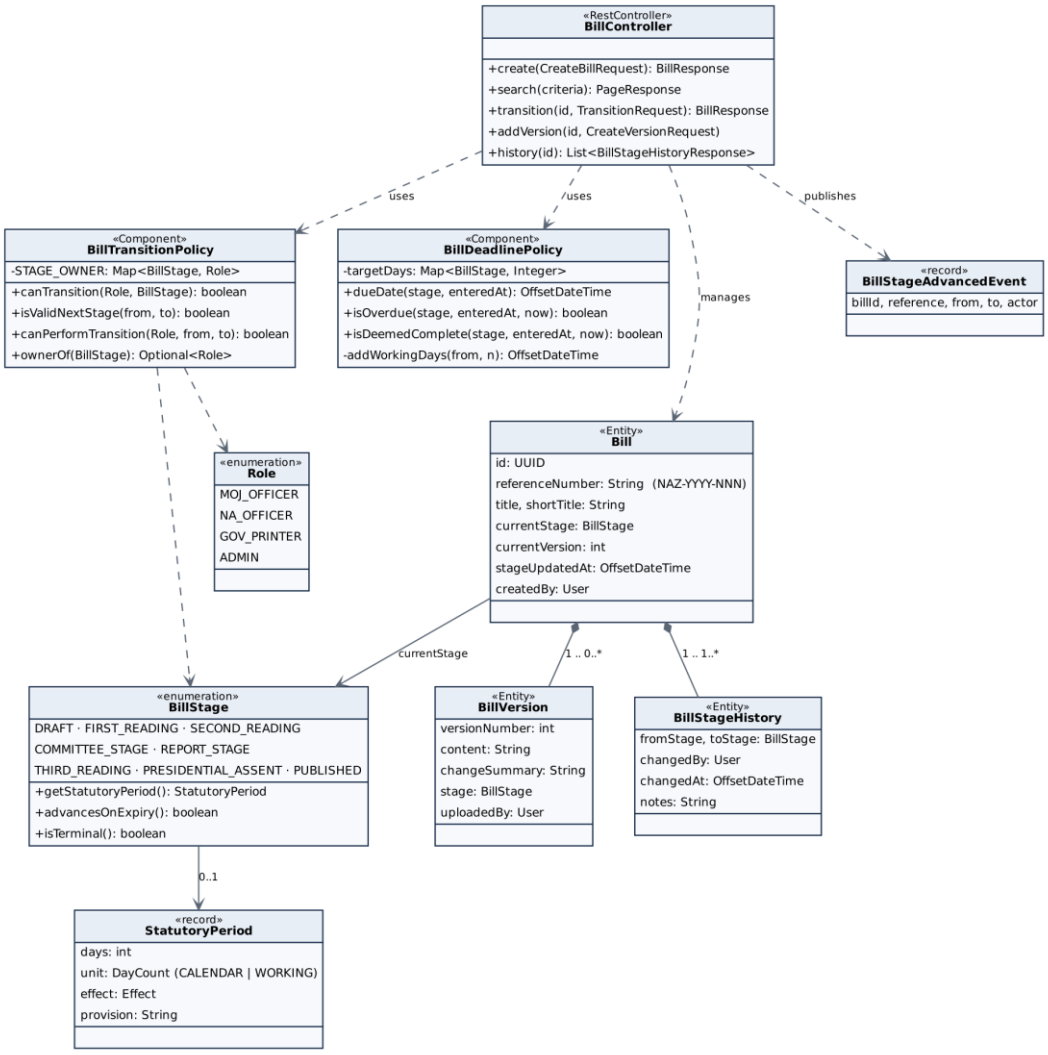


Figure 9: Class diagram of the Bill module

5.2 Authentication and Authorisation

Sign-in posts a username and password to */auth/login*. The password is checked against its BCrypt hash and, if it matches, the server returns a JWT signed with HMAC-SHA256 that carries the username and role and expires after 24 hours by default. Spring Security is configured to be stateless; every other route requires a valid token. A custom *JwtAuthFilter* validates the token on each request and places the user and role in the security context. Both successful and failed sign-ins are written to the audit trail. Authorisation for Bill actions is not left to annotations alone: each action calls *BillTransitionPolicy*, so the rule "only the owner of the current stage may act" is expressed once and used everywhere.

```
public boolean canPerformTransition(Role role, BillStage current, BillStage target) {
    return isValidNextStage(current, target)  // one step forward only
        && canTransition(role, current);       // owner of stage, or ADMIN
}
```

Listing 1: BillTransitionPolicy: one entry point for every stage change

5.3 Bill Workflow and Stage Transitions

A transition request names the target stage. The controller loads the Bill, asks the policy whether the move is permitted, and returns *403 Forbidden* if the user's role does not own the stage or *409 Conflict* with a reason if the target is not the next stage. If the move is permitted, the controller updates the Bill, inserts a stage-history row and publishes a *BillStageAdvancedEvent*, all in one database transaction (Figure 4).

5.4 Events, Audit and Notifications

Six domain events are published: Bill created, stage advanced, version added, amendment proposed, amendment reviewed and document uploaded. Two listeners react to them, with deliberately different timing. *AuditListener* is a plain event listener, so it runs inside the publishing transaction: if the audit row cannot be written, the whole action rolls back, and no action can exist without its audit record. *NotificationListener* runs only after the transaction commits, and any failure is logged and swallowed, because a notification is a courtesy that must never stop a Bill. The notification goes to every user who holds the role that owns the Bill's new stage, as returned by *BillTransitionPolicy.ownerOf()*.

The audit table is protected by the database itself. Migration V8 installs a trigger that raises an exception on any UPDATE or DELETE, so even a future bug or a person with SQL access cannot quietly change the record:

```
CREATE FUNCTION audit_log_is_append_only() RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'audit_log is append-only: % is not permitted', TG_OP;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER audit_log_no_update_or_delete
    BEFORE UPDATE OR DELETE ON audit_log
    FOR EACH ROW EXECUTE FUNCTION audit_log_is_append_only();
```

Listing 2: Append-only audit trail enforced by a trigger (migration V8)

5.5 Version Control and Comparison

Two versions are compared line by line using the longest common subsequence (LCS) method of Hunt and McIlroy [20]. A dynamic-programming table of size $(m+1) \times (n+1)$ is filled for the m and n lines of the two versions, and the table is then traced back to classify each line as unchanged, removed or added. The time and space cost is $O(mn)$, which is acceptable for Bills of a few hundred lines.

A new version may be saved only by the owner of the Bill's current stage. The server assigns the next version number, records the stage at which the version was saved and the change summary, and updates the Bill's current version. Versions are never edited or deleted. The comparison view runs this LCS algorithm in the browser on the text of the two chosen versions; as a safeguard, if the two texts would need a table of more than four million cells, it falls back to showing the old and new text in full. Figure 10 shows the comparison of the Agricultural Marketing Bill as introduced with the Act as enacted: the summary counts the lines added and removed, removed lines are struck through in red and added lines are shown in green, so a reader sees exactly which clauses changed without opening the two versions side by side.

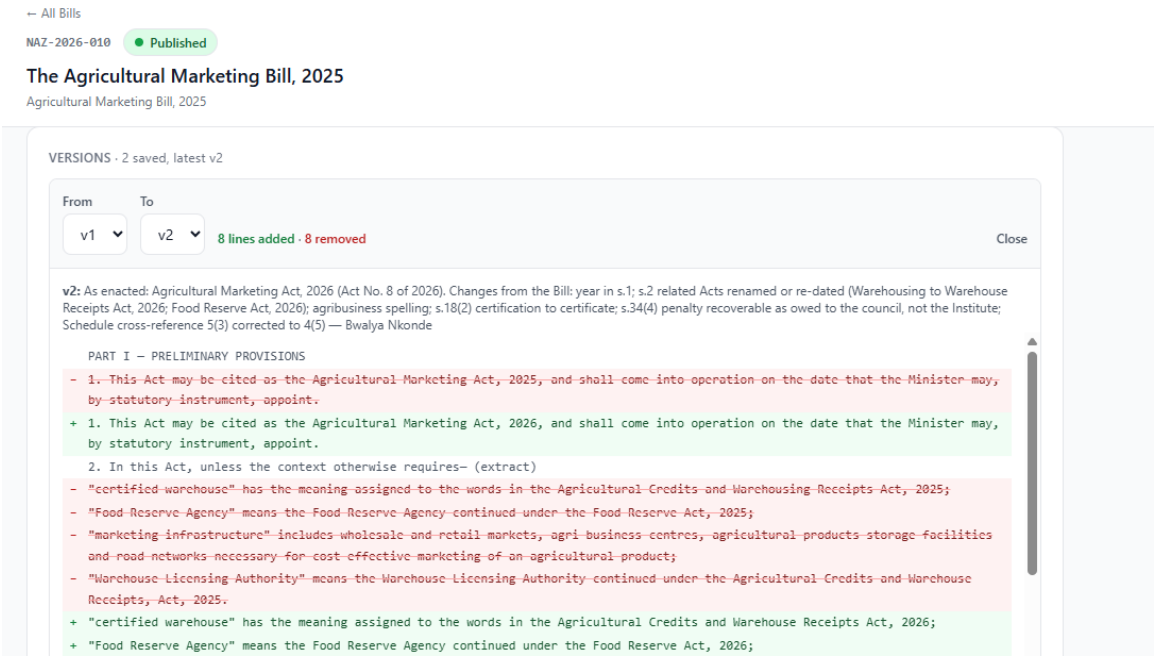


Figure 10: Comparison of version 1 (Bill as introduced) with version 2 (Act as enacted) (live deployment)

5.6 Amendments and Official Documents

Any signed-in officer may propose an amendment to a clause of a Bill that has not yet been published, with the proposed text, a rationale and, optionally, the version it amends. Only the owner of the Bill's current stage may accept or reject it, and the reviewer records the reasons for the decision (FR-020). The decision is final: an attempt to decide an amendment a second time is refused with *409 Conflict*, so the history is never rewritten (FR-019). Proposals and decisions both raise events, so they appear in the audit trail and notify the relevant users.

The official PDF of a Bill is uploaded as a multipart request. The server accepts the file only from the owner of the current stage, checks that it is no larger than 10 MB and that it actually starts with the PDF signature (not merely that it has a .pdf name), computes its SHA-256 checksum, and stores it in the database with the Bill and, optionally, the version it belongs to. Lists of documents never load the file contents.

5.7 Search, Reports and Statutory Periods

Search combines two PostgreSQL mechanisms [46], [47]. A generated *tsvector* column, weighted so that a match on the reference number ranks above a match on the short title, which ranks above a match on the long title, supports word search with stemming, so that "amendments" also finds "Amendment". Trigram indexes support matches on fragments and tolerate small spelling errors, so that "NAZ-2026" or "heal" also succeed. Both are served by GIN indexes, and the database maintains the search column itself, so it can never fall out of step with the data.

Search builds a query dynamically from optional criteria (keyword, stage, submitter, date range) and pages the results in a stable order. Special characters in the keyword are escaped before they are used in a pattern match, so that a search for "50%" does not match everything. Processing times are computed by a pure function that measures only completed visits to a stage (from entering it to entering the next stage), so a Bill that is still in a stage does not distort the averages. The statutory period panel on the Bill detail page displays the date computed by *BillDeadlinePolicy* together with its legal citation and an explanation of its effect.

5.8 Frontend Implementation

The frontend is a Next.js 14 application written in TypeScript and styled with Tailwind CSS. Shared TypeScript types describe every API response, a typed client wraps every endpoint, and TanStack Query caches server data and refreshes it after each change. Reusable components, such as the sidebar, notification feed, stage pipeline bar, stage badge, Bill card, search bar, versions panel and amendments panel, give every page the same structure.

The frontend has one route per screen. The figures below show the main screens.

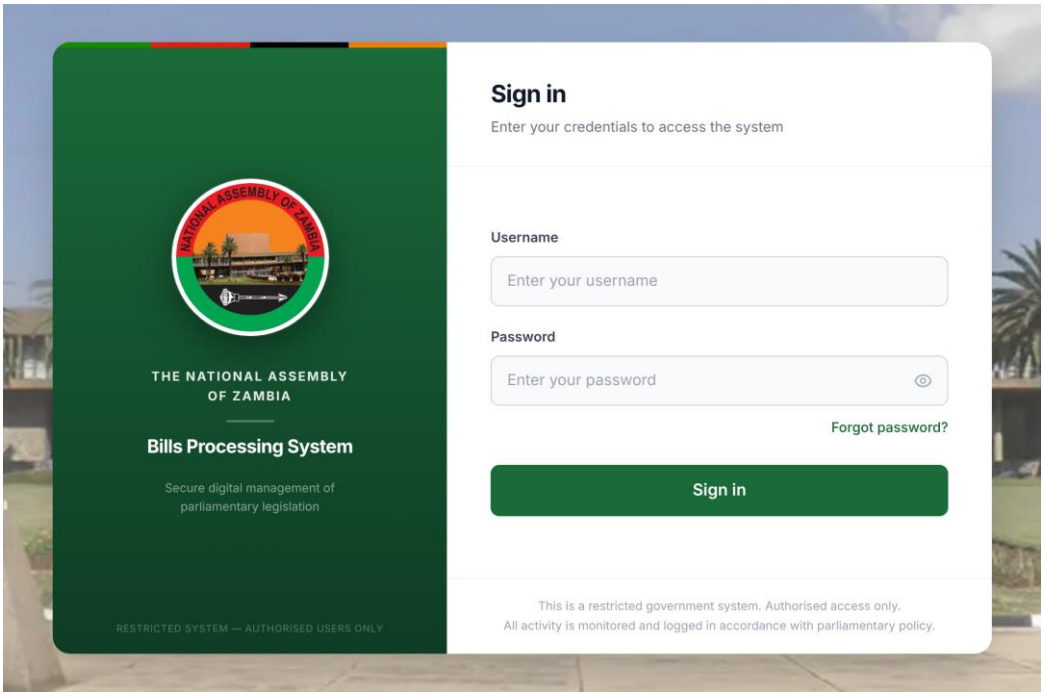


Figure 11: Sign-in page (demonstration data)

The sign-in page (Figure 11) authenticates the user and sends each role to its home screen. The remaining screens in this section were captured from the live deployment on 23 October 2026, after the user evaluation, and therefore include the practice Bills that were created for participants. The dashboard (Figure 12) shows the totals of all, active, overdue and published Bills and the active Bills arranged by stage. At the time of capture the system held 61 Bills; none was overdue, because no administrative targets are configured and a statutory period can never make a Bill overdue (Section 4.6.2).

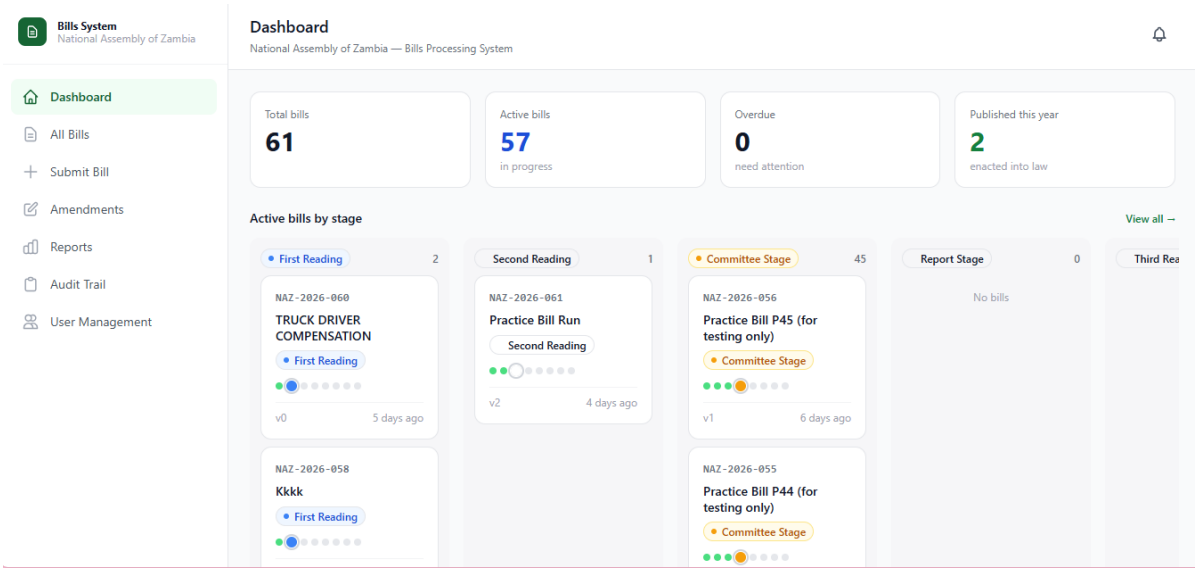


Figure 12: Dashboard on the live deployment, including practice Bills created for the user evaluation

The Bills page (Figure 13) lists Bills with their stage and supports keyword search, a stage filter and paging. Selecting a Bill opens the Bill detail page (Figure 14), which is the centre of the application. The example shown is the Agricultural Marketing Bill, 2025, which was loaded from its published sources and taken through every stage to publication as the Agricultural Marketing Act, 2026 (Act No. 8 of 2026) [13]. The pipeline shows that the Bill is published, the period panel states that no period is prescribed by law for that stage, and the action panel confirms that no further stages remain.

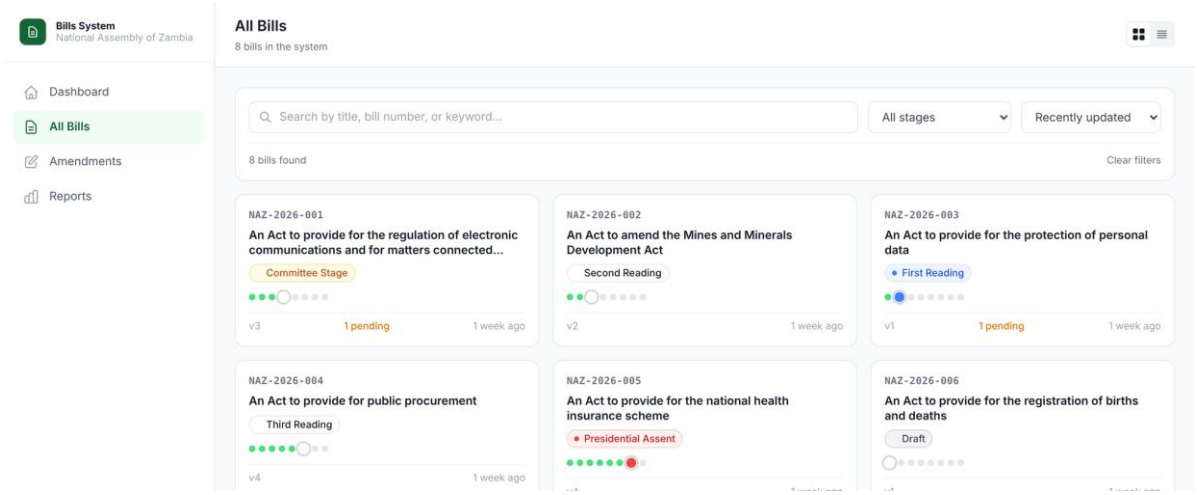


Figure 13: Bills list with search and stage filter (demonstration data)

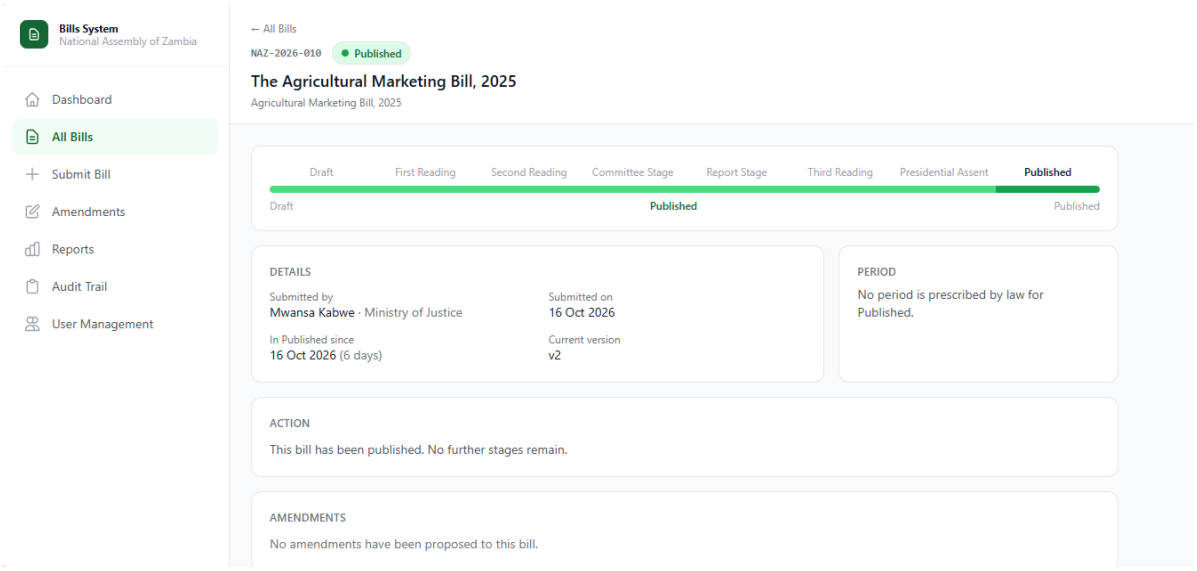


Figure 14: Bill detail page for a published Bill: stage pipeline, details, period panel and action panel (live deployment)

Further down the same page, the versions panel (Figure 15) lists every saved version with its change summary, author, office, time and stage, and links to the text, the changes from the previous version, the stored PDF and the public source. The stage history (Figure 16) records every transition with the officer, office, time and notes. Where the public record does not give the date of a stage, the note says so rather than inventing one.

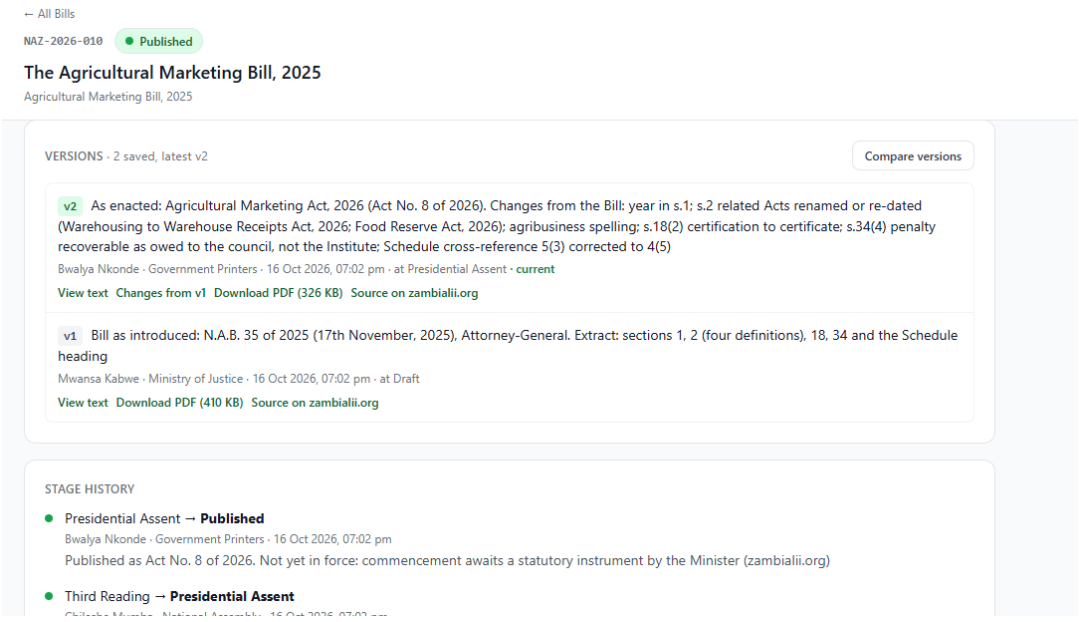


Figure 15: Versions panel: the Bill as introduced (v1) and as enacted (v2), with links to text, changes and PDFs (live deployment)

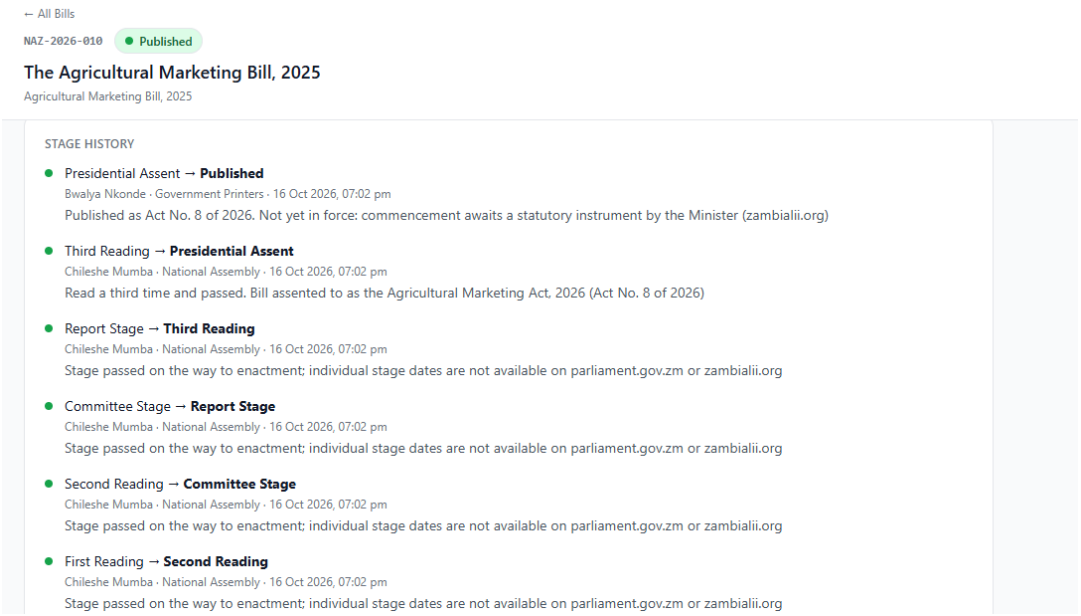


Figure 16: Stage history of the Agricultural Marketing Bill, with officer, office, time and notes for each transition (live deployment)

Amendments are proposed and decided on the Bill detail page and are also collected in a queue (Figure 17), with tabs for pending, accepted and rejected amendments. The Reports page shows the number of Bills in each stage (Figure 18) and the time Bills have spent in each stage (Figure 19), each exportable to PDF or Excel. The processing-time report places the legal period beside each stage and counts only completed passes; its times are close to zero here because the evaluation Bills were moved through their stages within minutes. The Audit Trail (Figure 20), available only to the Administrator, lists every recorded action, including failed sign-ins, and exports the full trail as CSV; by 23 October it held 468 entries.

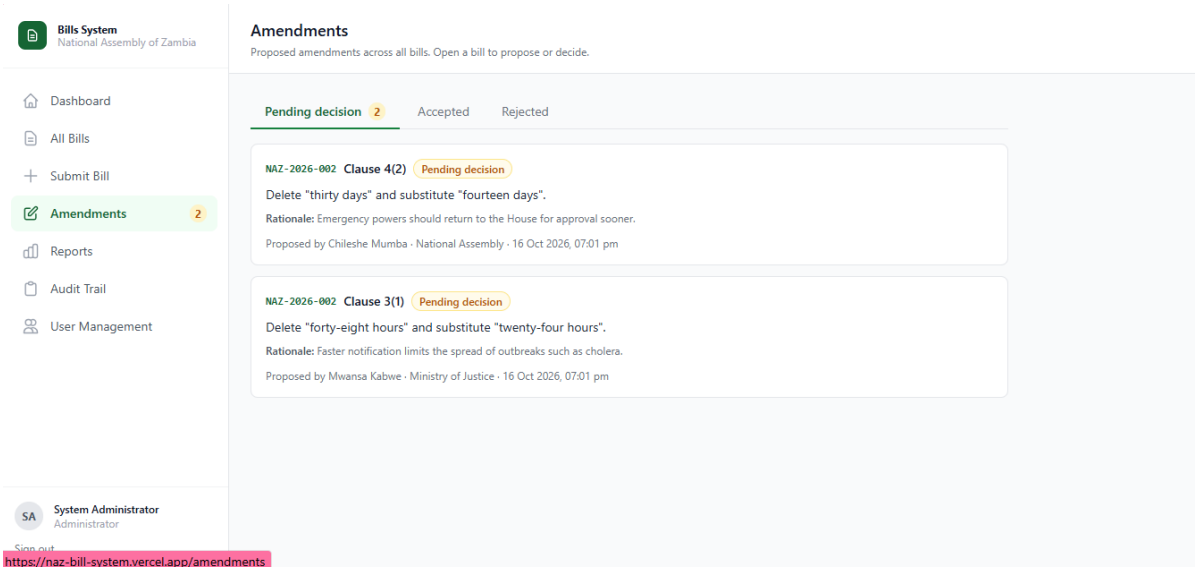


Figure 17: Amendments queue with two amendments pending decision (live deployment)

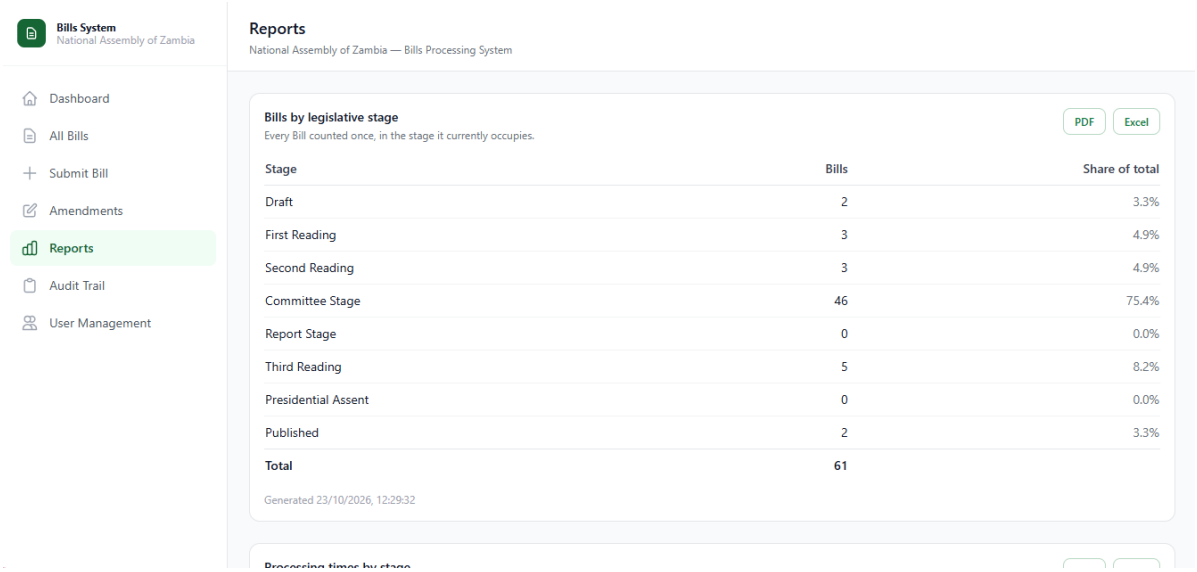


Figure 18: Reports page: Bills by legislative stage, with PDF and Excel export (live deployment)



Figure 19: Processing times by stage, with the period in law and its citation beside each stage (live deployment)

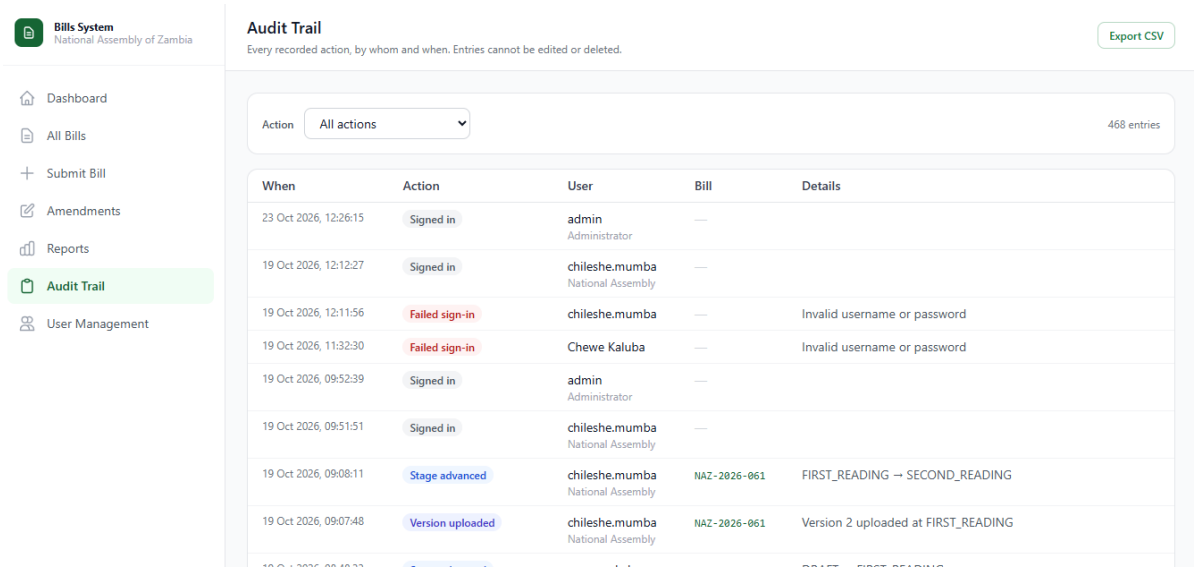


Figure 20: Audit trail with sign-ins, failed sign-ins, stage changes and version uploads, visible to the Administrator only (live deployment)

5.9 Integration, Deployment and Operations

The frontend and backend were developed in parallel against a shared contract. In Sprint 0 the frontend was built against a mock API, using Mock Service Worker and TypeScript types that described every response. As each backend module was completed, the frontend switched to the real endpoint, and the types were aligned with the actual DTOs ("Align backend API

contract with frontend types"). Cross-origin requests from the Vercel domains are allowed explicitly in the CORS configuration.

Integration was protected by continuous integration. Table 12 lists the GitHub Actions workflows. A pull request could be merged only when these checks passed. The frontend is deployed to Vercel automatically, while the backend is deployed by a manual workflow, because a deployment that runs database migrations is a deliberate decision, not a side effect of a merge [41].

Table 12: Continuous integration and deployment workflows

Workflow	Trigger	Steps
Frontend CI (ci.yml)	Every push and pull request	Install with npm ci; ESLint; TypeScript type-check; production build
Backend CI (backend-ci.yml)	Changes under backend/	Maven clean verify on JDK 21 (all JUnit tests); upload test reports; build Docker image; start container; smoke-test POST /auth/login
Deploy frontend (deploy-frontend.yml)	Push to main; manual	Build and deploy to Vercel production
Deploy backend (deploy-backend.yml)	Manual only	Require RDS snapshot confirmation; assume AWS role via OIDC; build and push image to ECR; run the new container on EC2 via SSM

5.10 Implementation Challenges and Solutions

Table 13 lists the main challenges the team met and how each was resolved.

Table 13: Challenges encountered and how they were solved

Challenge	Solution
The first JWT merge broke the shared branch and had to be reverted (PRs #3 to #5).	Restored on a fresh branch with the filter and security configuration tested together; adopted the rule that CI must pass before merging.
Response entities exposed internal fields, including the password hash of the user who created a version.	Introduced DTOs for every response (for example BillVersionResponse) so entities are never serialised directly.
Browser requests from Vercel were blocked by CORS, and some errors returned 403 instead of 404.	Listed the production origins explicitly and corrected the error handling.
The Vercel deploy workflow hung for up to six hours and blocked teammates' merges.	Fixed the workflow and deployed without git metadata, so that a teammate's merge could not block the deployment.
Statutory periods were at first modelled as "deadlines", which misstates the law (SO 113(1) is a minimum; SO 127(6) deems assent).	Remodelled each period with its unit, legal effect and citation; only administrative targets can be overdue.
Flyway migrations cannot be rolled back, so a bad migration in production would be permanent.	The deploy workflow refuses to run until an RDS snapshot is confirmed.
Search needed both whole-word matches and partial reference numbers.	Combined a weighted tsvector with trigram indexes (V5).
The abrupt closure of schools interrupted the team's in-person meetings.	Moved the biweekly Scrum meetings and supervisor reviews online, and resumed meeting in person when the university reopened.

Challenge	Solution
Time: comments, reminders and the amendment activity report could not all be finished.	Prioritised the critical features with the supervisor; deferred FR-022, FR-028 and FR-042, and froze implementation on 16 October for evaluation.

5.11 Team Contributions

The system was built jointly, with each member leading one architectural layer. Table 14 summarises the principal contribution of each member, based on the repository history; the full individual contribution statements are given in Appendix A.

Table 14: Principal contributions of each team member

Area	Chali Gerald Chibwana	Prince Daka	Mitchell Ndona
Project set-up	Spring Boot scaffold, users table	Backend CI workflow	Monorepo, README, Docker Compose, first CI and Vercel workflows
Backend	Auth, Bill model and workflow, versions, amendments, documents, notifications, audit, dashboard API	Search (V5), reports with PDF and Excel export	—
Frontend	Bill submission and detail, versions and comparison, amendments, audit pages	API client additions	Scaffold, types, mock API, layout and components, sign-in and Reports pages
Infrastructure	Dockerfile, CORS, Vercel deploy fixes	AWS EC2, RDS and ECR; OIDC/SSM deployment workflow	Vercel deployment workflow
Testing	39 test methods	85 test methods	Frontend lint and type checks
Evaluation	Real Bill data for the study	Live system during the study	Instrument, pilot, recruitment and analysis
Repository	60 commits	25 commits; reviewed and merged pull requests	10 commits; repository owner

CHAPTER SIX

TESTING, RESULTS AND EVALUATION

6.1 Introduction

This chapter presents the evidence from testing and evaluating the completed system. Testing and evaluation had two purposes. Verification asked whether the system was built correctly, that is, whether it satisfies its requirements and enforces the rules of the Bill process; validation asked whether it is the right system for its users, that is, whether people can use it and would accept it [39]. Section 6.2 describes the testing strategy and environment, Section 6.3 presents the test cases and their results, and Section 6.4 reports the performance and operational results observed on the live deployment. Section 6.5 presents the results of the user evaluation, Section 6.6 analyses the results, Section 6.7 evaluates each objective of the project against the evidence, and Section 6.8 identifies the results attributable to each member of the group. Section 6.9 summarises the findings, which are discussed in Chapter 7.

6.2Testing Strategy

Testing followed the V-model principle that each level of design is checked by a corresponding level of testing [39]:

- **Unit tests** verify the domain rules in isolation, without a database: the transition policy, statutory periods, search criteria and stage durations.
- **Web-layer (integration) tests** use Spring MockMvc with mocked repositories to call each endpoint as different roles and check status codes, response bodies and access rules, including that forbidden roles receive 403 and that responses never contain password hashes.
- **Build verification** in CI compiles the code, runs every test, builds the Docker image, starts the container and checks that it answers a sign-in request.
- **Manual system tests** with Postman and in the browser exercise complete scenarios on the deployed system, following the demonstration script.
- **User acceptance testing** checks that new users can complete the critical tasks and find the system useful and easy to use (Section 6.5).

All automated tests ran in the environment summarised in Table 15, on every pull request and on the final build before the implementation freeze.

Table 15: Test environment

Item	Configuration
CI runner	GitHub Actions, ubuntu-latest
Backend	JDK 21; Maven (mvn -B clean verify); JUnit 5, Mockito, Spring MockMvc
Database for tests	PostgreSQL 15 service container with all Flyway migrations applied
Container check	Docker build of the backend image, container started, POST /auth/login smoke test
Frontend checks	Node.js 22; ESLint; TypeScript type-check; production build
Manual testing	Postman collections; browsers on the live deployment (Vercel and AWS)

6.3 Test Cases and Results

6.3.1 Automated Test Suite

The automated suite contains 125 test methods in 15 JUnit test classes. Several are parameterised and run once per input, so the suite executes 158 test cases. Table 16 summarises the suite by class.

Table 16: Automated test suite by class

Test class	Tests	What is verified	Requirements
BillTransitionPolicyTest	6 (parameterised)	Every stage owner may act; other roles may not; no skipping; nothing leaves PUBLISHED	FR-009–012
BillDeadlinePolicyTest	13	SO 113(1) working-day date; Art. 66 21-day date; statutory stages never overdue; deemed assent; targets	FR-025–027
BillSearchCriteriaTest	8	Keyword, stage, submitter and date filters; escaping of special characters	FR-033–036
BillVersionTest	9	Only the stage owner saves versions; numbering; content and summary; ordering	FR-013–016
BillDocumentControllerTest	9	Owner-only upload; PDF signature check; 10 MB limit; checksum; download	FR-029–032
NotificationListenerTest	11	Correct recipients for each event; no notification to the actor	FR-023
NotificationControllerTest	8	Users see and mark only their own notifications	FR-023
AuditListenerTest	6	Each event produces the right audit entry	FR-037–038
AuditServiceTest	5	Actor snapshot; failed sign-ins recorded	FR-037–038
AuditControllerTest	10	Administrator only; filters; CSV export	FR-039–040
DashboardControllerTest	7	Counts by stage; active Bills; overdue only against administrative targets	FR-025–027
ReportControllerTest	12	Bills per stage; processing times; access	FR-041, 043
ReportExportTest	8	Valid PDF and Excel files with correct content	FR-044
StageDurationsTest	12	Only completed visits counted; averages; edge cases	FR-043
BackendApplicationTests	1	Application context starts	—

In the final CI run before the implementation freeze, on 16 October 2026, all 158 test cases passed (0 failures, 0 errors, 0 skipped) against a PostgreSQL 15 database, the build succeeded, and the Docker build and smoke test job also passed (Figure 21). Further test cases are listed in Appendix D.

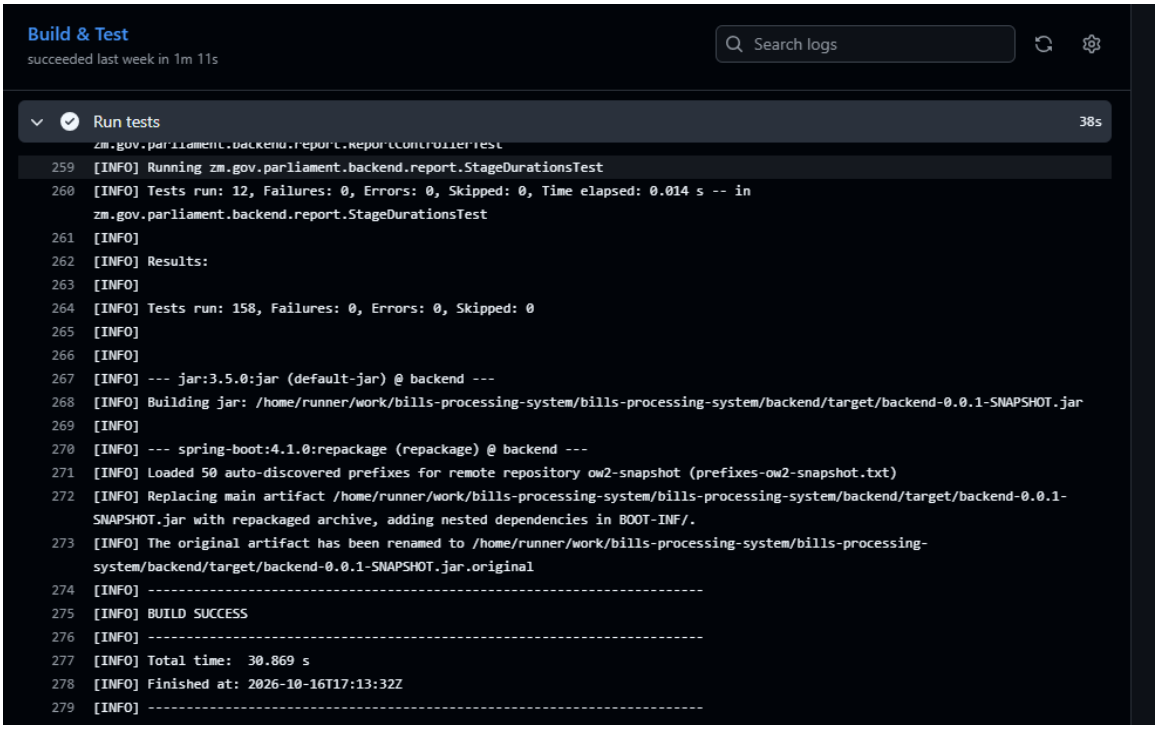


Figure 21: Backend CI run of 16 October 2026 on GitHub Actions: 158 tests run, 0 failures, 0 errors, BUILD SUCCESS

6.3.2 Representative System Test Cases

Table 17 lists representative system-level test cases derived from the requirements, with their results on the deployed system.

Table 17: Representative system test cases and results

ID	Test case	Expected result	Result
TC-01	Sign in with valid credentials for each role	Token issued; role home page shown	Pass
TC-02	Sign in with a wrong password	Rejected; LOGIN_FAILED recorded in audit trail	Pass
TC-03	Call an API endpoint without a token	401/403; no data returned	Pass
TC-04	MoJ officer submits a new Bill	Bill created in DRAFT with a reference of the form NAZ-2026-nnn; BILL_CREATED audit entry	Pass
TC-05	NA officer tries to submit a Bill	Refused (403)	Pass
TC-06	MoJ officer advances a Bill from DRAFT	Bill at FIRST_READING; history row; NA officers notified	Pass
TC-07	MoJ officer tries to advance a Bill at FIRST_READING	Refused (403): not the stage owner	Pass

ID	Test case	Expected result	Result
TC-08	Administrator requests DRAFT → SECOND_READING	Refused (409): stages cannot be skipped	Pass
TC-09	Stage owner saves a new version with a summary	Version n+1 saved; others notified	Pass
TC-10	Compare version 1 with version 2	Changed clauses shown as removed and added lines	Pass
TC-11	Propose an amendment; stage owner accepts it, then tries to decide it again	Accepted with reasons; second decision refused (409)	Pass
TC-12	Upload a non-PDF file renamed to .pdf	Rejected: not a PDF	Pass
TC-13	Search "heal" and filter by COMMITTEE_STAGE	Matching Bills at that stage only	Pass
TC-14	Bill enters COMMITTEE_STAGE	Panel shows earliest report date after 20 working days, citing SO 113(1)	Pass
TC-15	Export Bills per stage to PDF and Excel	Valid files downloaded with the same figures as on screen	Pass
TC-16	Attempt UPDATE on audit_log in SQL	Database raises "audit_log is append-only"	Pass
TC-17	Non-administrator opens the Audit page	Refused (403)	Pass

6.3.3 Functional Completeness Against Requirements

Of the 44 functional requirements in the SRS, 41 were implemented and are in use on the live system (Table 7). Three were deferred with the supervisor's agreement: comments on Bills (FR-022), scheduled reminders (FR-028) and the report on amendment activity (FR-042). All eleven modules are present, and the eight modules that carry the core workflow, namely authentication, registration, workflow, versions, amendments, documents, search and audit, are complete.

6.4 Performance and Operational Results

Formal load testing was outside the scope of the project. The operational results observed on the live deployment were:

- **Availability.** The frontend at naz-bill-system.vercel.app and the API on AWS served all 25 evaluation sessions, which took place between the evening of 18 October and the morning of 19 October 2026; no participant reported that the system was unavailable.
- **Responsiveness.** Participants worked mostly on mobile phones over ordinary connections, and one volunteered that they "liked the quick response time"; no participant reported slowness.
- **Volume and integrity.** By 23 October the system held 61 Bills and 468 audit entries (Figure 12, Figure 20). Every stage change, version upload and sign-in made during the study, including failed sign-ins, appears in the audit trail.

- **Legal periods.** The dashboard reported no overdue Bills although 46 Bills were in Committee Stage, as intended: SO 113(1) sets a minimum, not a deadline, and no administrative targets were configured (Figure 12).

6.5 User Evaluation

The user acceptance test was carried out on the live deployment from 18 to 19 October 2026. Participants opened the system from a link, signed in with the accounts provided, carried out three tasks and completed the questionnaire in Appendix E.

6.5.1 Participants

Twenty-five people completed the questionnaire, and all 25 gave consent. Twenty (80%) used a mobile phone and five (20%) a laptop or desktop computer. The questionnaire did not record programme or year of study, so device type is the only participant characteristic available for analysis. Participants were the team's classmates, junior students in the Department of Computing and Informatics and a few ICT students in the School of Education.

6.5.2 Task Completion

Figure 22 and Table 18 show how many participants completed each task. Overall, 68 of the 75 task attempts (90.7%) were completed, 49 of them (65.3%) easily.

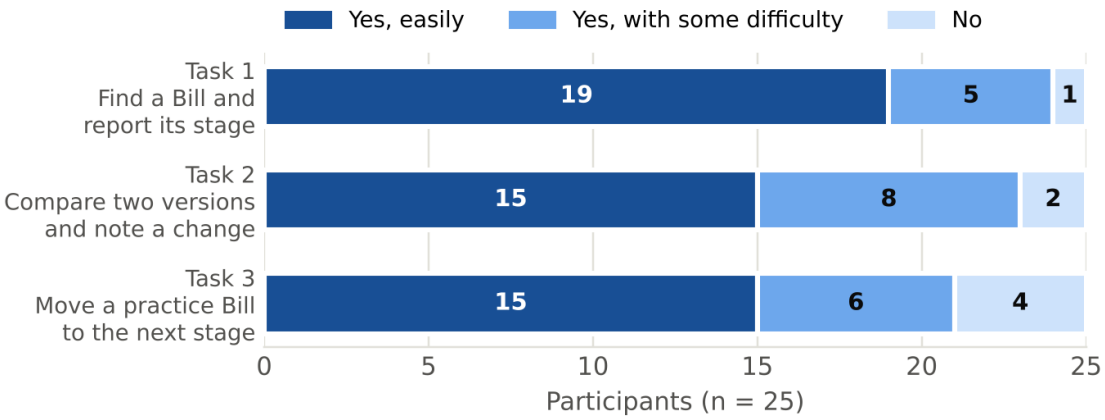


Figure 22: Self-reported task completion by task (n = 25)

Table 18: Task completion by task (n = 25)

Task	Completed easily	With some difficulty	Not completed	Completion rate
T1 Find a Bill and report its current stage	19	5	1	96%
T2 Compare two versions and write down one change	15	8	2	92%
T3 Move a practice Bill to the next stage	15	6	4	84%
All tasks (75 attempts)	49	19	7	90.7%

The written answers allow the first two tasks to be checked as well as self-reported. For T1, the correct answer was Second Reading, and 14 participants (56%) gave it. Six reported Committee Stage, which is most likely the stage they saw after advancing their practice Bill in T3, and five gave another answer or none. For T2, the team classified each written change against the actual differences between the two versions of the Agricultural Marketing Bill. Fourteen participants (56%) named a genuine change, such as the year in section 1, the renamed Warehouse Receipts Act or the corrected spelling of "agribusiness", and three of these quoted the version's change summary rather than the comparison itself. Three described the version only in general terms (for example "this was assented"), five gave an unrelated or incorrect answer, and three left the field blank. Self-reported completion (92%) is therefore higher than demonstrated accuracy (56%) for this task.

6.5.3 Usability (SUS)

Each participant's SUS score was computed in the standard way [28]: one is subtracted from each odd-numbered item, each even-numbered item is subtracted from five, and the sum is multiplied by 2.5. Figure 23 shows the 25 scores in order, and Table 19 summarises them.

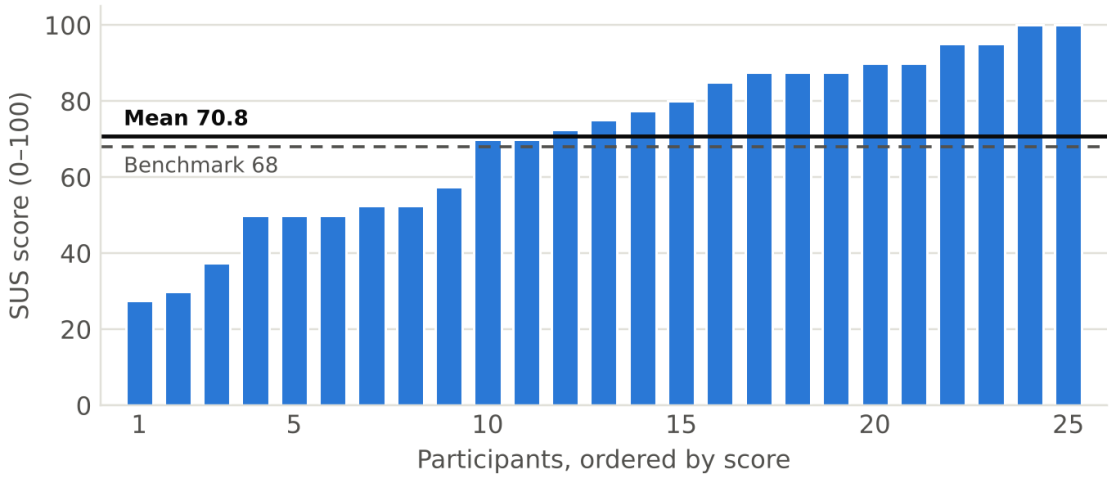


Figure 23: SUS score of each participant, in ascending order, with the mean and the benchmark of 68

Table 19: Summary of SUS scores (n = 25)

Statistic	Value
Mean (standard deviation)	70.8 (22.0)
95% confidence interval of the mean	61.7 to 79.9
Median (range)	75.0 (27.5 to 100)
Participants scoring 68 or more	16 of 25 (64%)
Participants scoring below 51	6 of 25 (24%)
Comparison with benchmark 68 (one-sample t-test)	t(24) = 0.64, p = 0.53
Interpretation	"Good"; acceptable range; grade C [29], [30], [49]

The mean SUS score of 70.8 is above the average of 68 across hundreds of published studies [29] and corresponds to "good" usability, just inside the acceptable range, on the scales of Bangor et al. [30], [49]. The confidence interval is wide, from 61.7 to 79.9, and includes 68, so with 25 participants the difference from the benchmark is not statistically significant. The distribution is bimodal: ten participants scored above 80, while six scored below 51. Table 20 shows the mean response to each item.

Table 20: Mean response to each SUS item (1 = strongly disagree, 5 = strongly agree; n = 25)

No.	Item	Mean	Desired
1	I think that I would like to use this system frequently.	3.96	High
2	I found the system unnecessarily complex.	1.96	Low
3	I thought the system was easy to use.	3.52	High
4	I think that I would need the support of a technical person to be able to use this system.	2.52	Low
5	I found the various functions in this system were well integrated.	3.92	High
6	I thought there was too much inconsistency in this system.	2.00	Low

No.	Item	Mean	Desired
7	I would imagine that most people would learn to use this system very quickly.	3.68	High
8	I found the system very cumbersome to use.	1.76	Low
9	I felt very confident using the system.	3.60	High
10	I needed to learn a lot of things before I could get going with this system.	2.12	Low

The strongest items were that the system is not cumbersome (1.76), not unnecessarily complex (1.96) and consistent (2.00), and that its functions are well integrated (3.92). The weakest were the need for support from a technical person (2.52) and ease of use (3.52). Laptop and desktop users scored higher (mean 83.0, n = 5) than phone users (67.8, n = 20), consistent with an interface designed for office computers, although the difference is not significant with groups this small (Mann–Whitney U, p = 0.16).

6.5.4 Acceptance (TAM)

Three further items measured the constructs of the Technology Acceptance Model [32] on the same five-point scale. The first two respondents left them blank, so 23 participants answered them (Table 21).

Table 21: Technology acceptance items (1 = strongly disagree, 5 = strongly agree; n = 23)

Construct	Item	Mean (SD)	Agree (4–5)
Perceived usefulness	Using this system would make processing Bills between offices easier.	4.22 (0.95)	19 of 23 (83%)
Perceived ease of use	Learning to use this system would be easy for me.	4.35 (0.93)	20 of 23 (87%)
Intention to use	If I worked in one of these offices, I would want to use this system.	4.35 (0.88)	19 of 23 (83%)

All three means are well above the scale midpoint of 3, and more than four in five participants agreed with each statement. Intention to use correlated moderately with the SUS score (r = 0.47), as TAM predicts: participants who found the system easier to use were more willing to use it.

6.5.5 Qualitative Feedback

Eight participants answered the optional question "Anything confusing, or anything you liked?". Their comments fell into two themes:

- **What participants liked (6 comments):** the version comparison ("the version comparison/highlights without having to open both versions side by side"; "I like how the

system's version control"), the speed ("I liked the quick response time"), easy access to information on Bills, and the system as a whole ("Loved the whole system, it's superb").

- **What made it harder (4 comments, two of them also positive):** unfamiliarity with the legislative domain and its language ("Liked the system but not familiar with bills so wasn't sure"; "it felt complex and technical in language"; "it requires more knowledge in order to use efficiently"; "Difficult to understand"). One more participant wrote in a task answer that the Bill named in the form did not seem to match the names in the system.

6.6 Analysis of Results

Verification and validation together give a consistent picture. The automated tests show that the rules most important to the institution hold: no role can act outside its stages, no stage can be skipped, statutory periods are computed as the law prescribes, and no action escapes the audit trail. These are exactly the properties that the problem statement found missing from the manual process.

The user evaluation shows that first-time users, with no training, could complete the core tasks: 90.7% of task attempts succeeded, finding a Bill was the easiest task (96%) and moving a Bill between stages the hardest (84%). The SUS score of 70.8 places the system slightly above average and in the "good" band, and the acceptance scores are high (4.22 to 4.35 out of 5). Read together, the quantitative and qualitative results suggest that the main obstacle for weaker scorers was not the interface itself but the legislative domain. The items on needing a technical person and learning a lot before starting were the weakest even-numbered items; four of the eight comments concerned unfamiliar legislative language; and the T2 answers show that some participants could open the comparison without being sure what a "change" to a Bill meant. Officers of the Ministry of Justice and the National Assembly, who work with Bills daily, should face this obstacle to a lesser degree, so these results may understate usability for the intended users. Phone users scored lower than computer users, which points to the mobile layout as a practical improvement.

Three cautions apply. The sample of 25 is below the target of 30, and the confidence interval of the SUS mean includes the benchmark. The participants were proxy users, not Assembly staff. And task completion was self-reported; the written answers show that it overstates accuracy (T1: 96% reported completion, 56% correct stage; T2: 92% and 56%).

The testing itself also had limits, which bound how far the verification results can be relied upon:

- **Frontend unit tests.** The frontend pipeline lints, type-checks and builds the application, but the frontend has no unit tests of its own; its behaviour was tested manually and through the user evaluation.
- **No load testing.** Performance was observed during the study rather than measured under load.
- **Database trigger.** The append-only trigger was tested directly against the database (TC-16) rather than by an automated test in the suite.
- **Mocked repositories.** Most web-layer tests mock the repositories, so database behaviour is covered by the start-up test and by manual testing rather than by integration tests of every query.

6.7 Objective-by-Objective Evaluation

Table 22 evaluates each specific objective of Section 1.3.2 against the evidence in this report.

Table 22: Evaluation of each objective

Objective	Evidence	Status
O1 Analyse the Bill process and specify requirements	SRS with 44 FRs in 11 modules and NFRs; stage model reviewed by NAZ co-supervisors (Sections 3.4, 4.2–4.4)	Achieved
O2 Implement the eight-stage workflow with ownership	State machine and RBAC policy; TC-04 to TC-08; BillTransitionPolicyTest; 84% of participants moved a Bill between stages (Sections 4.6, 5.2–5.7, 7.2)	Achieved
O3 Version control, comparison and amendments	Append-only versions, LCS comparison, final amendment decisions; TC-09 to TC-11; 92% completed the comparison task	Achieved
O4 Search, notifications, statutory periods, reports, audit	All implemented with tests and in use on the live system; FR-022, FR-028, FR-042 deferred	Largely achieved
O5 Deploy through an automated pipeline	Vercel and AWS deployment; CI with tests and smoke test; OIDC deployment (Section 5.9)	Achieved
O6 Verify and validate	125 test methods (158 cases) all passing; UAT with 25 participants; SUS 70.8; TAM means 4.22–4.35 (Sections 6.3, 7.2)	Achieved (sample below target)

6.8 Individual Results

The system was built jointly, with each member leading one architectural layer (Section 5.11 and Appendix A). This section identifies the results that are directly attributable to the work of each member. The single application start-up test (BackendApplicationTests) was part of the shared project set-up.

6.8.1 Chali Gerald Chibwana (Backend Lead)

- **Automated tests.** Wrote 39 of the 125 test methods (AuditControllerTest, AuditListenerTest, AuditServiceTest, BillVersionTest and BillDocumentControllerTest), all of which passed in the final CI run of 16 October 2026.
- **System test cases.** The modules he implemented (authentication, the Bill workflow and statutory periods, versions, amendments, documents and the audit trail) are exercised by TC-01 to TC-12, TC-14, TC-16 and TC-17, all of which passed (Table 17).
- **User evaluation.** Task 2 (version comparison) and Task 3 (advancing a Bill) used the workflow and comparison features and the Bill detail screens he built; they were completed by 92% and 84% of participants respectively, and the version comparison was the feature most often named in participants' positive comments. He also prepared the real Bill data, including both versions of the Agricultural Marketing Bill, used in the study.
- **Operational evidence.** The audit module recorded 468 entries by 23 October 2026, including every sign-in, failed sign-in, stage change and version upload made during the evaluation.

6.8.2 Prince Daka (DevOps and Testing Lead)

- **Automated tests.** Wrote 85 of the 125 test methods across nine classes (BillDeadlinePolicyTest, BillSearchCriteriaTest, BillTransitionPolicyTest, DashboardControllerTest, NotificationControllerTest, NotificationListenerTest, ReportControllerTest, ReportExportTest and StageDurationsTest), all of which passed.
- **Pipeline and deployment.** Built the backend CI pipeline that produced the final run of 158 passing test cases and the Docker smoke test (Figure 21), and the AWS deployment that served all 25 evaluation sessions without any reported unavailability (Section 6.4).
- **Search and reports.** His search and reporting modules passed TC-13 (search with stage filter) and TC-15 (export to PDF and Excel), and search supported Task 1, which 96% of participants reported completing.

6.8.3 Mitchell Ndona (Frontend Lead)

- **User evaluation.** Designed the evaluation instrument (three tasks, the SUS and three TAM items), piloted it with three people, recruited the participants and analysed the responses, producing the results in Section 6.5: 25 participants, 90.7% of task attempts completed, a mean SUS score of 70.8 and TAM means of 4.22 to 4.35.

- **Frontend foundation.** Built the layout and reusable components, including the search bar, stage pipeline bar and Bill card used throughout the evaluation tasks, together with the sign-in page (TC-01) and the Reports page with PDF and Excel export (TC-15; Figures 18 and 19).
- **Frontend checks.** Set up the first CI workflow, whose frontend lint and type-check steps run on every push and pull request (Table 12).

6.9 Chapter Summary

Verification showed that the system was built as specified. All 158 executed test cases from the 125 test methods passed, the Docker smoke test passed, and every representative system test case (TC-01 to TC-17) passed, including the database-level protection of the audit trail. Forty-one of the 44 functional requirements were implemented, with comments (FR-022), scheduled reminders (FR-028) and the report on amendment activity (FR-042) deferred. The system ran publicly throughout the evaluation and recorded every action in its audit trail.

Validation showed that first-time users could use the system. The 25 participants completed 90.7% of task attempts, although the written answers show that only 56% reported the correct stage in Task 1 and named a genuine change in Task 2. The mean SUS score was 70.8 (95% confidence interval 61.7 to 79.9), above the benchmark of 68 but not significantly different from it, and perceived usefulness, perceived ease of use and intention to use were rated between 4.22 and 4.35 out of 5. Objectives 1, 2, 3 and 5 were achieved, Objective 4 was largely achieved and Objective 6 was achieved with a sample below the target. Chapter 7 discusses what these results mean.

CHAPTER SEVEN

DISCUSSION

7.1 Introduction

Chapter 6 presented what was observed: the results of the automated and system tests, the operational record of the live deployment and the results of the user acceptance test. This chapter discusses what those results mean. Section 7.2 interprets the findings in relation to the problem stated in Chapter 1 and to the objectives and research questions of the project. Section 7.3 relates the system to the existing solutions reviewed in Chapter 2. Section 7.4 sets out the contributions of the project, and Section 7.5 discusses the limitations of the work and their implications for how the results should be read. Section 7.6 summarises the discussion. The conclusions drawn from it, and the recommendations that follow, are presented in Chapter 8.

7.2 Interpretation of Findings

The problem addressed by the project was that a Bill moves between the Ministry of Justice, the National Assembly and the Government Printer on paper and by e-mail, with three consequences: there is no shared view of where a Bill is, no authoritative history of its text, and no reliable record of who did what and when. The findings are interpreted below against each of these, and then against the question of whether the intended users would find such a system usable and acceptable. Each subsection indicates the research questions and objectives to which it relates.

7.2.1 A Single, Enforced Record of Each Bill (RQ1, RQ2; O1, O2)

The analysis of the Bill process (RQ1) showed that it is a strictly sequential workflow of eight stages, each owned by exactly one of three offices. Its most consequential finding was negative: the Constitution and the Standing Orders prescribe a period for only two of the eight stages, and those two periods have opposite legal effects. Standing Order 113(1) sets a minimum of twenty working days before a committee may report, while Article 66(1) of the Constitution, read with Standing Order 127, sets a maximum of twenty-one days for assent, after which assent is deemed to have been given [1], [2]. A system that attached a "deadline" to every stage, as an ordinary tracking tool would, would therefore have misstated the law. The behaviour of the live system confirms that the period model works as intended: when the

dashboard was captured on 23 October 2026, 46 Bills were in Committee Stage and none was reported as overdue, because a minimum period cannot be breached by waiting. This is the correct legal outcome rather than a missing feature.

The verification results answer RQ2. The transition tests and the system test cases on stage changes show that no role can act outside the stages it owns, that no stage can be skipped, even by the Administrator, and that a published Bill cannot be moved. Before the system, the order of stages and the responsibility of each office depended on officers following procedure; in the system they are enforced on the server, where the interface cannot bypass them, which is the measure the OWASP guidance on broken access control calls for [12]. Placing the rule in a single policy class, consulted by every action that changes a Bill, proved sufficient: the same rule governs stage changes, new versions, amendment decisions, document uploads and the choice of who is notified.

The user evaluation adds a qualification. Ninety-six per cent of participants reported that they had found the named Bill, but only 56% then reported its correct stage, and six of the eleven who did not gave Committee Stage, most likely the stage that their own practice Bill reached in Task 3. The shared record was therefore found easily, but a new user could confuse the Bill being examined with another one. For officers who handle a known set of Bills daily this risk is likely to be smaller, but the finding shows that the identity of the Bill on screen matters as much as the presence of its stage.

7.2.2 A Tamper-Evident History and Accountability (RQ3; O3, O4)

The second and third weaknesses of the manual process concerned the history of the text and the record of decisions. The results show that the system keeps that history in a form that can be added to but not rewritten. Versions are append-only, an amendment decision cannot be made a second time (test case TC-11), every stored PDF carries a SHA-256 checksum, and the database itself rejects any attempt to update or delete an audit entry (TC-16). By 23 October 2026 the audit trail held 468 entries, including the sign-ins, failed sign-ins, stage changes and version uploads made during the evaluation, so the evaluation was itself recorded by the mechanism it was testing. The answer to RQ3 is therefore that completeness comes from recording each action in the same transaction as the action itself, and tamper evidence from

enforcing immutability in the database rather than only in the application, because the threat includes future code and direct database access, not only users of the interface.

The comparison of versions, the feature that most directly addresses the lack of an authoritative history, was also the feature participants praised most often. Yet only 56% of participants named a genuine change between the Agricultural Marketing Bill as introduced and the Act as enacted, against 92% who reported completing the task, and three of those who did so quoted the version's change summary rather than the comparison. The comparison itself displayed the changes correctly; what varied was the participants' ability to recognise which differences in a Bill amount to a change. This points to the legislative domain, rather than the comparison view, as the main source of difficulty, a reading that the questionnaire results in Section 7.2.3 support.

One gap remains in this area. The report on amendment activity (FR-042) was deferred, together with comments on Bills (FR-022) and scheduled reminders (FR-028). Every amendment, its status and its reasons are recorded, so the gap lies in summarising the record rather than in keeping it, which is why Objective 4 is regarded as largely rather than fully achieved.

7.2.3 Usability and Acceptance (RQ4; O6)

The mean SUS score of 70.8 lies above the average of 68 reported across published studies [29] just inside the acceptable range of Bangor et al. [30], and close to the mean score that Bangor et al. associate with the adjective "good" [49]. The defensible interpretation is nevertheless a cautious one. The 95% confidence interval, from 61.7 to 79.9, includes 68, and the difference from the benchmark is not significant ($t(24) = 0.64$, $p = 0.53$). The results therefore show that first-time users rated the system at about the level of a typical system; they neither demonstrate that it is better than average nor suggest that it is poor.

The mean also conceals two groups. Ten participants scored above 80, while six scored below 51. The low scores do not, however, amount to a rejection of the system: four of the six participants who scored below 51 still agreed that it would make processing Bills between offices easier, and four said they would want to use it if they worked in one of these offices. The item-level results explain the pattern. Participants did not find the system cumbersome (mean 1.76), unnecessarily complex (1.96) or inconsistent (2.00), and agreed that its functions were well integrated (3.92). The weakest items were the need for support from a technical

person (2.52) and whether the system was easy to use (3.52). In other words, participants judged the structure of the interface to be coherent but were less confident in using it, which is what would be expected of people working in an unfamiliar domain. Four of the eight written comments made this point directly, describing the language of Bills as unfamiliar or technical.

The acceptance results reinforce this reading. Perceived usefulness (mean 4.22), perceived ease of use (4.35) and intention to use (4.35) were all high, with more than four in five participants agreeing with each statement. Intention to use correlated moderately with the SUS score ($r = 0.47$, $p = 0.03$, $n = 23$), the relationship between ease of use and intention that the Technology Acceptance Model predicts [32]. The perceived ease of use item, which asked whether learning to use the system would be easy, was rated well above the SUS item on whether the system was easy to use (4.35 against 3.52). Participants, in other words, expected to become proficient with practice even where their first session was not easy, which is consistent with a barrier of unfamiliarity rather than of design.

Finally, participants who used a laptop or desktop scored higher than those who used a phone (83.0 against 67.8). With only five computer users the difference is not significant (Mann–Whitney U, $p = 0.16$), so it indicates a direction rather than an effect. The direction is, however, consistent with an interface designed for office computers, and since 80% of participants used phones while officers at the Assembly and the Ministry would use office computers, the overall score is more likely to understate than to overstate usability in the intended setting. The answer to RQ4 is therefore that new users found the system usable at an acceptable level and clearly useful, with the main obstacle being the legislative domain rather than the interface; whether the intended users find it easier, as these results suggest, remains to be confirmed (Section 7.5).

7.2.4 Delivery and Verification (O5, O6)

Forty-one of the 44 functional requirements were implemented, including all of the eight modules that carry the core workflow. All 158 executed test cases from the 125 test methods passed, every pull request was checked by the continuous integration pipeline, and the backend could be redeployed only after an RDS snapshot was confirmed. The system remained publicly available throughout the evaluation window, and no participant reported that it was slow or unavailable. These results mean that a three-person team was able to change a live system safely within a fourteen-week schedule, and that the rules that matter most to the institution

are protected against being broken by later changes. They do not mean that every part of the system was verified to the same depth; the uneven coverage of the verification is discussed in Section 7.5.

Read against the objectives, the findings indicate that Objectives 1, 2, 3 and 5 were achieved, Objective 4 was largely achieved, and Objective 6 was achieved with a sample below the target size. The evidence for each objective is set out in the objective-by-objective evaluation in Chapter 6.

7.3 Comparison with Existing Work

Compared with the public tracking services reviewed in Section 2.2, the UK Bills site [5] and Congress.gov [6], this system supports the internal work that produces the public record rather than only publishing it: it decides who may act, records the act and informs the next office. Compared with Bungeni [7], it is less configurable, since its stages are fixed in code rather than in a workflow definition, but it is built on current, maintained frameworks, is far smaller to operate, and encodes the Zambian statutory periods with their legal effect, which Bungeni does not. Compared with LEOS [16], it does not offer structured legislative drafting, but it tracks the whole journey of a Bill across institutions, which LEOS does not attempt. Compared with the sibling Order Papers system at NAZ [11], the two projects share a similar architecture and a permission-based lifecycle; this project differs in its emphasis on version history, immutable audit and statutory periods, while the Order Papers system emphasises AI-assisted duplicate detection. The two systems are complementary and could share authentication and audit services in a future integrated platform.

7.4 Contributions of the Project

The project contributes at four levels: to the National Assembly in practice, to the design of legislative information systems technically, to knowledge about how such systems are received by users, and to the University and the Assembly's wider digitisation effort.

Practical contribution. The project delivered a working, publicly deployed system that tracks a Government Bill from drafting at the Ministry of Justice to publication by the Government Printer, with role-based ownership of stages, version control and comparison, amendments, official documents, search, notifications, statutory-period tracking, reports and an audit trail.

It was demonstrated with real Bills, notably the Agricultural Marketing Bill, 2025 and the resulting Act No. 8 of 2026, rather than invented examples, and it is accompanied by a user manual and deployment documentation (Appendix B). It gives the Assembly a system that is ready for a supervised pilot rather than a design that would still have to be built.

Technical contributions. Three design patterns developed in the project are reusable beyond it:

- **A single policy for authorisation and stage order.** The finite state machine of the Bill lifecycle and the RBAC rule are held in one class and consulted by every action, so the rule "only the owner of the current stage may act, one stage at a time" is stated once and enforced everywhere [17], [19].
- **A pattern for trustworthy records.** Audit entries are written in the same transaction as the action they record, notifications are sent only after it commits, and immutability is enforced by the database itself, with SHA-256 checksums on stored documents.
- **A legally faithful model of statutory periods.** Each period is held with its unit (working or calendar days), its legal effect (a minimum, or deemed completion on expiry) and its citation, and is kept separate from administrative targets, so the system never presents an internal target as a legal deadline. The model can be reused by other legislative systems in Zambia and in parliaments with similar rules.

In addition, the project leaves an engineering baseline for the Assembly: requirements traceable to code and tests through their FR numbers, an automated test suite, versioned database migrations and a CI/CD pipeline that deploys to the cloud without stored credentials.

Research contributions. Following design science research, the project contributes an evaluated artefact together with knowledge from its evaluation [35]:

- **Evidence on the usability and acceptance of a legislative workflow system.** Untrained users completed 90.7% of task attempts, rated usability at about the level of an average system (SUS 70.8) and acceptance highly (4.22 to 4.35 out of 5).
- **The finding that the domain, not the interface, was the main barrier.** The item-level SUS results, the gap between expected and experienced ease of use, the written comments and the accuracy of task answers all point the same way, which matters for how such systems are introduced to new staff.

- **A methodological observation.** Self-reported task completion substantially overstated demonstrated accuracy (96% against 56% for Task 1, and 92% against 56% for Task 2), so evaluations of legislative systems should verify task outcomes rather than rely on self-report.
- **A short evaluation instrument.** The five-minute instrument of three tasks, the ten SUS items and three TAM items (Appendix E) can be reused with busy parliamentary staff.

Other contributions. The system complements the sibling Order Papers project at the Assembly [11], and together the two show that the Assembly's core business processes can be digitised with shared architecture and practices. The project also supports the direction set by the Electronic Government Act of 2021 for the use of information and communication technology in public institutions [8].

7.5 Limitations and Implications

The results of the project must be read in the light of its limitations. Each limitation is stated below together with its implication for interpreting the findings.

- **Proxy users.** The participants were students rather than officers of the National Assembly or the Ministry of Justice. The results therefore measure learnability and perceived usefulness for people new to the legislative process; they neither confirm nor rule out acceptance by officers in their daily work. The expectation that officers would find the system easier, because they know the domain, is an inference from the evidence rather than a finding.
- **Sample size and precision.** Twenty-five people took part, below the target of 30, and the confidence interval of the SUS mean includes the benchmark of 68. The SUS result cannot be claimed to exceed the benchmark, and comparisons between sub-groups, such as phone and computer users, lack the power to show more than a direction.
- **Composition of the sample.** Most participants used phones (20 of 25), and all were recruited by the team from among classmates and other students who may have known the developers. Device results are therefore indicative only, and positive ratings, especially on the acceptance items, may be inflated by a wish to support the team.
- **Self-reported task completion.** Completion was reported by participants, and the written answers show that it overstates accuracy. The completion rates should be read as upper bounds; for Task 3, which had no written answer, completion could not be checked at all.

- **Fixed task order and shared practice data.** All participants performed the tasks in the same order and on shared data, so effects of the order of tasks cannot be separated from effects of the interface. Some errors in Task 1 appear to be artefacts of the procedure, since participants appear to have reported the stage of their own practice Bill, and so understate how well search and Bill detail work in normal use.
- **Single-item acceptance measures.** Each TAM construct was measured with one item, so the reliability of the measures cannot be estimated. The acceptance results are indicative and should be confirmed with the full TAM instrument or SUMI [31], [32].
- **Uneven depth of verification.** The frontend has no unit tests of its own, most web-layer tests mock the repositories, the append-only trigger was tested manually rather than in the automated suite, and no load testing was done. Confidence is therefore highest in the business rules, lower in the behaviour of database queries and the user interface, and limited for performance, which rests on observation of 25 sessions held between the evening of 18 October and the morning of 19 October 2026 rather than on measurement under load.
- **Demonstration data.** Only published Bills and practice Bills were used, and the practice Bills were moved through their stages within minutes. The correctness of the processing-time report is supported by tests, but its usefulness on real stage durations has not been evaluated, and the handling of confidential drafts, with the security review that would require, was not tested.
- **Deferred requirements and working days.** Comments (FR-022), scheduled reminders (FR-028) and the report on amendment activity (FR-042) were not implemented, and public holidays are not yet excluded from the SO 113(1) calculation. Objective 4 is therefore only largely achieved, and the earliest committee report dates shown by the system are indicative rather than exact.
- **Single-instance hosting.** The API runs on one EC2 instance in one region. Its availability during the evaluation does not demonstrate the availability that production use would require.

Taken together, these limitations do not undermine the central technical finding of the project, that the design enforces the order of stages, the ownership of each stage and the immutability of the record, because that finding rests on verification rather than on the user study. They do

limit what can be claimed about usability and acceptance in daily use at the Assembly, which the present results indicate but cannot establish.

7.6 Chapter Summary

This chapter has interpreted the results of the project in relation to its problem, objectives and research questions. The analysis showed that the Bill process is a strictly sequential, role-owned workflow in which only two stages carry legal periods, with opposite effects, and the verification results show that the system enforces this workflow on the server and keeps a history of each Bill that can be added to but not rewritten. The user evaluation showed that first-time users could complete the core tasks and rated the system at about the level of an average system for usability and highly for usefulness and intention to use, with unfamiliarity with the legislative domain, rather than the interface, as the main difficulty. Compared with existing systems, the project supports the internal hand-offs that public tracking services only report, and encodes the Zambian statutory periods with their legal effect. Its contributions are a deployed system ready for a pilot, reusable patterns for authorisation, trustworthy records and statutory periods, and evidence on how such a system is received by new users. Its main limitations, chiefly the use of a small sample of proxy users and self-reported completion, mean that its usability and acceptance in daily use at the National Assembly remain to be confirmed. Chapter 8 draws the conclusions of the project and sets out recommendations and future work.

CHAPTER EIGHT

CONCLUSION AND RECOMMENDATIONS

8.1 Conclusion

The project set out to replace the manual, paper- and email-based tracking of Bills at the National Assembly of Zambia with a single, secure and auditable digital record. Using design science research and Scrum, the team analysed the Bill process with the National Assembly co-supervisors, specified 44 functional requirements, and designed and built a three-tier web system: a Next.js client on Vercel, a Spring Boot API in Docker on AWS EC2, and a PostgreSQL database on Amazon RDS. The system enforces the eight-stage lifecycle with role-based ownership, keeps every version and amendment decision, stores official PDFs with checksums, supports search, notifications, reports and statutory-period tracking, and records every action in an append-only audit trail. It was verified with 125 automated tests in a CI/CD pipeline and validated through a user acceptance test in which 25 participants completed 90.7% of the task attempts and gave a mean SUS score of 70.8.

The team draws the following conclusions:

- A role-based workflow system can make Bill processing at NAZ traceable: at any moment, an authorised officer can see where every Bill is, which text is current, what has been changed and by whom.
- Encoding legal rules in code requires legal precision. Modelling statutory periods with their legal effect and citation avoided misrepresenting the Standing Orders and the Constitution.
- Integrity is best guaranteed at the lowest layer that can enforce it: in the transaction for audit records and in the database for immutability.
- Automated testing and continuous delivery allowed a three-person team to change a live system safely within a fourteen-week schedule.
- First-time users found the system usable and useful (SUS 70.8; usefulness 4.22 and intention to use 4.35 out of 5), which supports proceeding to a pilot with National Assembly staff.

Objectives O1, O2, O3, O5 and O6 were achieved, and objective O4 was largely achieved, with three requirements deferred (Section 7.7). The aim of designing, implementing and

evaluating a secure web-based system that gives each Bill a single, traceable record from drafting to publication has therefore been met.

8.2 Recommendations for the NAZ Handover

1. **Pilot at NAZ.** Run a supervised pilot with officers of the Table Office, a committee and the Ministry of Justice on a small number of live Bills, and repeat the evaluation with the full TAM or SUMI questionnaire.
2. **Security review before production use.** Commission a security assessment, enable multi-factor authentication, and complete a data-protection impact assessment under the Data Protection Act [9].
3. **Host locally or in-region.** Consider hosting in a Zambian or African data centre, or the Government's own infrastructure, in line with the Electronic Government Act [8].
4. **Help new users with the legislative domain.** Add short in-app explanations of each stage and of terms such as "assent" and "Committee Stage", since unfamiliar language was the main difficulty reported in the evaluation.
5. **Improve the phone layout.** Phone users scored lower than computer users (67.8 against 83.0), so the version comparison and Bill detail pages should be adapted for small screens.
6. **Maintain a holiday calendar.** Have the National Assembly maintain the gazetted public holidays in the system, so that working-day periods are exact.
7. **Assign ownership.** Assign NAZ ICT staff to own the system, the pipeline and the AWS account, using the README, deployment runbook and this report as hand-over documentation.

8.3 Future Work

- Implement the deferred requirements: comments on Bills (FR-022), scheduled reminders ahead of the assent period (FR-028) and the report on amendment activity (FR-042), and a user-management screen.
- Apply deemed assent automatically when the 21-day period under SO 127(6) expires, with a recorded system action.
- Export Bills in the Akoma Ntoso format [15] and publish Bill status as open data on the Parliament website.
- Add e-mail or SMS notifications and integrate with the Government Gazette for publication.

- Move document storage to object storage (Amazon S3) and add high availability and load testing.
- Integrate with the Order Papers system [11] so that Bills appear on the Order Paper automatically when they are scheduled for a reading.
- Extend the workflow to Private Members' Bills, which follow a different initiation route.

REFERENCES

[1] Government of the Republic of Zambia, Constitution of Zambia (Amendment) Act No. 2 of 2016. Lusaka, Zambia: Government Printer, 2016.

[2] National Assembly of Zambia, Standing Orders, 2024. Lusaka, Zambia: National Assembly of Zambia, 2024. [Online]. Available: https://www.parliament.gov.zm/sites/default/files/documents/general/Standing%20Orders%202024.pdf

[3] Inter-Parliamentary Union, World e-Parliament Report 2020. Geneva, Switzerland: Inter-Parliamentary Union, 2021.

[4] Inter-Parliamentary Union, "Centre for Innovation in Parliament," Inter-Parliamentary Union, Geneva. [Online]. Available: https://www.ipu.org/impact/democracy-and-strong-parliaments/centre-innovation-in-parliament. [Accessed: Oct. 2026].

[5] UK Parliament, "Parliamentary Bills," UK Parliament, London. [Online]. Available: https://bills.parliament.uk. [Accessed: Oct. 2026].

[6] Library of Congress, "Congress.gov: Legislation," Washington, DC, USA. [Online]. Available: https://www.congress.gov. [Accessed: Oct. 2026].

[7] United Nations Department of Economic and Social Affairs, "Bungeni: Parliamentary and Legislative Information System," Africa i-Parliaments Action Plan, 2010. [Online]. Available: https://www.bungeni.org. [Accessed: Oct. 2026].

[8] Government of the Republic of Zambia, The Electronic Government Act No. 41 of 2021. Lusaka, Zambia: Government Printer, 2021.

[9] Government of the Republic of Zambia, The Data Protection Act No. 3 of 2021. Lusaka, Zambia: Government Printer, 2021.

[10] Government of the Republic of Zambia, The Cyber Security and Cyber Crimes Act No. 2 of 2021. Lusaka, Zambia: Government Printer, 2021.

[11] B. Mainza, D. Zulu, and M. Chavula, "NAZ Order Papers System: A questions and motions management system with hybrid AI-assisted duplicate and similarity detection for the National Assembly of Zambia," B.Sc. final year project report, Dept. of Computing and Informatics, Univ. of Zambia, Lusaka, Zambia, 2026.

[12] OWASP Foundation, "OWASP Top 10:2021," 2021. [Online]. Available: https://owasp.org/Top10/. [Accessed: Oct. 2026].

[13] ZambiaLII – Zambia Legal Information Institute, "Legislation," Lusaka, Zambia. [Online]. Available: https://zambialii.org. [Accessed: Oct. 2026].

[14] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, “Workflow patterns,” Distributed and Parallel Databases, vol. 14, no. 1, pp. 5–51, 2003.

[15] M. Palmirani and F. Vitali, “Akoma-Ntoso for legal documents,” in Legislative XML for the Semantic Web, G. Sartor, M. Palmirani, E. Francesconi, and M. A. Biasiotti, Eds. Dordrecht, The Netherlands: Springer, 2011, pp. 75–100.

[16] European Commission, “LEOS – Open source software for editing legislation,” Interoperable Europe. [Online]. Available: https://interoperable-europe.ec.europa.eu/collection/justice-law-and-security/solution/leos-open-source-software-editing-legislation. [Accessed: Oct. 2026].

[17] D. Harel, “Statecharts: A visual formalism for complex systems,” Science of Computer Programming, vol. 8, no. 3, pp. 231–274, 1987.

[18] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, “Role-based access control models,” Computer, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[19] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, “Proposed NIST standard for role-based access control,” ACM Trans. Information and System Security, vol. 4, no. 3, pp. 224–274, 2001.

[20] J. W. Hunt and M. D. McIlroy, “An algorithm for differential file comparison,” Bell Laboratories, Murray Hill, NJ, USA, Computing Science Tech. Rep. 41, 1976.

[21] E. W. Myers, “An O(ND) difference algorithm and its variations,” Algorithmica, vol. 1, no. 1–4, pp. 251–266, 1986.

[22] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, Dept. of Information and Computer Science, Univ. of California, Irvine, CA, USA, 2000.

[23] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” Internet Engineering Task Force, RFC 7519, May 2015.

[24] N. Provos and D. Mazières, “A future-adaptable password scheme,” in Proc. USENIX Annual Technical Conf., FREENIX Track, Monterey, CA, USA, 1999, pp. 81–91.

[25] M. Fowler, Patterns of Enterprise Application Architecture. Boston, MA, USA: Addison-Wesley, 2002.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA, USA: Addison-Wesley, 1994.

[27] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston, MA, USA: Addison-Wesley, 2003.

[28] J. Brooke, “SUS: A ‘quick and dirty’ usability scale,” in Usability Evaluation in Industry, P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, Eds. London, U.K.: Taylor & Francis, 1996, pp. 189–194.

[29] J. Sauro and J. R. Lewis, Quantifying the User Experience: Practical Statistics for User Research, 2nd ed. Cambridge, MA, USA: Morgan Kaufmann, 2016.

[30] A. Bangor, P. T. Kortum, and J. T. Miller, “An empirical evaluation of the System Usability Scale,” Int. J. Human–Computer Interaction, vol. 24, no. 6, pp. 574–594, 2008.

[31] J. Kirakowski and M. Corbett, “SUMI: The Software Usability Measurement Inventory,” British J. Educational Technology, vol. 24, no. 3, pp. 210–212, 1993.

[32] F. D. Davis, “Perceived usefulness, perceived ease of use, and user acceptance of information technology,” MIS Quarterly, vol. 13, no. 3, pp. 319–340, 1989.

[33] K. Finstad, “The usability metric for user experience,” Interacting with Computers, vol. 22, no. 5, pp. 323–327, 2010.

[34] J. R. Lewis, B. S. Utesch, and D. E. Maher, “UMUX-LITE: When there’s no time for the SUS,” in Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI ’13), Paris, France, 2013, pp. 2099–2102.

[35] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design science in information systems research,” MIS Quarterly, vol. 28, no. 1, pp. 75–105, 2004.

[36] ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva, Switzerland: ISO, 2011.

[37] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” J. Management Information Systems, vol. 24, no. 3, pp. 45–77, 2007.

[38] K. Schwaber and J. Sutherland, The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game, Nov. 2020. [Online]. Available: https://scrumguides.org

[39] I. Sommerville, Software Engineering, 10th ed. Harlow, U.K.: Pearson, 2016.

[40] Redgate Software, “Flyway documentation,” [Online]. Available: https://documentation.red-gate.com/flyway. [Accessed: Oct. 2026].

[41] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Upper Saddle River, NJ, USA: Addison-Wesley, 2010.

[42] ISO/IEC/IEEE 29148:2018, Systems and software engineering — Life cycle processes — Requirements engineering. Geneva, Switzerland: ISO, 2018.

[43] National Assembly of Zambia, “Bills,” National Assembly of Zambia, Lusaka. [Online]. Available: https://www.parliament.gov.zm. [Accessed: Oct. 2026].

[44] Vercel Inc., “Next.js documentation,” [Online]. Available: https://nextjs.org/docs. [Accessed: Oct. 2026].

[45] VMware Tanzu, “Spring Boot reference documentation,” [Online]. Available: https://docs.spring.io/spring-boot/. [Accessed: Oct. 2026].

[46] PostgreSQL Global Development Group, “Chapter 12. Full Text Search,” PostgreSQL Documentation. [Online]. Available: https://www.postgresql.org/docs/current/textsearch.html. [Accessed: Oct. 2026].

[47] PostgreSQL Global Development Group, “pg_trgm — support for similarity of text using trigram matching,” PostgreSQL Documentation. [Online]. Available: https://www.postgresql.org/docs/current/pgtrgm.html. [Accessed: Oct. 2026].

[48] D. Merkel, “Docker: Lightweight Linux containers for consistent development and deployment,” Linux Journal, vol. 2014, no. 239, Art. no. 2, Mar. 2014.

[49] A. Bangor, P. T. Kortum, and J. T. Miller, “Determining what individual SUS scores mean: Adding an adjective rating scale,” J. Usability Studies, vol. 4, no. 3, pp. 114–123, 2009.

APPENDICES

APPENDIX A: INDIVIDUAL CONTRIBUTION STATEMENTS

This appendix contains one contribution statement for each member of the group, in the form required by the Department. Each statement is consistent with the summary in Section 5.11.

A.1. Chali Gerald Chibwana (Backend Lead)

Purpose of this Statement

This statement documents my individual contribution to a group project, consistent with the content of the main report. I have not claimed the work of another group member as my own.

Project and Group Information

Item	Details
Project Title	Digital Bill Processing System for the National Assembly of Zambia
Student Name	Chali Gerald Chibwana
Computer Number	2017011498
Supervisor	Dr Lighton Phiri (NAZ co-supervisors: Mr Paul Chola, Mr Alinani Sichone, Mr Daniel Sichanga)
Group Size	3
Group Members	Chali Gerald Chibwana (2017011498), Backend Lead; Prince Daka (2021418987), DevOps and Testing Lead; Mitchell Ndona (2021395111), Frontend Lead

Overall Group Project

The group designed, implemented, deployed and evaluated a Digital Bill Processing System for the National Assembly of Zambia: a web application that tracks each Government Bill through eight stages from drafting at the Ministry of Justice to publication by the Government Printer, with role-based ownership of stages, version control and comparison, amendments, document storage, search, notifications, statutory-period tracking, reports and an append-only audit trail. The work was divided by architectural layer: backend (Chali Chibwana), DevOps and testing (Prince Daka), and frontend and evaluation (Mitchell Ndona).

Individual Role and Responsibilities

Backend Lead. Responsible for the design and implementation of the REST API, the database schema and migrations, authentication and authorisation, the Bill workflow and statutory-

period rules, and the event-driven audit and notification modules; also built the frontend screens for the features I implemented.

Detailed Individual Contribution

My contribution is summarised in Section 5.11. In detail:

- Analysed the Bill stages, stage owners and statutory periods with the team and the NAZ co-supervisors, and modelled them as the BillStage state machine and BillTransitionPolicy.
- Designed the database (ERD) and wrote the migrations for users, Bills, history, versions, notifications, amendments, audit, version text and documents.
- Implemented JWT authentication and the stateless security configuration.
- Implemented the Bill, version, amendment, document, notification, dashboard and audit modules, including the domain events and listeners.
- Built the Bill submission, Bill detail, versions and comparison, amendments and audit screens in the frontend.
- Wrote 39 automated tests and contributed to the requirements, design and documentation.

Individual Deliverables

1. **Backend API** — Spring Boot application: auth, bill, amendment, document, notification, dashboard, audit and user packages
2. **Database schema** — ERD and Flyway migrations V1–V4 and V6–V10
3. **Workflow and legal rules** — BillStage, BillTransitionPolicy and BillDeadlinePolicy
4. **Feature screens** — Bill submission, Bill detail, versions and comparison, amendments and audit pages
5. **Tests** — 39 test methods: audit, version and document tests

Contribution to the Main System or Research Output

I primarily developed the backend core: the data model, authentication, the workflow and statutory-period rules, and the version, amendment, document, notification and audit modules, as well as the frontend screens that use them. The search and reporting modules were developed by Prince Daka, and the frontend foundation, sign-in and Reports pages by Mitchell Ndona. The requirements and the overall design were developed jointly.

Individual Implementation and Technical Work

Java 21 and Spring Boot 4 (Spring Security, Spring Data JPA, application events and transactional listeners), PostgreSQL and Flyway (including the append-only trigger), JWT and BCrypt, SHA-256 checksums, and React/TypeScript for the feature screens, including the LCS comparison.

Individual Testing and Evaluation Contribution

I wrote 39 test methods (AuditControllerTest, AuditListenerTest, AuditServiceTest, BillVersionTest and BillDocumentControllerTest), tested the API manually with Postman, debugged integration issues such as CORS and the response contract, and took part in the demonstrations and the user study.

Collaboration with Other Group Members

The team agreed the API contract as shared TypeScript types and DTOs, so the frontend and backend could be built in parallel. Every change went through a pull request reviewed by another member and the CI checks. Work was planned on a Trello board and coordinated in biweekly Scrum meetings (held online while schools were closed) and a group chat. The backend depended on the pipeline for verification and deployment; the frontend depended on the backend endpoints; and the evaluation depended on all three being deployed.

Contribution Summary

Category	**Work**
Primarily my work	Backend core, database schema, workflow and legal rules, audit, versions, amendments, documents, notifications; related frontend screens; 39 tests.
Joint work	Problem analysis, requirements (SRS), architecture, documentation and presentations.
Others' work used in mine	CI/CD and deployment, search and reports (Prince Daka); frontend foundation, components and evaluation (Mitchell Ndona).

Estimated Contribution (individual % / collaborative %)

Activity	**Individual % / Collaborative %**
Problem Analysis	35 / 100
Literature Review	30 / 100
Requirements Analysis	40 / 100
System/Research Design	50 / 100
Implementation/Development	60 / 100

Activity	Individual % / Collaborative %
Testing and Evaluation	25 / 100
Documentation	35 / 100
Other	30 / 100 (deployment configuration)

Statement of Authorship and Contribution

I confirm that the contribution described in this statement accurately represents the work that I personally undertook as part of the group project. I have identified collaborative work and have not knowingly claimed the work of another group member as my own.

Student Name: Chali Gerald Chibwana

Computer Number: 2017011498

Signature: _____

Date: _____

Supervisor Verification

I confirm that, to the best of my knowledge, the contribution described above is consistent with the student's participation in the group project.

Supervisor Name: Dr Lighton Phiri

Signature: _____

Date: _____

A.2. Prince Daka (DevOps and Testing Lead)

Purpose of this Statement

This statement documents my individual contribution to a group project, consistent with the content of the main report. I have not claimed the work of another group member as my own.

Project and Group Information

Item	Details
Project Title	Digital Bill Processing System for the National Assembly of Zambia
Student Name	Prince Daka
Computer Number	2021418987
Supervisor	Dr Lighton Phiri (NAZ co-supervisors: Mr Paul Chola, Mr Alinani Sichone, Mr Daniel Sichanga)
Group Size	3
Group Members	Chali Gerald Chibwana (2017011498), Backend Lead; Prince Daka (2021418987), DevOps and Testing Lead; Mitchell Ndona (2021395111), Frontend Lead

Overall Group Project

The group designed, implemented, deployed and evaluated a Digital Bill Processing System for the National Assembly of Zambia: a web application that tracks each Government Bill through eight stages from drafting at the Ministry of Justice to publication by the Government Printer, with role-based ownership of stages, version control and comparison, amendments, document storage, search, notifications, statutory-period tracking, reports and an append-only audit trail. The work was divided by architectural layer: backend (Chali Chibwana), DevOps and testing (Prince Daka), and frontend and evaluation (Mitchell Ndona).

Individual Role and Responsibilities

DevOps and Testing Lead. Responsible for the continuous integration pipeline, the cloud deployment to AWS, the automated test suite, and the search and reporting modules of the backend; also reviewed and merged pull requests.

Detailed Individual Contribution

My contribution is summarised in Section 5.11. In detail:

- Wrote the backend CI workflow with tests, Docker build and smoke test, and kept the CI workflows current.

- Set up the AWS resources (EC2, RDS and ECR) and wrote the manual backend deployment workflow using OIDC, ECR and SSM with an RDS snapshot gate.
- Implemented search and retrieval (V5 migration, BillSearchCriteria, paging and escaping).
- Implemented reports, the StageDurations calculation and the PDF and Excel exports.
- Wrote 85 of the 125 automated tests across nine test classes.
- Reviewed and merged pull requests, and contributed to the requirements, testing strategy and documentation.

Individual Deliverables

1. **CI pipeline** — backend-ci.yml: JUnit, Docker build and smoke test
2. **Deployment workflow** — deploy-backend.yml: OIDC, ECR, SSM and RDS snapshot gate
3. **Search module** — Migration V5 and BillSearchCriteria
4. **Reporting module** — ReportController, StageDurations, PDF and Excel writers
5. **Test suite** — 85 test methods in nine classes

Contribution to the Main System or Research Output

I primarily developed the CI/CD pipeline and deployment, the search and reporting modules, and most of the automated tests. The backend core was developed by Chali Chibwana, and the frontend foundation, sign-in and Reports pages by Mitchell Ndona; the Reports page uses my report endpoints. The requirements and the overall design were developed jointly.

Individual Implementation and Technical Work

GitHub Actions, Docker, AWS (EC2, ECR, RDS, SSM, IAM OIDC), PostgreSQL full-text and trigram search, Apache PDFBox and Apache POI, and JUnit 5, Mockito and Spring MockMvc.

Individual Testing and Evaluation Contribution

I wrote 85 of the 125 test methods, built and maintained the pipeline that runs them on every change, and verified each deployment with the Docker smoke test.

Collaboration with Other Group Members

The team agreed the API contract as shared TypeScript types and DTOs, so the frontend and backend could be built in parallel. Every change went through a pull request reviewed by

another member and the CI checks. Work was planned on a Trello board and coordinated in biweekly Scrum meetings (held online while schools were closed) and a group chat. The backend depended on the pipeline for verification and deployment; the frontend depended on the backend endpoints; and the evaluation depended on all three being deployed.

Contribution Summary

Category	**Work**
Primarily my work	CI pipeline, AWS deployment workflow, search, reports, 85 tests.
Joint work	Problem analysis, requirements (SRS), architecture, documentation and presentations.
Others' work used in mine	Backend core that my tests verify (Chali Chibwana); Reports page that uses my endpoints (Mitchell Ndona).

Estimated Contribution (individual % / collaborative %)

Activity	**Individual % / Collaborative %**
Problem Analysis	30 / 100
Literature Review	30 / 100
Requirements Analysis	30 / 100
System/Research Design	25 / 100
Implementation/Development	20 / 100
Testing and Evaluation	45 / 100
Documentation	30 / 100
Other	60 / 100 (CI/CD and cloud)

Statement of Authorship and Contribution

I confirm that the contribution described in this statement accurately represents the work that I personally undertook as part of the group project. I have identified collaborative work and have not knowingly claimed the work of another group member as my own.

Student Name: Prince Daka

Computer Number: 2021418987

Signature: _____

Date: _____

Supervisor Verification

I confirm that, to the best of my knowledge, the contribution described above is consistent with the student's participation in the group project.

Supervisor Name: Dr Lighton Phiri

Signature: _____

Date: _____

A.3. Mitchell Ndona (Frontend Lead)

Purpose of this Statement

This statement documents my individual contribution to a group project, consistent with the content of the main report. I have not claimed the work of another group member as my own.

Project and Group Information

Item	Details
Project Title	Digital Bill Processing System for the National Assembly of Zambia
Student Name	Mitchell Ndona
Computer Number	2021395111
Supervisor	Dr Lighton Phiri (NAZ co-supervisors: Mr Paul Chola, Mr Alinani Sichone, Mr Daniel Sichanga)
Group Size	3
Group Members	Chali Gerald Chibwana (2017011498), Backend Lead; Prince Daka (2021418987), DevOps and Testing Lead; Mitchell Ndona (2021395111), Frontend Lead

Overall Group Project

The group designed, implemented, deployed and evaluated a Digital Bill Processing System for the National Assembly of Zambia: a web application that tracks each Government Bill through eight stages from drafting at the Ministry of Justice to publication by the Government Printer, with role-based ownership of stages, version control and comparison, amendments, document storage, search, notifications, statutory-period tracking, reports and an append-only audit trail. The work was divided by architectural layer: backend (Chali Chibwana), DevOps and testing (Prince Daka), and frontend and evaluation (Mitchell Ndona).

Individual Role and Responsibilities

Frontend Lead. Responsible for the repository and project set-up, the frontend foundation (application structure, shared types, API client, mock API and reusable components), the sign-in and Reports pages, and the design and execution of the user evaluation.

Detailed Individual Contribution

My contribution is summarised in Section 5.11. In detail:

- Created the repository, README, Git workflow, environment template, Docker Compose file and first CI and Vercel workflows.

- Scaffolded the Next.js frontend with shared types, the API client, stage definitions and a mock API.
- Built the layout and reusable components (sidebar, notification feed, pipeline bar, stage badge, Bill card, search bar) and the first dashboard and Bills pages.
- Built the sign-in page and the Reports page with PDF and Excel export.
- Designed the evaluation instrument (three tasks, SUS and TAM items), built the Google Form, piloted it with three people, ran the user study with 25 participants and analysed the results.
- Prepared the presentation and demonstration script, and contributed to the requirements and documentation.

Individual Deliverables

1. **Project set-up** — Monorepo, README, environment template, Docker Compose, CI and Vercel workflows
2. **Frontend foundation** — Next.js scaffold, types, API client, mock API and reusable components
3. **Sign-in and Reports pages** — Branded sign-in with role redirect; reports with PDF and Excel export
4. **Evaluation instrument** — Five-minute Google Form with three tasks, the SUS and TAM items (Appendix E)
5. **Evaluation results** — User study with 25 participants: task completion, SUS 70.8, TAM and themes (Section 7.2)

Contribution to the Main System or Research Output

I primarily developed the project set-up and frontend foundation, the sign-in and Reports pages, and the user evaluation. The backend and the feature screens that depend on it (Bill submission and detail, versions, amendments and audit) were developed by Chali Chibwana; the pipeline, deployment, search, reports API and most tests by Prince Daka. The requirements and the overall design were developed jointly.

Individual Implementation and Technical Work

Next.js 14, React, TypeScript, Tailwind CSS, TanStack Query, Mock Service Worker, ESLint, GitHub Actions and Vercel; Google Forms and Google Sheets for the evaluation.

Individual Testing and Evaluation Contribution

I designed and ran the user evaluation: the instrument, recruitment, scoring and the analysis of the quantitative and qualitative results; I also tested the frontend pages and their exports.

Collaboration with Other Group Members

The team agreed the API contract as shared TypeScript types and DTOs, so the frontend and backend could be built in parallel. Every change went through a pull request reviewed by another member and the CI checks. Work was planned on a Trello board and coordinated in biweekly Scrum meetings (held online while schools were closed) and a group chat. The backend depended on the pipeline for verification and deployment; the frontend depended on the backend endpoints; and the evaluation depended on all three being deployed.

Contribution Summary

Category	**Work**
Primarily my work	Project set-up, frontend foundation and components, sign-in and Reports pages, user evaluation.
Joint work	Problem analysis, requirements (SRS), architecture, documentation and presentations.
Others' work used in mine	Backend API and feature screens (Chali Chibwana); report endpoints, pipeline and deployment (Prince Daka).

Estimated Contribution (individual % / collaborative %)

Activity	**Individual % / Collaborative %**
Problem Analysis	35 / 100
Literature Review	40 / 100
Requirements Analysis	30 / 100
System/Research Design	25 / 100
Implementation/Development	20 / 100
Testing and Evaluation	30 / 100
Documentation	35 / 100
Other	50 / 100 (evaluation and presentation)

Statement of Authorship and Contribution

I confirm that the contribution described in this statement accurately represents the work that I personally undertook as part of the group project. I have identified collaborative work and have not knowingly claimed the work of another group member as my own.

Student Name: Mitchell Ndona

Computer Number: 2021395111

Signature: _____

Date: _____

Supervisor Verification

I confirm that, to the best of my knowledge, the contribution described above is consistent with the student's participation in the group project.

Supervisor Name: Dr Lighton Phiri

Signature: _____

Date: _____

APPENDIX B: USER MANUAL / INSTALLATION GUIDE

B.1 Using the deployed system

1. Open https://naz-bill-system.vercel.app in a current version of Chrome, Edge or Firefox.
2. Sign in with the account for your role. The demonstration accounts are *mwansa.kabwe* (Ministry of Justice officer), *chileshe.mumba* (National Assembly officer), *bwalya.nkonde* (Government Printer) and *admin* (Administrator); passwords are issued separately by the administrator.
3. **Submit a Bill (Ministry of Justice).** Choose *All Bills* → *Submit bill*, enter the long title and the short title, then select *Submit bill*. The Bill is created at the Draft stage with a reference such as NAZ-2026-007.
4. **Find a Bill.** On *All Bills*, type part of a title or reference in the search box and, optionally, choose a stage in the filter.
5. **Save a new version.** Open the Bill and, in the versions section, select *Save new version*, paste the revised text of the Bill and describe what changed. Only the office that owns the current stage sees this button; earlier versions stay exactly as they are.
6. **Compare versions.** Select *Compare versions* and choose the two versions. Removed lines are shown in red and added lines in green.
7. **Propose or decide an amendment.** On the Bill page, select *Propose amendment* and enter the clause, the proposed wording and the rationale. The owner of the current stage selects *Accept* or *Reject* and records the reasons; the decision is final.
8. **Upload the official PDF.** In the documents section of the Bill page, choose a PDF of up to 10 MB.
9. **Advance the Bill.** Select *Advance to <next stage>*. The next office receives a notification (bell icon).
10. **Reports and audit.** *Reports* shows Bills per stage and processing times, with PDF and Excel export. The Administrator can open *Audit* to view and export the audit trail.

B.2 Running the system locally

```
git clone https://github.com/mitchellndona/bills-processing-system.git
cd bills-processing-system
cp .env.example .env          # set DATABASE_URL, JWT_SECRET (32+ chars), JWT_EXPIRY_HOURS
docker-compose up --build     # PostgreSQL on 5432, API on http://localhost:8080
cd frontend && npm install && npm run dev    # http://localhost:3000
cd ../backend && ./mvnw test     # run the automated tests
```

Listing 3: Commands to run the system and its tests locally

B.3 Deploying

The frontend deploys to Vercel automatically when changes are merged into *main*. The backend is deployed by running the *Deploy Backend to EC2* workflow in GitHub Actions: first take an RDS snapshot if the build adds a migration, then run the workflow and tick the snapshot confirmation. The workflow builds the image, pushes it to Amazon ECR and replaces the container on EC2 through AWS Systems Manager.

B.4 Troubleshooting

Table 23: Common problems and solutions

Problem	**Solution**
"Session expired" or redirected to sign-in	The token lasts 24 hours by default; sign in again.
An action button is missing	Only the office that owns the Bill's current stage can act; check the stage and your role.
Two roles in the same browser interfere	Sessions are stored per browser; use separate browser windows or profiles for different roles.
Upload rejected	Only genuine PDF files of up to 10 MB are accepted.
Backend fails to start after a deployment	Check the container logs; if a migration failed, restore the RDS snapshot.

APPENDIX C: ADDITIONAL SYSTEM DESIGNS / DIAGRAMS

Figure 24 and Figure 25 show the sequence of interactions for signing in and for proposing and deciding an amendment.

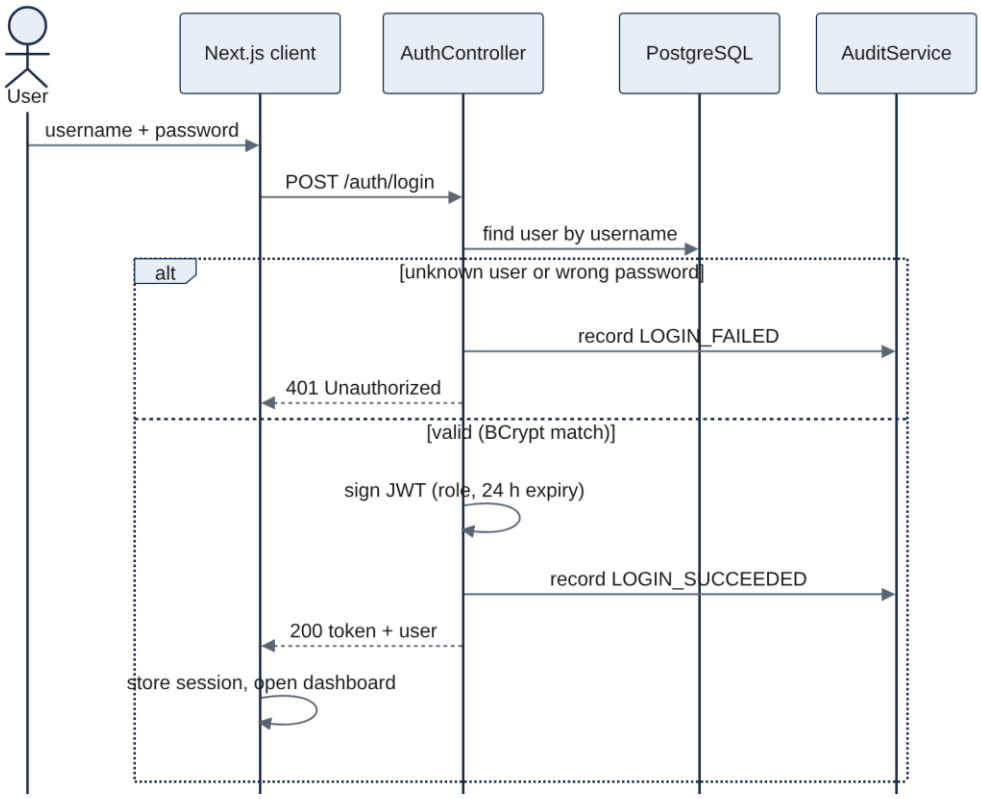


Figure 24: Sequence diagram: signing in and calling a protected endpoint (UC1)

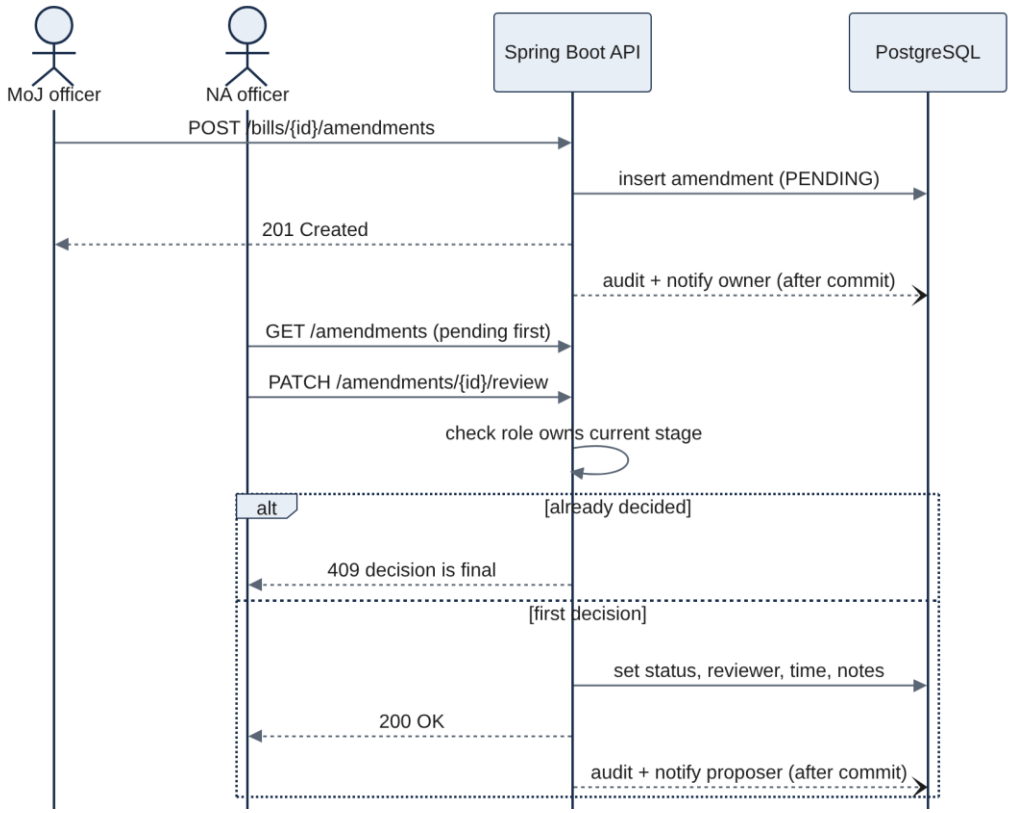


Figure 25: Sequence diagram: proposing and deciding an amendment (UC7, UC8)

Table 24: Bill stages, owners and statutory periods

Stage	Owner	Statutory period	Legal effect
Draft	MoJ officer	None	—
First Reading	NA officer	None	—
Second Reading	NA officer	None	—
Committee Stage	NA officer	20 working days (SO 113(1))	Minimum: the committee may not report earlier
Report Stage	NA officer	None	—
Third Reading	NA officer	None	—
Presidential Assent	Government Printer	21 days (Art. 66(1); SO 127(1), (6))	Maximum: assent deemed given on expiry
Published	— (terminal)	None	—

APPENDIX D: ADDITIONAL TEST CASES AND RESULTS

Table 25 lists every automated test method in the backend suite, taken from the source code. Each method name states the behaviour it verifies.

Table 25: All automated test methods by class

Class	n	Test methods
BackendApplicationTests	1	contextLoads
AuditControllerTest	10	listIsAdminOnly, exportIsAdminOnly, anonymousRefused, adminListsEntries, filtersByAction, pageSizeCapped, billTrailOpenToAll, unknownBill, exportIsCsv, formulaInjectionNeutralised
AuditListenerTest	6	billCreated, stageAdvanced, versionAdded, amendmentProposed, amendmentReviewed, handlersAreSynchronous
AuditServiceTest	5	copiesActorDetails, unknownActorStillRecorded, loginSucceeded, loginFailed, longUsernameTruncated
BillDeadlinePolicyTest	13	committeeStageCountsWorkingDays, committeeStageStartingOnSaturday, presidentialAssentCountsCalendarDays, targetProducesDueDate, overdueOnlyAfterTarget, noTargetNeverOverdue, committeeStageNeverOverdue, assentExpiryIsNotOverdue, publishedNeverOverdue, deemedCompleteAfterAssentPeriod, onlyAssentAdvancesOnExpiry, daysInStage, nullInputsAreSafe
BillSearchCriteriaTest	8	rejectsUnknownSortField, alwaysTieBreaksOnId, mapsEveryWhitelistedField, defaultsToDescending, blankQueryBecomesNull, escapesIlikeWildcards, normalisesPaging, dateToCoversWholeDay
BillTransitionPolicyTest	6	adminCanAdvanceAnyStage, noSkipping, noBackwardsOrSameStage, pipelineOrderIsIntact, publishedIsTerminal, publishedHasNoOwner
BillVersionTest	9	stageOwnerSavesVersion, numbering, otherOfficesRefused, draftBelongsToMinistry, publishedIsClosed, blankLinkIsNull, unknownBill, listing, requestValidation
DashboardControllerTest	7	emptySystemHasAllStages, missingStagesAreZero, activeExcludesDraftAndPublished, noTargetsMeansNothingOverdue, overdueFromTargetsOnly, overdueQueriesNonTerminalStages, publishedThisYearUsesLusakaYear
BillDocumentControllerTest	9	ownerUploads, otherOfficeRefused, notPdfRefused, tooLargeRefused, versionMustBelongToBill, unknownBill, download, listUsesSummaries, safeFileNames
NotificationControllerTest	8	anotherUsersNotificationIs404, unknownIdIsAlsoA404, lookupIsScopedToTheCaller, markingOwnNotificationRead, markingReadIsIdempotent, listIsScopedToTheCaller, readAllIsScopedToTheCaller, unknownAuthenticatedUserIsRejected
NotificationListenerTest	11	ownersAndSubmitterAreNotified, actorIsExcluded, submitterWhoActsIsNotNotified, recipientsAreDeduplicated, publishedNotifiesOnlyTheSubmitter, missingSubmitterIsNotFatal, nobodyToNotifyWritesNothing, notificationCarriesTheBillAndTheMove, presidentialAssentCarriesThePrintingInstruction, versionUploadUsesTheSameRule, writeFailureDoesNotPropagate
ReportControllerTest	12	emptySystemReportsEveryStage, countsPerStage, sharesAddUp, reportCarriesItsOwnDate, emptyHistoryReportsNoFigures, timesComeFromHistory, partDaysSurvive, waitingBillsAreReportedButNotMeasured, reportsAgreeOnOccupancy, statutoryPeriodsAreCarriedWithTheirCitation, publishedHasNoProcessingTime, eachReportIssuesOneReadOfEachSource
ReportExportTest	8	billsPerStagePdfCarriesTheFigures, billsPerStageSpreadsheetStoresNumbers, processingTimePdfCarriesTheFiguresAndTheCaveats, absentFiguresAreBlankInTheSpreadsheet, emptySystemStillExports, exportsAreNamedDownloads, exportsCarryTheRightMediaType, exportAgreesWithTheJson
StageDurationsTest	12	visitEndsAtTheNextStageChange, openVisitIsExcluded, stalledBillDoesNotFlatterTheFigures, reentryCountsTwice, billsAreNotRunTogether, medianOfOddCount, medianOfEvenCount, orderOfRowsIsIrrelevant, simultaneousRowsDoNotGoNegative, emptyHistoryStillReportsEveryStage, publishedNeverCompletes, unattributableRowsAreSkipped

The final run of the full suite, on GitHub Actions on 16 October 2026, executed 158 test cases (each parameterised method counted once per input) with 0 failures, 0 errors and 0 skipped, in 30.9 seconds; the log is shown in Figure 21.

APPENDIX E: DATA COLLECTION INSTRUMENTS

The user acceptance test was delivered as a Google Form titled "Digital Bill Processing System: 5-minute usability test". Its questions are reproduced below in the order in which they appeared.

E.1 Consent and device

Table 26: Consent and device questions

Question	Response options
I agree to take part, and understand my answers are anonymous.	I agree
Which device are you using?	Phone; Laptop / desktop

E.2 Tasks

Table 27: Task questions

Task	Question	Response options
T1 Find a Bill and report its stage	What is its current stage?	Free text
	Did you complete Task 1?	Yes, easily; Yes, with some difficulty; No
T2 Compare two versions of the Agricultural Marketing Bill	Write down one change you found	Free text
	Did you complete Task 2?	Yes, easily; Yes, with some difficulty; No
T3 Move a practice Bill to the next stage	Did you complete Task 3?	Yes, easily; Yes, with some difficulty; No

E.3 System Usability Scale and acceptance items

All items were answered on a five-point scale from 1 (strongly disagree) to 5 (strongly agree). Items 1 to 10 are the System Usability Scale [28]; items 11 to 13 measure perceived usefulness, perceived ease of use and intention to use [32].

Table 28: Usability and acceptance items

No.	Statement
1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

No.	Statement
11	Using this system would make processing Bills between offices easier.
12	Learning to use this system would be easy for me.
13	If I worked in one of these offices, I would want to use this system.
14	Anything confusing, or anything you liked? (optional, free text)

E.4 Scoring

SUS score = 2.5 × [Σ(odd item − 1) + Σ(5 − even item)], giving 0 to 100 [28]. For example, a participant who answers 4 to every odd item and 2 to every even item scores 2.5 × (5 × 3 + 5 × 3) = 75. Task completion rate = ("Yes, easily" + "Yes, with some difficulty") ÷ all responses. TAM items are reported as means and as the share of participants answering 4 or 5. The calculations are reproducible from the exported responses with the analysis script kept with the project files.

E.5 Individual results

Table 29 lists the scored results of every participant, in the order of submission. T1 to T3 are the self-reported task outcomes (E = easily, D = with some difficulty, N = not completed).

Table 29: Scored results of each participant (n = 25)

ID	Device	T1	T2	T3	SUS	PU	PEOU	BI
P01	Phone	E	E	E	70.0	—	—	—
P02	Computer	E	D	E	57.5	—	—	—
P03	Computer	E	E	E	87.5	4	5	5
P04	Phone	D	D	D	77.5	4	4	4
P05	Phone	E	E	E	90.0	5	5	5
P06	Phone	E	D	D	37.5	5	4	5
P07	Phone	E	E	E	75.0	5	5	5
P08	Phone	D	D	N	80.0	4	4	4
P09	Phone	E	D	D	50.0	4	4	4
P10	Phone	N	N	N	30.0	4	5	3
P11	Phone	E	E	N	27.5	5	5	5
P12	Phone	D	N	D	50.0	3	2	3
P13	Phone	E	E	E	70.0	5	5	5
P14	Phone	E	E	E	87.5	5	5	5
P15	Phone	E	E	E	100.0	4	4	4
P16	Phone	E	E	E	50.0	2	4	4
P17	Computer	E	E	E	85.0	5	4	4
P18	Phone	E	E	E	87.5	5	5	5
P19	Computer	D	D	E	90.0	5	5	5
P20	Phone	E	E	E	72.5	4	5	5
P21	Computer	E	E	E	95.0	5	5	5
P22	Phone	E	D	D	52.5	3	3	3

ID	Device	T1	T2	T3	SUS	PU	PEOU	BI
P23	Phone	D	D	N	52.5	2	2	2
P24	Phone	E	E	D	95.0	5	5	5
P25	Phone	E	E	E	100.0	4	5	5

APPENDIX F: ADDITIONAL CODE / CONFIGURATION

The complete source code is in the project repository at https://github.com/mitchellndona/bills-processing-system. Selected extracts follow.

```
public enum BillStage {
    DRAFT(null), FIRST_READING(null), SECOND_READING(null),
    COMMITTEE_STAGE(new StatutoryPeriod(20, DayCount.WORKING, Effect.EARLIEST_REPORT,
        "Standing Orders 2024, SO 113(1)")),
    REPORT_STAGE(null), THIRD_READING(null),
    PRESIDENTIAL_ASSENT(new StatutoryPeriod(21, DayCount.CALENDAR,
Effect.DEEMED_COMPLETE_ON_EXPIRY,
        "Constitution Art. 66(1); Standing Orders 2024, SO 127(1) and (6)")),
    PUBLISHED(null);
    ...
}
```

Listing 4: Stages with statutory periods, units, legal effects and citations (BillStage.java)

```
export function diffLines(before: string, after: string): DiffLine[] {
  const a = before.split('\n'), b = after.split('\n'), n = a.length, m = b.length
  // lcs[i][j] = length of the longest common subsequence of a[i..] and b[j..]
  const lcs = Array.from({ length: n + 1 }, () => new Uint32Array(m + 1))
  for (let i = n - 1; i >= 0; i--)
    for (let j = m - 1; j >= 0; j--)
      lcs[i][j] = a[i] === b[j] ? lcs[i + 1][j + 1] + 1 : Math.max(lcs[i + 1][j], lcs[i][j +
1])
  // trace back: equal lines are 'same'; otherwise follow the larger subproblem
  ...
}
```

Listing 5: Line-based LCS comparison of two versions (diff.ts, abridged)

```
on:
  workflow_dispatch:
    inputs:
      snapshot_taken:
        description: 'RDS snapshot taken and Available?'
        type: boolean
        required: true
permissions:
  id-token: write      # assume the AWS role via OIDC; no stored keys
steps:
  - name: Require snapshot confirmation
    if: ${{ inputs.snapshot_taken != true }}
    run: exit 1        # Flyway has no down migrations
  - uses: aws-actions/configure-aws-credentials@v4
  - uses: aws-actions/amazon-ecr-login@v2
  - uses: docker/build-push-action@v6
  - name: Deploy over SSM   # replace the container on EC2
```

Listing 6: Backend deployment workflow (deploy-backend.yml, abridged)

APPENDIX G: OTHER SUPPORTING MATERIAL

G.1 Response to supervisor feedback

Table 30: Supervisor feedback and how it was addressed

Feedback	How it was addressed in this report and the system
Take a software-engineering-centred approach	Design science, Scrum and design principles (Chapter 3); requirements traceability, ISO/IEC 25010 NFRs, CI/CD and testing (Chapters 4–6)
Reference the requirements	FR identifiers used in Chapters 4–7, in commit messages, code and tests
Add a literature review and describe elicitation	Chapter 2; Section 3.4
Correct the architecture diagram and the ERD	Figure 6 and Figure 7 redrawn from the actual code and migrations
Acknowledge the NAZ co-supervisors	Title page and Acknowledgements; Section 3.4
Drop the team name	The team is referred to by its members' names
Use a standard instrument; at least 30 participants; demographics; pilot	SUS with TAM items on the live system; pilot with 3 people; 25 participants, including ICT students from the School of Education; device recorded (Sections 3.9 and 7.2; Appendix E); shortfalls stated in Section 7.6
Demonstrate crucial features: version control and search	Tasks T1 (search) and T2 (version comparison) in the user study; Sections 5.5 and 5.7

G.2 Project links

- Live system: https://naz-bill-system.vercel.app
- Source code: https://github.com/mitchellndona/bills-processing-system
- User evaluation form: https://docs.google.com/forms/d/e/1FAIpQLSfZO5T1q3q-1TrwecrTt6XA_lVOxSnalwQ5XhFz1NipQZRAAw/viewform