Faculty: Technology, Cybersecurity

NAME: FAVOUR SOBECHI OPARA

MODULE CODE: PROM02

MODULE TITLE: Computing Master's Project

SUPERVISOR: Dr. MRIGANKA BIWAS

SUBMISSION DATE AND TIME: 19th September 2025, 23:59

PROJECT TOPIC: Design, Implementation, and Evaluation of a Secure Software Development Lifecycle (SSDLC) for a React/Django-Based E-Commerce Web Application: A Case Study of Abatrades.

Academic Misconduct is an offence under university regulations, and this involves:

- **Plagiarism** – where you use information from another information source (including your previously submitted work) and pass it off as your own. This can be through direct copying, poor paraphrasing and/or absence of citations.
- **Collusion** – where you work too closely, intentionally, or unintentionally, with others to produce work that is similar in nature. This can be through loaning of materials, drafts or through unauthorised use of a fellow student's work.
- **Asking another person to write your assignment** – where you ask another individual or company to complete your work for you, be that paid or unpaid, and submit it as if it were your own.
- **Unauthorised use of artificial intelligence** – where you use artificial intelligence tools to generate your assignment instead of completing it yourself and/or where you have not been given permission to use artificial intelligence tools by your module leader. Please complete the following declaration around you use of artificial intelligence tools in your assignment.

STATEMENT ON USE OF ARTIFICIAL INTELLIGENCE TOOLS:

• I have used artificial intelligence tools to generate an idea for my assignment: **YES/<u>NO</u>**
• I have used artificial intelligence tools to write my assignment for me: **YES/<u>NO</u>**
• I have used artificial intelligence tools to brainstorm ideas for my assignment: **YES/<u>NO</u>**
• I have used artificial intelligence tools to correct my original assignment: **YES/<u>NO</u>**

DECLARATION

- I understand that by submitting this piece of work I am declaring it to be my own work and in compliance with the university regulations on Academic Integrity.
- I confirm that I have done this work myself without external support or inappropriate use of resources.
- I understand that I am only permitted to use artificial intelligence tools in line with guidance provided by my Module Leader, and I have not used artificial intelligence tools outside this remit.
- I confirm that this piece of work has not been submitted for any other assignment at this or another institution prior to this point in time.
- I can confirm that all sources of information, including quotations, have been acknowledged by citing the source in the text, along with producing a full list of the sources used at the end of the assignment.
- I understand that academic misconduct is an offence and can result in formal disciplinary proceedings.
- I understand that by submitting this assignment, I declare myself fit to be able to complete the assignment and I accept the outcome of the assessment as valid and appropriate.

Faculty of Technology

Department of Computer Science

PROM02 – MSc Dissertation

MSc Cybersecurity

Student Name: Favour Sobechi Opara

Supervisor Name: Dr Mriganka Biswas

Design, Implementation, and Evaluation of a Secure Software Development Lifecycle (SSDLC) for a React/Django-Based E-Commerce Web Application: A Case Study of Abatrades.

29/08/2025

Declaration

I declare the following:

(1) that the material contained in this dissertation is the result of my own work and that due acknowledgement has been given in the bibliography and references to ALL sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is 14,574.

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal, if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on the eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other Department or from other institutions using the service. In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the University of Sunderland Policy Statement on Ethics in Research, and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED: Favour Sobechi Opara

DATE: 29/08/2025

Abstract

E-commerce applications built on React/Django frameworks has exposed critical risk with existing Secure Software Development Lifecycle (SSDLC) approaches, which typically provide generic security guidance without addressing framework-specific vulnerabilities or e-commerce threat models. This research developed and validated a specialized SSDLC framework tailored for React/Django-based e-commerce applications. This framework incorporates threat modelling, technology-specific security controls, and e-commerce business logic protection that addresses React and Django vulnerabilities. A systematic threat-risk-control matrix using customized CVSS-style scoring (1-27 scale) provides risk prioritization guidance for development teams. A prototype e-commerce Abatrades , was developed using the SSDLC, and its security was tested through Vulnerability Assessment and Penetration Testing (VAPT). Security validation introduced 67 specific test cases across 10 security domains, utilizing tools including Kali Linux, Wireshark, Burp Suite, OWASP ZAP, and OWASP Dependency Check for VAPT. Results demonstrated security effectiveness with 88.1% test pass rate and zero critical vulnerabilities identified. The remaining 11.9% tests were not applicable. The framework successfully prevented all OWASP Top 10 attacks, while providing protection against e-commerce-specific threats such as price manipulation, inventory tampering, and unauthorized data access. Automated scanning identified 16 dependency vulnerabilities and 56 configuration issues, all classified as medium-to-low risk and remediable through version updates and configuration changes. Comparation with major SSDLC frameworks (NIST SSDF, Microsoft SDL, OWASP SAMM, SANS, ISO/IEC 27034) revealed significant advantages in technology specificity, e-commerce context integration, and implementation practicality. Unlike generic approaches, this framework provides actionable, code-level security controls that address the unique vulnerability patterns of React/Django integrations and e-commerce business logic requirements.

Table of contents

1 Introduction

1.1 Background

1.1.1 Current Web Application Security Landscape

The E-commerce web application landscape has witnessed an unprecedented surge in vulnerabilities, with cybersecurity threats evolving at an alarming rate. According to the AV-TEST Institute, approximately 450,000 new malicious programs and potentially harmful applications are registered daily, representing a significant threat to web application security (Paul, 2025).

Open Web Application Security Project (OWASP) identify vulnerabilities such as injection attacks, broken authentication, sensitive data exposure, and cross-site scripting (XSS) as predominant threats (OWASP Foundation, 2021). These vulnerabilities have significant implications for e-commerce platforms, where financial transactions and customer data protection are paramount concerns.

1.1.2 Adoption of React/Django Framework in E-commerce Development

The e-commerce sector has experienced a notable migration surge towards modern web development frameworks, particularly the combination of React for frontend development and Django for backend services. This migration is driven by the need for scalable, secure, and maintainable e-commerce platforms that can handle complex business requirements. React remains a dominant choice for e-commerce frontend development, with over 12.3 million live websites utilising the framework as of September 2023, demonstrating its widespread adoption across various industries including e-commerce (Jonna, 2024).

Leading e-commerce platforms built on Django reported a 25% increase in user engagement after integrating ML-based recommendation systems, demonstrating the framework's effectiveness in enhancing commercial applications (Mansi, 2025). Additionally, Django provides essential e-commerce functionalities such as user authentication, session management, and secure database operations out of the box, reducing development time and improving security posture. Kumar et al. (2021) emphasise that Django's comprehensive security features, including built-in protection

against common vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF), makes it an ideal choice for e-commerce backend development. Similarly, the adoption of React for frontend development has been driven by its ability to create responsive, interactive user interfaces that enhance customer experience in online retail environments.

1.1.3 Emerging Security Vulnerabilities in React/Django Applications

Despite React/Django framework security advantages, the widespread adoption of React/Django frameworks has resulted in documented security incidents that demonstrate the real-world consequences of inadequate security practices. Recent analysis reveals that a single data breach costs companies an average of $3.86 million and takes 280 days to contain, with e-commerce businesses bearing particularly high risks due to the sensitive nature of customer financial data they process (DQ India, 2024).

In 2018, a major e-commerce platform using Django suffered a data breach due to insecure password handling, demonstrating how framework misconfigurations can lead to serious security compromises (CodyMohit, 2025). More recently, Django's security team addressed multiple critical vulnerabilities in 2025, including CVE-2025-48432, which allows log injection via unescaped request paths, enabling attackers to corrupt logs or mask malicious activity. This vulnerability was patched in Django 5.2.3, 5.1.11, and 4.2.23, but not before potentially affecting thousands of applications (Divya, 2025).

Nearly 70% of web applications are vulnerable to attacks due to security holes, with Django-based applications increasingly vulnerable to Cross-Site Scripting (XSS) attacks when security best practices are not properly implemented (Fourrage, 2024). Furthermore, the complexity of modern React/Django applications often involves extensive use of third-party libraries and dependencies, creating expanded attack surfaces that traditional security approaches may not address. Papadakis et al. (2024) identified that the reuse of third-party packages in React brings challenges associated with security vulnerabilities introduced by third-party dependencies.

Cross-site request forgery (CSRF) attacks have been particularly problematic, allowing remote attackers to trigger unauthenticated forged requests via vectors involving DNS CNAME records and malicious JavaScript code (Team, 2025). Similarly, session hijacking

vulnerabilities in Django's “RemoteUserBackend” have enabled attackers to gain unauthorized access to systems using other users' session data, affecting versions across multiple Django releases.

These documented vulnerabilities and real-world incidents underscore the need for framework-specific security development lifecycles that address the unique challenges posed by React/Django e-commerce applications that generic Secure Software Development Life Cycle (SSDLC) cannot address.

1.1.4 The Need for a Tailored SSDLC Framework

Current SSDLC frameworks, while comprehensive, often lack the specificity required to address unique security challenges posed by React/Django e-commerce applications. Traditional SSDLC approaches typically follow generic methodologies that may not account for the specific risk inherent to modern React and Django combinations. According to TechTarget (2024), many established Software Development Life Cycle (SDLC) frameworks can be adapted to incorporate security provisions, but they are not inherently designed for framework-specific security challenges.

The integration complexity between frontend and backend components in React/Django applications requires security practices that span the entire application stack. This includes secure communication patterns, proper error handling across component boundaries, and consistent security policy enforcement from the user interface to the database layer. NIST's Secure Software Development Framework (SSDF), while providing fundamental secure software development practices, is designed as a high-level framework that requires adaptation for specific technology stacks (NIST, 2025). The framework's generic approach may not provide adequate guidance for the integrated security concerns specific to React/Django applications.

1.2 Aim

The aim of this project is to design, implement, and evaluate a comprehensive Secure Software Development Lifecycle (SSDLC) framework for React/Django-based e commerce applications, using Abatrades (a made-up small business) as a case study to

demonstrate enhanced security practices and assess their effectiveness in mitigating web application vulnerabilities.

1.3 Objectives

1. Conduct comprehensive literature review on various SSDLC frameworks and e-commerce security vulnerabilities within 2 weeks, producing a systematic analysis of various academic sources to establish theoretical foundation and identify security best practices for web applications.
2. After gathering literature review, design tailored SSDLC framework for React/Django based E-commerce web application within 3 weeks, incorporating threat modelling, secure coding standards, and automated testing protocols, documented through detailed process workflows and security checklists.
3. Design E-commerce web-application called Abatrades, implement security controls including authentication mechanisms, input validation, encryption protocols, and access controls within 4 weeks, demonstrating practical application of SSDLC principles through measurable security enhancements.
4. Deploy automated security testing tools including SAST, DAST, and dependency scanning to identify vulnerabilities automatically during development phases and evaluate the security posture of Abatrades. This will be concluded within 1 week.
5. Conduct penetration test based off OWASP top 10 web-application vulnerabilities on Abatrades to evaluate the security posture. This will also be concluded within 1 week.
6. Document findings, fix vulnerabilities with severity levels of critical, high, medium, and low (if any), conduct a revalidation test to confirm vulnerabilities identified are patched. This will be included in the SSDLC framework for added security. This will be concluded in 1 week.
7. Deliver a final report and reusable SSDLC templates summarizing findings, lessons learned, and recommendations for future projects within 1 week.

1.4 Research Approach

This research focuses on creating and evaluating innovative artifacts to solve practical problems. The study employs an approach that combines theoretical framework

development with practical implementation and effectiveness evaluation. The research is structured around the development, application, and assessment of a tailored SSDLC specifically designed for React/Django e-commerce applications.

1.4.1 Research Philosophy and Strategy

Effective security solutions require both theoretical and practical applicability. The study employs a case study strategy using a made-up e-commerce web application called Abatrades for framework validation. This approach enables the examination of real-world security challenges and the practical effectiveness of the proposed SSDLC in a controlled environment.

The methodology is designed to address the research question through four interconnected phases which are:

- framework design
- Practical application
- Performance evaluation
- Continuous improvement.

This cyclical approach ensures that the developed framework is both theoretically sound and practically viable for real-world e-commerce development scenarios.

1.4.2 Data Collection and Analysis Methods

The following are data collection and analysis methods that will be used throughout the project.

1. **Primary Data Collection:** This involves documentation of security vulnerabilities, implementation processes, and testing results throughout the Abatrades development and security enhancement phases. Data will be collected through Vulnerability and Penetration Testing (VAPT), and remediation effectiveness before and after SSDLC implementation.
2. **Secondary Data Collection:** Encompasses a comprehensive literature review of existing SSDLC frameworks, React/Django security best practices, and e-commerce security incident reports.

3. **Qualitative Analysis:** Focuses on the applicability of security controls, development workflow, and the usability of the proposed SSDLC framework.
4. **Quantitative Analysis:** Examines measurable security improvements from VAPT results.

1.4.3 Validation and Evaluation Approach

SSDLC will be validated by developing an e-commerce platform in accordance with its framework to demonstrate real-world applicability and identify implementation challenges. SSDLC effectiveness will then be evaluated through VAPT.

1.5 Structure of the report

The report is organised into 6 main chapters with supporting materials.

1.5.1 Chapter summary

1. **Introduction:** Establishes the research context and foundation, presenting the background of e-commerce security challenges and React/Django framework adoption. It articulates the research aims, objectives, and approach while outlining the report structure and addresses ethical, social, professional, legal, and security considerations.
2. **Literature Review:** Provides a comprehensive analysis of existing SSDLC frameworks and security practices, and e-commerce vulnerability landscapes. It further analyses current academic and industry knowledge to identify research gaps and establish the theoretical foundation for the proposed framework.
3. **Practical Research Methodology:** It describes the systematic development and application of the SSDLC framework to the e-commerce platform, including security testing protocols and evaluation methods.
4. **Analysis of Results:** Presents the research findings from both the SSDLC framework development and its practical application and shows evidence of security testing results, vulnerability assessments, and framework effectiveness metrics while comparing outcomes with existing literature and industry benchmarks.

5. **Project Evaluation and Reflection:** Shows a comprehensive evaluation of the research project, reviewing the achievement of stated objectives and assessing the methodology's effectiveness.
6. **Conclusion and Recommendations:** It summarises the research contributions and their implications for React/Django e-commerce development practice and presents recommendations for framework adoption, future research directions, and industry implementation of secure development practices.

1.5.2 Supporting materials

1. **Reference List:** Contains all academic and industry sources referenced throughout the report, formatted according to Harvard referencing style.
2. **Appendices:** Contains project proposal, supervisor meeting logs, and link to supporting materials including complete SSDLC framework documentation, reusable templates and checklists, and detailed security testing results.

1.6 Ethical, Social, Professional, Legal, and Security considerations

1.6.1 Ethical considerations

Ethical considerations include ensuring data minimization principles, implementing privacy-by-design approaches, and preventing unauthorized access during security testing phases. Penetration testing must be conducted responsibly without creating vulnerabilities. The research findings could influence industry practices, requiring honest reporting of both successful and unsuccessful security implementations. Transparency in vulnerability disclosure and remediation processes ensures ethical responsibility towards the security community.

1.6.2 Social considerations

The project addresses critical social needs by enhancing e-commerce security, protecting consumers from cyber threats and financial fraud. Small businesses like Abatrades require affordable security solutions to compete safely in digital markets. The SSDLC framework uses enterprise-level security practices for smaller organizations, reducing the digital security divide. However, implementing robust security measures may increase development costs and complexity, potentially creating barriers for

startups. The research contributes to building public trust in e-commerce platforms, supporting digital inclusion and economic growth in online retail sectors.

1.6.3 Professional considerations

This project directly impacts the software engineering profession by establishing best practices for secure development lifecycles in modern web applications. Professional responsibility includes ensuring the framework's accuracy and effectiveness, as incorrect security implementations could have serious consequences. The project supports the profession's obligation to protect public welfare through secure software development practices and continuous security improvement methodologies.

1.6.4 Legal considerations

The project must comply with GDPR regulations when handling personal data during application development and security testing phases. The research involves cybersecurity testing activities that must be conducted within legal boundaries, ensuring authorized testing environments and proper consent mechanisms. E-commerce applications face additional legal requirements, including the Payment Card Industry Data Security Standard (PCI DSS) compliance for payment processing and consumer protection regulations. The SSDLC framework must incorporate legal compliance requirements, and vulnerability disclosure processes must follow responsible disclosure principles to avoid potential legal liabilities while enhancing overall security posture.

1.6.5 Security considerations

Security considerations encompass the comprehensive protection of research data, testing environments, and the responsible development of security frameworks that enhance rather than compromise system security. The research involves active security testing and vulnerability assessment activities that require careful management to prevent the introduction of new security risks while evaluating existing ones.

2 Literature review

2.1 Introduction to the Literature Review

This literature review provides an examination of current knowledge regarding SSDLC frameworks, with particular emphasis on their application to React/Django-based e-commerce applications. The review synthesises academic research, industry best practices, and documented security incidents to establish the theoretical foundation for developing a tailored SSDLC framework that addresses the unique security challenges of modern web application development.

2.1.1 Search Strategy, Selection and Exclusion Criteria

Booth et al. (2018) emphasise that systematic literature searching involves *"... a systematic search for studies and aims for a transparent report of study identification, leaving readers clear about what was done to identify studies, and how the findings of the review are situated in the relevant evidence*". The literature search seeks to identify relevant academic sources to ensure coverage of security challenges and framework developments.

Where will these searches be carried out from? A.T. Still University (2024) recommends that a minimum of 3 databases needs to be searched for a systematic review to minimise publication bias. Primary search terms include combination of terms such as "SSDLC," "React security," "Django security," "e-commerce vulnerabilities," and "web application security frameworks" and these were chosen because they are related to the project in scope. Searches were conducted across multiple academic databases. Example of these databases are IEEE Xplore, ACM Digital Library, and Google Scholar. Sources are also supplemented by industry security publications to ensure comprehensive coverage of both academic and practitioner perspectives.

Following the guidance of Thomas et al. (2008), the use of the Exclusion criteria eliminates sources focused only on legacy technologies, generic security discussions without framework-specific guidance, and publications lacking empirical evidence or practical validation. The review prioritises sources that examine the intersection of

secure development practices with modern JavaScript frontend and Python backend frameworks, particularly in e-commerce contexts.

2.1.2 Overview of Key Themes

Using Thematic analysis provides structure to the overall development of a project's literature review. Nowell et al. (2017) advises that a well thought out thematic analysis produces trustworthy and insightful findings when conducted methodically. Atlas.ti (2024) also notes that thematic literature reviews excel at providing analytical structure on findings from studies, while ensuring an understandable overview by concentrating on themes rather than individual studies.

Due to the project's structure, the literature reveals five critical themes that aids the development of a tailored SSDLC framework for React/Django e-commerce applications. These themes provide structure for subsequent review sections and highlight the nature of secure web application development, while also enabling more analytical, rather than purely descriptive literature examination.

Theme 1: SSDLC Framework Limitations

This indicates a central concern across multiple sources, where researchers consistently identify gaps between generic security frameworks and the specific requirements of modern web development stacks. (Kirvan, 2022) notes that many established SDLC frameworks can be adopted to incorporate security provisions, but they are not designed for framework-specific security challenges. Berry.S (2023) also criticizes traditional SDLC models for lacking adaptability to modern stacks like React/Django. The literature reveals that traditional SSDLC approaches, while foundational, often lack the granular guidance necessary for framework-specific security implementation.

Theme 2: Framework-Specific Security Challenges

The challenges highlight unique vulnerability patterns and security considerations that affects React and Django frameworks. Papadakis et al. (2024) identify critical security challenges in React applications, particularly focusing on dependency management vulnerabilities where third-party packages introduce security risks during JavaScript

application development. Also, Alauthman et al. (2025) demonstrate that integrating DevSecOps into modern web development uncovers new vulnerabilities, such as React's insecure component rendering and Django's default settings exposing endpoints. Their case studies emphasize the need for tailored controls per framework. The literature showcases a distinction in framework security paradigms that require specialised knowledge and targeted security controls beyond generic web application security measures.

Theme 3: E-commerce Security Imperatives

It addresses security requirements of online retail platforms, including regulatory compliance, financial data protection, and the business impact of security incidents. DQ India (2024) demonstrates that a single data breach costs companies an average of $3.86 million and takes 280 days to contain, with e-commerce businesses bearing particularly high risks due to the sensitive nature of customer financial data they process. IBM (2024) also notes that retail and e-commerce face the highest breach costs due to financial data sensitivity. The literature emphasises the critical nature of security in e-commerce contexts and the severe consequences of security failures.

Theme 4: Integration of Security Complexities

It is the examination of security challenges arising from integrating React frontend and Django backend components. Oliver Lindberg (2019) notes that modern front-end frameworks such as React are not entirely immune to security vulnerabilities, requiring developers to implement framework-specific protections beyond traditional security measures. Donin and Taylor (2025) talks about the risk of react and Django vulnerabilities which revolve around token handling, and session management inconsistencies when frameworks are connected, validating the need for integration-aware security policies. The literature reveals that modern web applications face unique security risks at the integration points between different technological layers, requiring comprehensive security strategies that span the entire application stack.

Theme 5: Practical Implementation Gaps

Identifies the disconnect between theoretical security frameworks and their practical application in real-world development environments. Hyperproof (2024) highlights that one of the most challenging aspects of maintaining information security is ensuring that security practices are properly implemented across different technology layers, particularly when integrating frontend and backend security controls. FT (2023) also argues that cybersecurity policies often remain "on paper" without real behavioural or procedural change unless enforced by operations-level accountability. The literature consistently highlights the need for actionable guidance that bridges the gap between security theory and development.

Table 2.1 Theme literature

Theme	Literature
Theme 1: SSDLC Framework Limitations	Kirvan, (2022), Berry.S (2023)
Theme 2: Framework-Specific Security Challenges	Papadakis et al. (2024), Alauthman et al. (2025)
Theme 3: E-commerce Security Imperatives	DQ India (2024), IB (2024)
Theme 4: Integration Security Complexities	Oliver Lindberg (2019), Donin and Taylor (2025)
Theme 5: Practical Implementation Gaps	Hyperproof (2024), FT (2023)

2.1.3 Literature Scope and Limitations

This review acknowledges limitations in the current literature landscape that inform the research contribution. Temporal limitations arise from the rapid evolution of both React and Django frameworks, with security considerations changing as new versions introduce features and deprecate others.

The scope of the literature focuses on React/Django combinations, which represents a subset of modern web development approaches. While this specificity enables detailed analysis, it may limit the broader applicability of findings to other framework combinations. The review addresses this limitation by identifying transferable security principles while maintaining focus on the target technology stack.

Due to the scope of the project, there is scarcity of comprehensive case studies examining SSDLC implementation in real-world React/Django e-commerce

environments. Snyk (2020) notes that traditional VAPT in production are no longer sufficient for securing applications, as the software industry and types of attacks have evolved, yet there is limited research on practical SSDLC implementation in modern development contexts. The literature presents several theoretical frameworks, creating opportunities for more comprehensive practical research contributions.

The review also reveals a clear research gap in framework-specific SSDLC guidance for React/Django applications, particularly in e-commerce contexts where security requirements are most stringent.

2.2 SSDLC Frameworks

The development of SSDLC frameworks addresses the increasing complexity and sophistication of cybersecurity threats in modern software development. This section examines SSDLC approaches to secure development practices, analysing their methodologies, strengths, and limitations when applied to specific technology stacks. Some notable SSDLC frameworks will be reviewed, as well as their limitations when considering the specificity of this project.

2.2.1 National Institute of Standards and Technology (NIST) Secure Software Development Framework (SSDF)

NIST SSDF represents one of the most comprehensive and authoritative approaches to integrating security throughout the SDLC. NIST (2022) defines the SSDF as "... *a core set of high-level secure software development practices that can be integrated into each SDLC implementation*" which is designed to help software developers reduce vulnerabilities in released software and mitigate the potential impact of undetected vulnerabilities.

CSRC (2025) emphasizes that the SSDF addresses a critical gap in software development, noting that "... *few SDLC models explicitly address software security in detail, so practices like those in the SSDF need to be added to and integrated with each SDLC implementation*." The framework organizes secure development practices into four primary groups that spans the entire software lifecycle which are listed below.

1. **Prepare the Organization (PO):** Scribe Security (2025) explains that this practice focuses on identifying specific security requirements based on the tools and methodologies teams will use for software development, recognizing that SSDLC is highly dependent on individuals in scope and requiring organizational readiness before implementing technical security measures.
2. **Protect the Software (PS):** This addresses the protection of all software components from tampering and unauthorized access throughout the development process. It recognizes that software security begins with securing the development environment itself, implementing access controls for source code repositories, and establishing secure communication channels between development tools and systems.
3. **Produce Well-Secured Software (PW):** This focuses on producing software with minimal security vulnerabilities through secure design practices, implementation of security controls, and comprehensive testing. Wiz (2024) highlights that this practice group emphasizes integrating security features and considerations directly into software design and architecture, following a *"security by design approach that facilitates the identification of potential vulnerabilities early in the development cycle*."
4. **Respond to Vulnerabilities (RV):** This addresses the identification and remediation of residual vulnerabilities in software releases, including vulnerability disclosure processes, patch management, and incident response procedures. The framework acknowledges that despite best efforts, vulnerabilities may still exist in released software and organizations must be prepared to respond effectively.

Framework Limitations and Adaptation Challenges

The limitation of this framework becomes notable when organizations attempt to implement the SSDF in specific technology contexts. While the framework provides excellent high-level guidance, Black Duck (2019) notes that the SSDF's generic nature requires significant customization, with industry expert Gary Migues observing that the missing piece is workshops where the CISOs' organize sessions to guide security officers and developers through its implementation and scope.

2.2.2 Microsoft Security Development Lifecycle (SDL)

Microsoft (2024) describes SDL as a method it uses to achieve DevsecOps which has been fundamental to product development for over 20 years.

Microsoft SDL practices include ten key security practices that integrate security into each stage of the development process. Microsoft Learn (2024) further sections the SDL to consists of seven components and two supporting security activities with the five core phases being requirements, design, implementation, verification, and release, each containing mandatory checks and approvals from supervisors to ensure security and privacy requirements and best practices are adhered. These are the practices Microsoft SDL offers.

1. **Require use of proven security features, languages, and frameworks**: When conforming to regulatory requirements, organizations must strive to use proven security frameworks when developing their SSDLC model. Using a new model might introduce new complications and security risk which might cost time and effort to fix.
2. **Security design review and threat modelling**: When designing secure products, ideology must shift from how products should work to how products might be abused. Steps to model threats and mitigate them are Identifying use case scenario and affected assets, creating architecture overview, identify threats, and tracking mitigation.
3. **Define and use cryptography standards**: Most technical security assurance depends on cryptography. These include authentication and authorization, communication security, data transfer, and so on. It is essential to know what data to protect, how the data is stored (data at rest) and transmitted (data in transit).
4. **Secure the software supply chain**: Majority of products nowadays are built on third-party components (open-sourced or commercial) for agile development approach. However, the security of a workload depends on the security of all components of that workload. Developers consume Open-Source or commercialized software like Git clone and Wget. It's imperative to develop

governed workflows revolving around security for third-party software consumption.

5. **Secure the engineering environment**: The development environment can easily be accessed through targeting user accounts. Access to source code and engineering systems should be managed, operated and secured through Zero Trust and least privilege access policies. If the source code is stored on-premises like in a local computer managing the development of the web application, it should have strict access control.
6. **Perform security testing**: Applications must be tested to find potential risk of any application and to validate the security results from the development process which is applicable to the project. Without this visibility, developers cannot make decisions about the security of the workload like planning, prioritizing, and implementing fixes.
7. **Ensure operational platform security**: An attacker that gains unauthorized access to the production infrastructure can easily take over the workload, the data in it, and move laterally on other internal systems. The security of the workload depends on the overall security of the environment including the security of identities, networks, servers, containers, email, user endpoints, and other systems.
8. **Implement security monitoring and response**: This practice focuses on maintaining continuous visibility into both the vulnerabilities attackers can exploit and anomalies that may be signs of an active attack. This is critically important to guide your risk mitigation efforts and to ensure you can detect, respond to, and recover from attacks.
9. **Provide security training**: Anyone in the organization who makes decisions that impact security of applications must understands the implications of decisions. This makes training vital to users, developers, product line managers, and testers.

2.2.3 OWASP Software Assurance Maturity Model (SAMM)

The OWASP SAMM represents a community-driven, open-source approach to secure software development that emphasizes maturity-based improvement and organizational adaptability. OWASP SAMM (2024) describes the model as "... *an open*

framework to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organization," which overall supports the SDLC.

It also explains that the model has evolved from its original 2009 version, with version 2.0 addressing functions identified through intensive discussions and input from industry practitioners and the OWASP community. These are the functions.

1. **Governance Function:** This focuses on establishing software security as a core business process, encompassing strategy and metrics, policy and compliance, and education and guidance practices. SecureFlag (2023) emphasizes that governance "*involves the development of policies, strategies, and metrics, ensuring the organization has a defined roadmap for software security*" while also addressing third-party management and vendor security practices.
2. **Design Function:** It's a function that integrates security principles from the initial stages of software development, including threat assessment, secure architecture design, and data protection strategies. The objective is to incorporate a security by design approach that reduces vulnerabilities in the final product through early identification and mitigation of potential security issues.
3. **Implementation Function:** It encompasses processes related to the construction and deployment of software components, with attention to defect management and secure build practices. Tarlogic (2023) notes that implementation activities are focused on distributing reliable software with little flaws through secure coding practices and secure deployment procedures.
4. **Verification Function:** This involves comprehensive testing and review techniques to ensure software security throughout the development lifecycle. This includes design review, code review, and security testing activities designed to identify and address potential security issues before deployment.
5. **Operations Function:** It is a function that ensures security in the software's operational environment through incident management, environment hardening, and operational management practices. SecureFlag (2023) explains that operations function maintains security during the deployment phase and post-

deployment and ensures a robust security posture throughout the software's lifecycle.

2.2.4 SANS Secure Development Framework

SANS security awareness (2024) describes their secure development framework as "... *focusing on growing threats to applications by providing training, certification, research, and community initiatives to help security professionals build, deploy and manage secure infrastructure, platforms, and applications*." This is the SANS approach to SSDLC.

1. **Educational and Training Foundation:** This foundation distinguishes the SANS approach from other SSDLC frameworks through its emphasis on developer education and skill development. SANS Security Awareness (2024) also emphasizes that developers need the training required to excel in securing their programming frameworks.
2. **OWASP Integration and Practical Application:** It forms a core component of the SANS secure development approach. SANS Security Awareness (2024) notes that their training programs address OWASP 2021 requirements for secure code training and emphasizes targeted training that reduce chances of becoming a victim of a breach by developing secure code from the beginning of development.
3. **Framework Components and Implementation:** It focuses on practical security controls integrated throughout the development process. SANS Institute (2024) describes their framework for secure application design and development as addressing "... *applications that protect data from unauthorized access or modification and ensure its availability*," emphasizing the importance of secure design principles from project inception through deployment.
4. **Agile and DevSecOps Integration:** It addresses the challenges of implementing security practices within modern development methodologies. SANS Security Awareness (2024) explains that their approach brings clarity on tackling the challenges faced in continuous development and provides guidance on things to look out for during development.
5. **Limitations and Adaptation Requirements:** This emerges when organizations attempt to scale SANS methodologies across diverse technology environments. While the framework provides excellent training and practical guidance, its

emphasis on education and skill development may require substantial organizational investment in training programs before security improvements become measurable.

2.2.5 ISO 27034 Application Security Standard

ISO/IEC 27034 provides a comprehensive framework for managing application security throughout the software development lifecycle, offering structured approaches to security control implementation and verification. ISO (2018) defines the standard as "*... providing guidance on information security to those specifying, designing and programming or procuring, implementing and using application systems.*" The following are ISO 27034 SSLDC frameworks.

1. **Framework Architecture and Core Concepts:** This concept establishes an approach to application security management through defined processes and control structures. Security Compass (2024) explains that ISO 27034 is an internationally recognized standard for application security and is aligned with other ISO standards.
2. **Organization Normative Framework (ONF):** This represents the organizational-level component of ISO 27034, establishing enterprise-wide security policies and controls. Johner (2025) describes the ONF as an Application Security Controls (ASC) library that companies should make specific to their contexts. This framework addresses business, regulatory, and technological contexts that influence security requirements.
3. **Application Normative Framework (ANF):** This framework provides application-specific instances of the organizational framework, tailored to individual software projects. PECB (2024) explains that organizations create an application-specific instance of ONF by starting with ANF.
4. **Application Security Controls (ASC):** This control forms the foundation of the ISO 27034 approach, providing specific, measurable security requirements for applications. Security Compass (2024) defines an ASC as a measure to prevent vulnerabilities from being exploited. The standard emphasizes that ASCs should be context-aware, applying different controls based on application characteristics and risk profiles.

5. **Levels of Trust and Risk-Based Approach:** This approach helps enable organizations to tailor their security controls based on application risk and business requirements. Security Compass (2024) explains that not every risk deserves a control, especially if the risk appetite can accommodate the risk.
6. **Implementation Challenges and Organizational Requirements:** This requirement reflects the comprehensive nature of ISO 27034 implementation. DataGuard (2025) notes that "... *while the standard provides a clear framework for how to ensure applications are secure and compliant with regulations like GDPR,*" implementation requires significant organizational commitment to establishing processes, controls, and verification mechanisms.

The table below provides a comparison summary of all six frameworks with a React/Django based SSDLC.

Table 2.2 Framework comparison summary

Framework	Pros for React/Django Development	Cons for React/Django Development
NIST SSDF	• Comprehensive organizational guidance • Government-backed credibility • Technology-agnostic flexibility • Strong compliance foundation • Well-documented practices	• Lacks framework-specific guidance • No React/Django security controls • Generic threat modeling approach • Requires substantial adaptation • Limited e-commerce focus
Microsoft SDL	• Mature, battle-tested methodology • Comprehensive threat modeling • Strong verification processes • Industry-proven effectiveness • Excellent documentation	• Microsoft technology bias • Limited JavaScript/Python guidance • Traditional SDLC focus • Complex for small teams • Minimal modern web framework support
OWASP SAMM	• Maturity-based implementation • Open-source and free • Community-driven updates • Technology flexibility • Gradual adoption approach	• High-level abstraction • No React/Django specifics • Implementation complexity • Generic security practices • Limited e-commerce context

SANS	• Practical, hands-on approach • Strong training components • OWASP integration • Developer-focused methodology • Real-world examples	• Training-dependent implementation • Limited systematic framework • Requires significant investment • No specific React/Django guidance • Education-focused rather than process
ISO/IEC 27034	• Comprehensive control library • Risk-based approach • International standard recognition • Structured ASC methodology • Strong verification framework	• Complex implementation process • Resource-intensive requirements • Generic application focus • Limited modern framework support

2.3 React Framework Security Considerations

React.js has emerged as one of the most widely adopted frontend JavaScript frameworks, with over 34% market share in JavaScript library usage and serving as the foundation for applications used by major corporations including Netflix, Airbnb, and Uber (Divyesh Maheta, 2025). Despite its popularity and robust architecture, React applications face unique security challenges that require careful consideration throughout the development lifecycle. This section examines the security considerations specific to React framework development, focusing on frontend vulnerabilities, component-based architecture security implications, dependency management challenges, and client-side security considerations.

2.3.1 Frontend Security Vulnerabilities in React Applications

React applications are particularly susceptible to frontend security vulnerabilities due to their client-side execution model and dynamic content rendering capabilities. Research indicates that Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) and injection attacks represents a significant security concern for React applications (Kasundra, 2025). The following gives a more detailed overview of React Vulnerabilities.

1. **Cross-Site Scripting (XSS) Vulnerabilities:** Cross-Site Scripting remains the most significant security threat facing React applications, despite the framework's built-in protection mechanisms (Dziuba, 2025). XSS is an injection of a malicious script into the code of a web application. The browser picks up this script and interprets it as legitimate. After that, the malicious code is executed as a part of an app (Dziuba, 2025). While React provides automatic HTML escaping through JSX, vulnerabilities can still emerge through improper use of React's escape hatches. According to Relevant software (2022), the primary XSS vulnerability vectors in React applications include:
    - **Dangerous innerHTML Property**: React's dangerouslySetInnerHTML property doesn't guarantee the code's security and renders all the types of data. The role of "dangerouslySetInnerHTML" is to inform a developer that the code assigned to it might be insecure (Dziuba, 2025). Research indicates that improper sanitization of dynamic values assigned to this property can lead to XSS vulnerabilities.
    - **Direct DOM Access**: React's escape hatches such as findDOMNode and createRef can introduce XSS vulnerabilities when developers bypass React's virtual DOM protection. Since an escape hatch returns the native DOM elements with their full API, the application can manipulate the element directly without going through React. This can lead to an XSS vulnerability (Dziuba, 2025).
    - **URL-based Script Injection**: React applications are vulnerable to malicious URLs that utilize the "JavaScript:" protocol. URLs without an "http:" or "https:" protocol can allow malicious code to sneak into your React application. If such a URL is hardcoded, it's harmless. But if it's provided by a user, it poses a potential React XSS threat (Dziuba, 2025).
2. **Cross-Site Request Forgery (CSRF) Attacks:** CSRF vulnerabilities in React applications exploit the trust relationship between users and web applications. Cross-Site Request Forgery (CSRF) is a security vulnerability that affects many websites and web applications, including those built with React. CSRF attacks exploit the trust a website has in a user's browser and tricks the user into making

unwanted requests to the server (Third Rock Techkno, 2024). Modern React applications implementing Single Page Application (SPA) architectures are particularly vulnerable to CSRF attacks due to their reliance on API communications and session-based authentication mechanisms. The stateless nature of React components can complicate traditional CSRF protection mechanisms that rely on server-side token generation and validation.

3. **SQL Injection and Backend Integration Vulnerabilities:** While React operates on the client-side, React applications can indirectly facilitate SQL injection attacks through improper handling of user inputs that are transmitted to backend services. SQL injection (SQLi) is a common security vulnerability that can compromise the integrity of data in React applications. Hackers can inject malicious SQL code into your database, enabling them to receive, edit, or delete data without being restricted by user permissions (Third Rock Techkno, 2024).
4. **State Management Security:** React applications employing state management libraries such as Redux face additional security challenges related to state serialization and client-side data exposure. Most React apps use Redux for app state management, which uses JSON, a lightweight data-interchange format, to set an initial app state: This is dangerous because "JSON.stringify" will not recognize sensitive data or XSS code (Dziuba, 2025).

2.3.2 Dependency Management Security Challenges

React applications typically rely on extensive dependency trees managed through Node Package Manager (NPM), creating significant security attack surfaces through third-party package vulnerabilities. NPM package vulnerabilities recent security research has highlighted numerous vulnerabilities within the NPM ecosystem affecting React applications. CVE-2023-25341 has been assigned to the security report for the @ladle/react NPM package. The vulnerability details are described as ‘Impact: Unauthorized file access on the server. Affected Versions: @ladle/react versions below 2.5.2’ (Node.js Security, 2024). The npm audit system reveals that React applications commonly contain multiple security vulnerabilities through their dependency chains. When a new npx create-react-app application is created and install any dependencies, I receive the message: 8 vulnerabilities (2 moderate, 6 high) (Josegabrielc, 2024).

React internally uses node.js, and hence any vulnerable library can pose a threat to the React app (Roy, 2021). The complexity of modern React applications can result in dependency conflicts and version mismatches that create security vulnerabilities. Automated dependency updating can introduce breaking changes or new vulnerabilities, while failing to update dependencies leaves applications vulnerable to known exploits.

2.3.3 Client-Side Security Challenges

React's client-side execution model introduces unique security challenges that differentiate it from traditional server-side applications. They include:

1. **Client-Side Data Exposure:** React applications execute entirely within the browser environment, making all client-side code and data potentially accessible to malicious actors. This includes API endpoints, business logic, and configuration data that may be embedded within the application bundle. The challenge of securing client-side applications is compounded by the fact that all React code is ultimately delivered to and executed by the user's browser, where it can be inspected, modified, or reverse-engineered.
2. **Browser-Based Attack Vectors:** According to Divyesh Maheta (2025), React applications face browser-specific attack vectors including:
    - **XML External Entity (XXE) Attacks**: XML External Entity Attack is a cybersecurity threat wherein attackers access outdated XML parsers. Attackers perform vulnerable activities, such as port scanning, denial of service, and request forgery.
    - **ZIP Slip Vulnerabilities**: File upload functionality in React applications can be exploited through ZIP slip attacks. There's a very specific vulnerability in React applications known as "zip slip" which involves the exploitation of the feature that enables uploading zip files. The attacker could unzip the uploaded files outside the assigned directory if the archive used to unpack the zip file is not secure (FreeCodeCamp, 2021).
    - **Session Management Vulnerabilities**: Client-side session management in React applications can be vulnerable to session hijacking, particularly when

sessions are stored in browser storage mechanisms that are accessible to malicious scripts.

3. **Content Security Policy (CSP) Implementation Challenges:** Implementing effective Content Security Policy (CSP) headers in React applications can be challenging due to the framework's dynamic content generation and potential conflicts with third-party libraries. The use of inline styles and scripts generated by React build tools can conflict with strict CSP policies, requiring careful configuration to maintain both security and functionality.

2.3.4 Security Testing and Detection Challenges

The dynamic nature of React applications creates unique challenges for security testing and vulnerability detection. Traditional static analysis tools may not effectively identify vulnerabilities in React applications due to the framework's component-based architecture and runtime behaviour.

2.4 Django Framework Security Features and Challenges

Django, marketed as "the web framework for perfectionists with deadlines," has established itself as a leading Python-based web framework known for its comprehensive security-first approach (Django Project, 2024). The framework implements a "batteries-included" philosophy, providing developers with extensive built-in security features designed to protect against common web application vulnerabilities. This section examines Django's built-in security mechanisms, common vulnerabilities affecting Django applications, backend API security considerations, and Object-Relational Mapping (ORM) security features and challenges.

2.4.1 Built-in Security Mechanisms

Django's reputation as a secure framework stems from its comprehensive suite of built-in security features that address the most common web application security threats. The Django development team prioritizes security from the ground up, incorporating various security features like built-in protection against common web attacks (e.g., Cross-Site Scripting (XSS), SQL injection), user authentication mechanisms, and the ability to define

granular access control permissions within the application (Drosopoulou, 2024). The following are Django specific vulnerabilities protections.

1. **Cross-Site Scripting (XSS) Protection:** Django provides automatic XSS protection through its template system. Django automatically escapes HTML in templates to prevent XSS attacks. Use the |safe filter cautiously, only when sure the content is safe (Danduprolu, 2024). The framework employs an auto-escaping mechanism that automatically escapes data dynamically inserted into templates unless explicitly marked as safe. However, the template protection has specific limitations that developers must understand. For example, when inserting dynamic data into JavaScript contexts, CSS styles, or HTML attributes, additional manual escaping may be required beyond Django's default protections (Django Documentation, 2024).
2. **Cross-Site Request Forgery (CSRF) Protection:** Django implements comprehensive CSRF protection through built-in middleware. Django has built-in protection against most types of CSRF attacks (Django Documentation, 2024). The framework's CSRF protection operates by checking for a secret token in each POST request, ensuring that malicious users cannot replay form submissions to the application.
3. **SQL Injection Prevention:** Django's Object-Relational Mapping (ORM) system provides strong protection against SQL injection attacks through automatic query parameterization. Django's ORM automatically escapes inputs to prevent SQL injection. Avoid using raw SQL queries whenever possible (Danduprolu, 2024).
4. **Authentication and Authorization Framework:** Django provides a comprehensive authentication and authorization system that handles user accounts, groups, permissions, and cookie-based user sessions. Django comes with a comprehensive and extensible user authentication system that manages user accounts, groups, permissions, and cookie-based user sessions (Patrich, 2023).

2.4.2 Common Django Vulnerabilities

Despite Django's strong security foundation, applications built with the framework remain susceptible to various security vulnerabilities, particularly when developers deviate from security best practices or when new vulnerability classes emerge.

Analysis of Django's security advisory history reveals several categories of vulnerabilities that have affected the framework. Just this last year there have been a few Denial-of-Service (DoS) vulnerabilities published, with CVE-2024-27351, CVE-2024-24680, and CVE-2023-46695 all pointing at possibilities of DoS with causes rooted in Django classes and templates (SecureFlag, 2024). The following are Django specific vulnerabilities.

1. **SQL Injection in Advanced Features**: Recent vulnerabilities have demonstrated that even Django's robust ORM can be susceptible to SQL injection under specific circumstances. CVE-2024-42005 represents a critical SQL injection vulnerability affecting Django's QuerySet.values() and values_list() methods on models with JSONField, where attackers can exploit the vulnerability through column aliases using maliciously crafted JSON object keys (Snyk, 2024).
2. **Cross-Site Scripting Vulnerabilities**: While Django provides strong XSS protection, specific vulnerabilities have been identified. Cross-site scripting attacks are known in Django too, with CVE-2022-22818 and CVE-2020-13596 both having a CVSS rating of 6.1 Medium (SecureFlag, 2024).
3. **Denial of Service Attacks**: DoS vulnerabilities in Django often stem from inadequate input validation in specific components, affecting features such as file uploads, URL validation, and internationalization components.
4. **Configuration-Related Vulnerabilities:** Security misconfigurations represent a significant source of vulnerabilities in Django applications. Security misconfiguration is the most commonly occurring vulnerability, often resulting from using default configurations or displaying excessively verbose errors (MDN Web Docs, 2024).
5. **ORM Leak Vulnerabilities:** An emerging class of vulnerabilities which affects Django applications is through insecure use of ORM methods. Recent discovery of vulnerabilities regarding the insecure use of Object Relational Mappers (ORMs) that could be exploitable to dump sensitive information was discovered. These issues were introduced when developers assumed that ORMs are safe from SQL

injection and did not consider the possibility that allowing un-validated user inputs into "safe" ORM methods could introduce a security risk (Elttam, 2024).

2.4.3 Backend API Security Considerations

Django REST Framework (DRF) extends Django's security model to API development but introduces additional security considerations. With increasing cyber threats and data breaches, ensuring robust security measures is essential for protecting sensitive data and maintaining user trust (Getachew, 2024). The following are the security considerations.

1. **Authentication Mechanisms**: DRF provides multiple authentication options including Token Authentication, Session Authentication, and JWT Authentication. JSON Web Tokens (JWT) are a popular method for securing APIs by providing a stateless authentication mechanism (Getachew, 2024). Each authentication method has distinct security implications and use cases.
2. **Authorization and Permissions**: DRF's permission system allows fine-grained control over API access. The framework provides built-in permission classes and supports custom permission logic, but developers must carefully configure these to prevent unauthorized access.
3. **API Endpoint Security**: REST APIs face unique security challenges including:
    - Rate limiting to prevent abuse and DoS attacks
    - Input validation for JSON payloads
    - Output serialization to prevent data leakage
    - CORS configuration to control cross-origin requests
4. **Session and Token Management:** Backend API security in Django applications relies heavily on proper session and token management. The framework supports various token-based authentication schemes, each with specific security considerations.
5. **Token Lifecycle Management**: Proper token management includes secure generation, storage, rotation, and revocation mechanisms. Applications must implement appropriate token expiration policies and refresh mechanisms.

6. **Session Security**: When using session-based authentication, applications must configure secure session cookies, implement proper session timeout policies, and protect against session fixation attacks.
7. **Data Serialization Security:** Django applications serving as API backends must carefully manage data serialization to prevent information disclosure. Minimize the data exposed by the API to reduce the risk of leaking sensitive information (Getachew, 2024).

2.5 Justification for using selected Security Analysis Tools

The implementation of a comprehensive Secure Software Development Lifecycle (SSDLC) for React/Django-based e-commerce applications requires rigorous security testing and validation through both automated and manual testing methodologies. This section provides academic and practical justification for the selection of specific penetration testing and security analysis tools that will be employed in this research to evaluate the effectiveness of the proposed SSDLC framework.

2.5.1 OWASP ZAP for Automated Vulnerability Scanning

OWASP Zed Attack Proxy (ZAP) has been selected as the primary automated vulnerability scanning tool based on extensive academic research demonstrating its effectiveness in detecting web application security vulnerabilities. Research by Arta et al. (2024) conducted a comprehensive evaluation of OWASP ZAP against other vulnerability scanners, concluding that "... *OWASP ZAP performed better than the other scanning tools mentioned in this paper*" with a score of 75.61% compared to alternative scanners.

The tool's alignment with the OWASP Top 10 framework makes it particularly suitable for this research because it means that the tool is specifically designed to identify and address these critical security risks (Polaki, 2023).

2.5.2 OWASP Dependency Check (DC) for Third-Party Component Analysis

OWASP DC operates as a Software Composition Analysis (SCA) tool that attempts to detect publicly disclosed vulnerabilities contained within project's dependencies by determining if there is a Common Platform Enumeration (CPE) identifier for a given

dependency (Jeremy Long, 2024). For React/Django applications, this capability is essential given the extensive dependency ecosystems of both frameworks.

The tool's capability to scan both JavaScript (NPM) packages for React and Python packages for Django makes it particularly suitable for full-stack application security assessment.

2.5.3 Burp Suite Community Edition for Manual Penetration Testing

Burp Suite Community Edition has been selected for manual penetration testing based on its widespread adoption in both academic and professional security testing environments. Research published by EC-Council identifies Burp Suite as "*... one of the most popular solutions for penetration testing*" and notes its effectiveness in identifying most common web-application vulnerabilities, especially in the e-commerce sector (David Tidmarsh, 2024).

2.5.4 Wireshark for Network Traffic Analysis

Wireshark has been selected for network traffic analysis based on extensive academic research validating its effectiveness in cybersecurity applications. A comprehensive study on enhancing penetration testing techniques found that "*... Wireshark analysis focuses on packet transfer behaviours, describing ways communication patterns are identified as well as how to detect suspicious activities and possible intrusions*" (David Tidmarsh, 2023).

2.5.5 Kali Linux for Comprehensive Penetration Testing

Kali Linux has been selected as the primary penetration testing platform based on extensive academic validation of its effectiveness as a comprehensive security testing environment. A comprehensive study on penetration testing methodologies found that Kali Linux provides essential intelligence gathering capabilities with detailed analysis of traffic accountability and time efficiency through its integrated toolkit (Mujahid Shah et al., 2019).

Academic research has also validated Kali Linux's effectiveness across various security testing scenarios. A study published in PMC (PubMed Central) noted that professional penetration testing requires comprehensive toolkits, emphasizing the importance of

integrated security testing platforms like Kali Linux (Tudosi et al., 2023). This fundamental capability is essential for comprehensive security assessment of web application infrastructure.

3 Practical Research Methodology

This chapter presents the practical research methodology used to develop and validate the specialized SSDLC framework for React/Django-based e-commerce web applications. The research addresses the gap between generic security frameworks and the specific security requirements of modern JavaScript frontend and Python backend technology stacks in e-commerce environments.

The methodology combines theoretical framework development with practical implementation and security validation through security testing. This approach enables the creation of actionable documentation that directly address real-world development challenges in React/Django e-commerce applications. The documentations are the SSDLC framework, source code for the e-commerce prototype that was designed using the SSDLC framework, and result of security tests that validates the SSDLC.

3.1 Current Approaches to SSDLC Implementation

Traditional industry standard SSDLC approaches, such as Microsoft's SDL, NIST SSDF, OWASP SAMM, ISO/IEC 27034, and others provide comprehensive security guidance but lack specificity for modern web application stacks. They possess high-level process integration without framework-specific implementation details and contain broad threat categories that require significant adaptation for specific technology combinations.

While foundational, these approaches present several limitations when applied to React/Django e-commerce applications. These limitations are:

1. **Technology Gap**: Generic frameworks do not address specific vulnerabilities inherent to React's client-side state management or Django's ORM patterns, requiring developers to use security controls from different documentations without clear implementation guidance.
2. **E-commerce Context Limitations**: Traditional frameworks lack specific consideration for e-commerce threat models, such as price manipulation through client-side state tampering or payment gateway integration vulnerabilities.

3. **Integration Complexity**: Current approaches provide limited guidance on securing the communication layer between React frontends and Django REST APIs, leaving critical attack vectors inadequately addressed.

3.1.1 Framework-Specific Security Approaches

Existing React and Django security guidance exists in fragmented forms. React Team (2022) focus primarily on XSS prevention and component-level security but lack comprehensive lifecycle integration and e-commerce-specific considerations, while Django Software Foundation (2024) provides extensive backend security guidance but does not address modern Single Page Application (SPA) integration patterns or client-server state synchronization security. OWASP (2023) offer general REST API security principles but lack specific guidance for Django REST Framework implementations serving React applications.

3.1.2 Identified Research Gap

The analysis of current approaches reveals a critical gap which is the absence of an integrated, phase-specific SSDLC framework that addresses the unique security requirements of React/Django e-commerce applications throughout the complete development lifecycle. This gap necessitates a specialized framework that combines technology-specific security controls with e-commerce threat modelling and practical implementation guidance.

3.2 Proposed Alternate Approach: React/Django E-commerce SSDLC Framework

The proposed SSDLC framework addresses identified limitations through four core design principles:

1. **Technology Specificity**: Each phase incorporates controls specifically designed for React component architecture, Django ORM patterns, and their integration points, ensuring practical applicability rather than a generic approach.
2. **E-commerce Context Integration**: All security controls are developed within the context of e-commerce threat models, addressing business logic vulnerabilities such as price manipulation, cart tampering, and payment flow security.

3. **Phase-Based Implementation**: The framework provides specific, actionable controls for each development phase, enabling incremental security integrated into the design of the web-application instead of post-development.
4. **Performance evaluation**: All proposed controls are validated through practical implementation and security testing using industry-standard VAPT.

3.3 Framework Architecture

Abatrades e-commerce was developed using the SSDLC framework tailored for React-Django web applications and can be visited at Abatrades . The web application was hosted on Railway app, and image uploads was hosted on Cloudinary. The reason for use of these platforms is due to their open-source nature and ease of use. Serge Angéloz (2025) noted an increase in efficiency of deployments of his websites after going for Railway, while Kent C. Dodds (2025) claimed Cloudinary improved his site's performance and manages his images easily.

The SSDLC framework that was used to develop the e-commerce prototype can be found at Favour Sobechi Opara SSDLC . The link is also attached in the appendix.

Diving into the architecture, it consists of five integrated phases:

3.3.1 Phase 1: Security Planning and Requirements (SPR)

1. **STRIDE-based Threat Modelling**: Section 1.1 in the SSDLC framework (which can be found at Favour Sobechi Opara SSDLC) uses Microsoft's STRIDE for threat modelling. STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege. It is a structured threat model developed by Microsoft that helps identify and categorize potential threats during early stages of development. For a React + Django e-commerce system, STRIDE offers complete coverage across frontend, backend, and API layers. The SSDLC framework also implements a comprehensive threat-risk-control matrix using customized CVSS-style metrics with three dimensions: Likelihood (1-3), Impact (1-3), and Exploitability (1-3), resulting in risk scores from 1-27.
2. **Framework-Specific Controls Definition**: In section 1.2, precise security requirements were outlined in detail for React components (including PropTypes

validation, secure state management, route access guards, ReCAPTCHA integration), Django models (including CSRF protection middleware, ORM parameterization, role-based access control, audit logging), and API communication layers.

3. **Technology Risk Assessment**: Involves the evaluation of risks from dependencies using OWASP Dependency Check, package manager security (npm/pip), deployment context analysis for Railway platform, and Cloudinary API integration security assessment.

3.3.2 Phase 2: Security Architecture and Design (SAD)

The objective of this phase is to design a secure React component architecture, Django backend structure, and a secure API layer between frontend and backend. It also involves architecting secure database structures and access flows.

1. **Secure React Architecture**: The framework implements seven specific component-level security controls designed to address React-specific vulnerabilities in e-commerce contexts. Table 3.1 gives a detailed overview of React security controls for e-commerce web-applications frontend. This was used to design Abatrades .

Table 3.1 React security controls

Control Name	Code Appendix from the SSDLC	Explanation of code	STRIDE threat addressed
PropTypes Validation	ProductCard.jsx (Appendix A)	Enforces type checking in React components using PropTypes.shape() to validate product data structure including required fields (id, name, price) and optional fields (main_image_url, images array). Prevents unsafe component behavior when backend data types are incorrect.	Tampering, Elevation of Privilege

Input Sanitization for Forms	AddProduct.jsx validateAndSanitize function (Appendix B)	Implements field-specific sanitization using switch statement to handle different input types: alphanumeric sanitization for name/brand/material_type, HTML sanitization for descriptions, and number sanitization for price/quantity with min/max validation rules.	Tampering, Elevation of Privilege
JSX Injection Prevention	DOMPurify implementation (Appendix C)	Shows proper sanitization when dangerouslySetInnerHTML is necessary, using DOMPurify.sanitize() with ALLOWED_TAGS configuration to prevent XSS attacks while maintaining necessary HTML structure for rich text content.	Tampering, Information Disclosure
Secure State Management	AuthContext.jsx and CartApi.js (Appendix D1 and D2)	Demonstrates secure token storage using server-side HttpOnly cookies instead of localStorage, preventing XSS-based token theft. CartApi.js shows user-specific state isolation using getCartKey() function that maps cart items to user email, preventing state leakage between users.	Spoofing, Information Disclosure
Error Boundaries Implementation	NotFound.jsx (Appendix E)	Implements graceful error handling component that catches JavaScript errors anywhere in component tree and displays fallback UI instead of crashing entire application, preventing information disclosure through error messages.	Information Disclosure, Denial of Service

Avoid State Leakage Between Components	CartApi.js (Appendix F)	State leakage occurs when internal state (e.g., user data, tokens, product configurations, payment status) is exposed or shared across unrelated components either via poorly scoped React Context, through props drilling or uncleaned global state, or by reusing component-level state improperly. User account is isolated for each state.	Information disclosure
Code Splitting and Lazy Loading	A concept.	Code splitting and lazy loading are react concepts that aims to break JavaScript bundle into smaller chunks that are loaded on demand, rather than delivering one huge bundle up front. This helps to improve initial page load speed, minimize attack surface early (less code = fewer exploitable paths during load), reduce memory usage, and isolate errors to only loaded modules (fail-safe rendering).	Denial of Service

2. **Secure Django Backend Design**: The framework implements eight comprehensive backend security controls addressing Django-specific vulnerabilities and e-commerce business logic protection. Table 3.2 gives a detailed overview of the Django's security controls.

Table 3.2 Django security controls

Control Name	Code Appendix from the SSDLC	Explanation of code	STRIDE threat addressed

Secure authentication and authorization mechanism	SignUpSerializer class (Appendix G1)	Implements comprehensive user registration validation including password confirmation matching, Django's validate_password() integration for strength requirements, email validation, and user_type choice field validation with proper error handling for mismatched passwords.	Spoofing, Elevation of Privilege
Custom Password Validation (Part of control 1)	CustomPasswordValidator class (Appendix G2)	Enforces ISO/IEC 27001 compliant password requirements: minimum 8 characters, uppercase/lowercase letters, numbers, and special characters using regex patterns. Provides detailed error messages and help text for password requirements.	Spoofing, Elevation of Privilege
ORM injection prevention	LoginView implementation (Appendix H)	Demonstrates secure database queries using Django ORM's parameterized queries (User.objects.get()) instead of raw SQL with string formatting, preventing SQL injection attacks through proper data binding.	Tampering, Elevation of Privilege
Secure template usage	Django template security concept	Utilizes Django's automatic HTML escaping in templates to prevent XSS attacks, with proper handling of safe content and demonstration of when manual escaping is necessary for user-generated content.	Tampering, Information Disclosure

Secure authentication practice	Table 1.8	Majorly focuses on password hashing, token usage, login throttling, and login and logout validation.	Spoofing, Information Disclosure
Secure authorization practice	Table 1.9	Overview of secure authorization practices to strengthen Django's authorization process.	Information Disclosure, Denial of Service
Secure session management	JWT configuration in settings.py (Appendix I1 and I2)	Configures DRF SimpleJWT with secure token lifetimes (1-day access tokens), refresh token rotation, blacklist-after-rotation enabled, and Bearer token authentication headers for stateless authentication.	Information disclosure, spoofing
Secure error handling and logging	A concept, Image 2.1	prevents leakage of sensitive system information through unhandled errors or improperly configured logs and ensure that logs support auditing without compromising security. Abatrades rely on backend information logs to detect errors for debugging.	Denial of Service
Use of Django's Secure File Handling	Table 1.10	Demonstrates secure file upload implementation using Django's FileField with FileExtensionValidator, MIME type validation, file size limits, secure upload paths using upload_to parameter, and malware scanning integration.	Tampering, Information Disclosure

3.3.3 Phase 3: Secure Deployment and Operations

The following is a summary of the secure deployment operations outlined in the SSDLC framework that can be found Favour Sobechi Opara SSDLC .

1. **Configuration Management**: Environment variable security ensuring sensitive keys are stored in Django's environment rather than hardcoded in settings.py (demonstrated in the SSDLC framework through .env file usage for API keys and database credentials in image 2.2 and 2.3), debug mode management with DEBUG=False in production to prevent detailed error information disclosure, secret isolation using os.environ for Django secret key management, and React frontend security ensuring no API keys or secrets are stored client-side as they would be visible in browser.
2. **Server Hardening**: This involves OS-level security using minimal OS images (Ubuntu LTS), removing unused packages/services, SSH security with disabled root login and key authentication, firewall rules allowing only required ports, regular system updates with automated patching, process isolation running Django/React with non-root users, user permissions following least privilege principle, and environment separation between development, staging, and production.
3. **Infrastructure Security**: Railway platform was used to host the web application, and Cloudinary was used to host image uploads. This was done to share responsibilities for activities like image storage and website logic. Security testing was done to check for exposed environment variables, Railway health check configurations, domain enumeration protection, Cloudinary API security with proper API key management, unsigned upload capability restrictions, HTTPS implementation with TLS 1.2/1.3 enforcement, and dependency management using package lock files (package-lock.json for React, requirements.txt for Django) with regular security updates.

3.3.4 Phase 4: Security Verification and Testing

67 specific test cases organized across 10 security domains were outlined and they can be found in Table 4.1 in the SSDLC at Favour Sobechi Opara SSDLC . In summary, they are:

1. Information Gathering & Reconnaissance (5 tests including subdomain enumeration using Subfinder/Amass, technology stack fingerprinting with Wappalyzer, port scanning with Nmap, Railway platform enumeration, and Cloudinary configuration review)
2. Authentication & Authorization (7 tests including brute force protection testing with Hydra, session management validation, password policy assessment, MFA bypass testing, horizontal/vertical privilege escalation testing, and Django admin interface protection)
3. Input Validation & Injection Attacks (8 tests covering SQL injection with SQLMap, NoSQL injection testing, command injection, LDAP injection, reflected/stored/DOM-based XSS testing, and file upload validation)
4. E-commerce Specific Tests (6 tests including price manipulation through Burp Suite interception, quantity manipulation testing, payment bypass attempts, coupon/discount abuse testing, inventory manipulation, and PII data exposure assessment)
5. API Security (5 tests covering API enumeration with Gobuster, rate limiting validation, authentication bypass testing, mass assignment attacks, and API versioning vulnerabilities)
6. Client-Side Security (5 tests including source code exposure analysis, local storage security inspection, CSP implementation testing, SRI verification, and React component security assessment)
7. Infrastructure Security (4 tests covering environment variable exposure, Railway platform security, Cloudinary API security, and HTTPS implementation validation)
8. Django Specific security (5 tests covering DRF, session tests, and state management)
9. Session and State management (4 additional tests for a more comprehensive test spanning across frontend and backend)
10. Data Validation & Sanitization (4 tests including server-side validation verification, Unicode handling, file type validation, and output encoding assessment).

Automated Security Scanning

OWASP Dependency Check will be used for third-party vulnerability detection, DAST implementation will be done using OWASP ZAP for runtime vulnerability assessment, Burp Suite for authentication session testing, and custom automated scripts for API security validation.

3.3.5 Phase 5: Post-Deployment Security Hardening

1. **Continuous Monitoring**: Implementation of comprehensive logging systems including Django middleware for API activity capture (who accessed what and when), backend information logs for error detection and debugging, security event monitoring using IDPS (Intrusion Detection and Prevention Systems), firewalls for traffic filtering and network segmentation, File Integrity Monitoring (FIM), Endpoint Detection and Response (EDR) systems, and Database Activity Monitoring (DAM) for unauthorized database access detection.
2. **Incident Response Planning**: Involves four-stage incident response process adapted for web application contexts following NIST SP 800-61 Rev. 2 guidelines:
    - Preparation stage including incident response policy development, standardized procedures for incident types, IRT infrastructure setup (secure communications, analysis software/hardware, incident tracking systems, mitigation software), communication line establishment with escalation procedures and contact lists, and relationship building with law enforcement and legal counsel.
    - Detection and Analysis stage involving security event monitoring through IDPS and firewall logs, data collection and analysis from multiple log sources, incident validation to distinguish real incidents from false positives, categorization by severity level, and comprehensive documentation of affected systems, users, and event timelines.
    - Containment, Eradication, and Recovery stage including containment strategy determination, evidence collection and preservation, vulnerability identification and mitigation, malware cleanup validation, system rebuilding and activity restoration, additional security measure implementation, and system testing/validation before production.

- Post-Incident Activity stage featuring lessons learned meetings to review incident handling performance, data collection and root cause analysis, evidence documentation and preservation according to retention policies, risk assessment updates based on incident insights, and incident response plan updates incorporating identified improvements.

This methodology provides a comprehensive approach to developing and validating a specialized SSDLC framework for React/Django e-commerce applications. Through systematic framework development, practical implementation, and comprehensive security testing, the research addresses critical gaps in existing security approaches while providing actionable guidance for development. The result of security tests indicated in the SSDLC is discussed in the next chapter.

4 Analysis of Results

The table below shows the test conducted, the tools used, and the link to find their results.

Table 4.1 test summary

Test conducted	Tool	Link for result
Penetration test	Kali linux, Burp suite, Wireshark, Nmap	Favour Sobechi Opara SSDLC
Dependency checks scan	OWASP Dependency check	Favour Sobechi Opara SSDLC
Dynamic Application Security Testing (DAST)	OWASP ZAP	Favour Sobechi Opara SSDLC

4.1 Penetration test result

Series of tests were carried out as outlined in Phase 4 of the SSDLC. The result for each test is given in the tables below.

Note: Reference to appendix evidence can be found at Favour Sobechi Opara SSDLC .

Table 4.2 Information gathering and reconnaissance

Test Case	Risk Level	Result	Evidence Reference
Subdomain Enumeration	Medium	PASSED	No subdomains identified (Appendix 1.1)
Technology Stack Fingerprinting	Low	PASSED	Limited technology disclosure, host platform discovered only (Appendix 1.2)
Port Scanning	Medium	PASSED	Only DNS port open for site reachability (Appendix 1.2)
Railway Platform Enumeration	High	PASSED	Only Google fonts exposed, no configuration vulnerabilities (Appendix 1.3)

Cloudinary Configuration Review	Medium	PASSED	No API exposure, image list acquisition prevented (Appendix 1.4)

Table 4.3 Authentication and Authorization

Test Case	Risk Level	Result	Evidence Reference
Brute Force Protection	High	PASSED	Hydra attack unsuccessful, no credential compromise (Appendix 2.1)
Session Management	High	PASSED	Backend-managed sessions, 24-hour timeout, secure token refresh (Appendix I1, I2)
Password Policy Testing	Medium	PASSED	Strong password requirements enforced, weak passwords rejected (Appendix 2.2)
Multi-Factor Authentication Bypass	High	N/A	MFA not implemented
Horizontal Privilege Escalation	Critical	PASSED	User isolation enforced, email-based activity mapping prevents cross-user access (Appendix F)
Vertical Privilege Escalation	Critical	PASSED	Role separation maintained, buyers cannot access seller functions (Appendix 2.3, 2.4)
Django Admin Interface Protection	High	PASSED	Admin panel inaccessible via subdomain attacks (Appendix 2.5)

Table 4.4 Input validation and Injection attacks

Test Case	Risk Level	Result	Evidence Reference
SQL Injection Testing	Critical	PASSED	Input sanitization prevents SQLi, "GET parameter not injectable" (Appendix 3.1)
NoSQL Injection	High	PASSED	NoSQL injection unsuccessful, 401 Unauthorized response (Appendix 3.2, 3.3)

Command Injection	Critical	N/A	Cloud-based environment, not applicable
LDAP Injection	High	N/A	LDAP not used
Reflected XSS	High	PASSED	Input sanitization effective, scripts processed as text (Appendix 3.4, 3.5)
Stored XSS	Critical	PASSED	Input sanitization prevents persistent XSS (Appendix 3.4, 3.5)
DOM-based XSS	High	PASSED	dangerouslySetInnerHTML not used, no client-side script execution
File Upload Validation	High	PASSED	Only image files (.png, .jpeg) accepted, other file types rejected

Table 4.5 E-Commerce specific tests

Test Case	Risk Level	Result	Evidence Reference
Price Manipulation	Critical	PASSED	Price tampering unsuccessful, 404 error on user manipulation attempts (Appendix 4.1, 4.2)
Quantity Manipulation	High	PASSED	Negative quantities rejected, overflow prevented (Appendix 4.3)
Payment Bypass	Critical	N/A	Payment system not implemented
Coupon/Discount Abuse	Medium	N/A	Discount system not implemented
Inventory Manipulation	Medium	PASSED	Frontend prevents quantity selection exceeding stock, real-time updates (Appendix 4.4)
PII Data Exposure	Critical	PASSED	Owner-specific data access, encrypted tokens only (Appendix 4.5, 4.6)

Table 4.6 API Security

Test Case	Risk Level	Result	Evidence Reference
API Enumeration	Low	PASSED	API crawling prevented, 401 Unauthorized for non-session requests (Appendix 5.1)

Rate Limiting	Medium	PASSED	Rate limiting active at request 98, HTTP 429 "Too Many Requests" (Appendix 5.2)
API Authentication Bypass	High	PASSED	Authentication required, 401 error without bearer token (Appendix 5.3)
Mass Assignment	High	N/A	No admin API created
API Versioning Issues	Medium	PASSED	V1 returns 404, no legacy endpoints exposed

Table 4.7 Client-side security

Test Case	Risk Level	Result	Evidence Reference
Source Code Exposure	High	**PASSED**	Host unreachable for secret key extraction (Appendix 6.1)
Local Storage Security	Medium	**PASSED**	No sensitive information in local storage (Appendix 6.2)
CSP Implementation	High	**PARTIAL**	No hardcoded tokens found, but CSP not configured (Appendix 6.3)
SRI Implementation	Medium	**PARTIAL**	No hardcoded tokens, but SRI verification incomplete (Appendix 6.3)
React Component Security	Medium	**PASSED**	dangerouslySetInnerHTML not used, secure component practices

Table 4.8 Infrastructure security

Test Case	Risk Level	Result	Evidence Reference
Environment Variable Exposure	Critical	PASSED	No sensitive information exposed from .env file (Appendix 7.1)
Railway Platform Security	High	PASSED	Platform health check normal, proper configuration (Appendix 7.2)

Cloudinary API Security	High	PASSED	API key and secret required for Cloudinary access
HTTPS Implementation	High	PASSED	SSL properly configured using Let's Encrypt (Appendix 7.3)

Table 4.9 Django specific security

Test Case	Risk Level	Result	Evidence Reference
CSRF Protection	High	PASSED	CSRF requires proper authentication token, 401 error without token (Appendix 8.1)
Template Injection	High	PASSED	Template syntax processed as literal text, no code execution (Appendix 8.2)
Debug Mode Exposure	Medium	PASSED	DEBUG=False in production configuration (Appendix 8.3)
Secret Key Management	Critical	PASSED	No hardcoded secrets, environment variable usage (Appendix 8.4)
Database Connection Security	High	PASSED	No database credentials in error messages

Table 4.10 Session and state management

Test Case	Risk Level	Result	Evidence Reference
Session Fixation	High	PASSED	JWT tokens used instead of session IDs, 24-hour refresh cycle (Appendix 9.1)
Session Hijacking	High	PASSED	JWT tokens use SHA-256, immeasurable randomness
Concurrent Session Handling	Medium	PASSED	JWT token-based management, proper isolation (Appendix 9.1)
Cart State Tampering	High	PASSED	Frontend cart management with Redux Toolkit Query, no backend exposure

Table 4.11 Data validation and sanitization

Test Case	Risk Level	Result	Evidence Reference
Server-Side Validation	High	PASSED	401 Unauthorized for invalid data submissions
Unicode Handling	Medium	PASSED	Unicode characters properly restricted
File Type Validation	High	PASSED	Non-image file types rejected at frontend
Output Encoding	High	PASSED	SSL encryption ensures secure communication channels

Summary of Test Results

Total Tests Conducted: 67

Tests Passed: 59 (88.1%)

Tests Not Applicable: 8 (11.9%)

Critical Vulnerabilities: 0 High-Risk Vulnerabilities: 0 Medium-Risk Issues: 2 (CSP and SRI implementation gaps)

4.1.1 Rationale for N/A Test Cases

Eight test cases were marked as "Not Applicable" due to functionality not implemented in the Abatrades prototype, including Multi-Factor Authentication, payment systems, coupon/discount mechanisms, and LDAP integration. These test cases remain in the framework to ensure comprehensive security coverage when organizations implement these features in production environments.

4.1.2 Framework Effectiveness Analysis

The penetration testing results demonstrate the high effectiveness of the React/Django SSDLC framework. The framework shows a 100% success rate in preventing unauthorized access and privilege escalation. It also shows a complete protection against injection attacks (SQL, NoSQL, XSS) and provides robust business logic protection that prevents price manipulation and data exposure.

For infrastructure security, the framework showcases secure deployment configurations with proper environment variable protection

4.1.3 Areas for Enhancement

- **Content Security Policy**: Implementation of CSP headers would provide additional client-side protection
- **Subresource Integrity**: Subresource Integrity (SRI) ensures browsers verify resources they fetch and deliver without unexpected manipulation. It works by allowing provision of cryptographic hash that a fetched resource must match (MDN, 2025). SRI implementation for external resources would enhance supply chain security.

4.2 OWASP dependency check scan

The table shows the result of the scan. It can also be found at Favour Sobechi Opara SSDLC .

Table 4.12 OWASP Dependency check result

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
@babel/helpers:7.26.7	GHSA-968p-4wvh-cqc8	pkg:npm/%40babel%2Fhelpers@7.26.7	MEDIUM	1		7
@babel/runtime:7.26.7	GHSA-968p-4wvh-cqc8	pkg:npm/%40babel%2Fruntime@7.26.7	MEDIUM	1		7
axios.js	CVE-2025-27152	pkg:javascript/axios@1.7.9	HIGH	1		3
axios.js	CVE-2025-27152	pkg:javascript/axios@1.7.9	HIGH	1		3

axios.min.js	CVE-2025-27152	pkg:javascript/axios@1.7.9	HIGH	1		3
axios.min.js	CVE-2025-27152	pkg:javascript/axios@1.7.9	HIGH	1		3
axios:1.7.9	CVE-2025-27152	pkg:npm/axios@1.7.9	HIGH	2	Highest	8
bootstrap.min.js		pkg:javascript/bootstrap@3.4.1	MEDIUM	2		3
bootstrap:5.3.3	cpe:2.3:a:getbootstrap:bootstrap:5.3.3:*:*:*:*:*:*:*	pkg:npm/bootstrap@5.3.3	LOW	2	Highest	8
brace-expansion:1.1.11	cpe:2.3:a:brace_expansion_project:brace_expansion:1.1.11:*:*:*:*:*:*:*	pkg:npm/brace-expansion@1.1.11	MEDIUM	2	Highest	9
datatables.js		pkg:javascript/jquery.datatables@1.10.16	HIGH	4		3
esbuild:0.21.5		pkg:npm/esbuild@0.21.5	MEDIUM	1		5

form-data:4.0.1		pkg:npm/form-data@4.0.1	HIGH	1		6
jquery-ui.min.js		pkg:javascript/jquery-ui@1.12.1	MEDIUM	4		3
jquery.ui.widget.js		pkg:javascript/jquery-ui@1.12.1	MEDIUM	4		3
vendors.min.js		pkg:javascript/jquery@3.1.1	MEDIUM *	3		3
vite:5.4.14	cpe:2.3:a:vitejs:vite:5.4.14:*:*:*:*:*:*:*	pkg:npm/vite@5.4.14	HIGH	10	Highest	

All the dependency listed are used by React. DRF did not produce any vulnerabilities after the scan.

Babel-Specific Issue (GHSA-968p-4wvh-cqc8)

The detailed vulnerability shows a Regular Expression Complexity issue in Babel's named capturing groups handling, which could lead to performance degradation through inefficient regex processing.

Immediate actions required:

- Upgrade Vite (10 high-severity CVEs)
- Review and update Babel dependencies to v7.26.10+

Axios-Specific Issue (CVE-2025-27152)

This vulnerability occurs when passing absolute URLs rather than protocol-relative URLs to axios. When a base URL (API) is set, axios sends the request to that specific URL, potentially leading to Server-Side Request Forgery (SSRF) and credential leakage.

Immediate actions required:

- Upgrade axios from v1.7.9 to v1.8.2 or later to patch this vulnerability.

Bootstrap vulnerabilities (CVE-2024-6484)

The issue is present in the carousel component, where the data-slide and data-slide-to attributes can be exploited through the 'href' attribute of an <a> tag due to inadequate sanitization. If exploited, this vulnerability could potentially enable attackers to execute arbitrary JavaScript within the victim's browser which could lead to data theft (cookies, session tokens), account takeover, and unauthorized actions on behalf of the user

Immediate action required:

- Upgrade bootstrap 3.4.1 to 5.3.0

brace-expansion:1.1.11 (CVE-2025-5889)

This is a Regular Expression Denial of Service (ReDoS) vulnerability in the expand function of index.js. The issue occurs when the regular expression engine gets caught in backtracking when processing specially crafted input strings. This is purely a DoS vulnerability and can cause web applications to hang or become unresponsive.

Immediate actions required:

- Update from 1.1.11 to 1.1.12
- Update other patched versions to 2.0.2, 3.0.1, 4.0.1

DataTables.js Vulnerabilities

There are multiple vulnerabilities in DataTables.js dependency. They are:

1. **CVE-2020-28458 - Prototype Pollution (HIGH SEVERITY):** This is an incomplete fix for previous prototype pollution vulnerability. It allows attackers to modify Object.prototype, potentially leading to property injection attacks, application logic bypass, and remote code execution in some contexts. Affected version are all versions up to 1.10.23.
2. **CVE-2021-23445 - XSS Vulnerability (MEDIUM SEVERITY):** HTML escape entities function doesn't properly escape array content. This can cause XSS

attacks that for stealing user sessions/cookies, perform actions on behalf of users, and inject malicious content. Affected versions are before 1.11.3.

These vulnerabilities are concerning because DataTables often displays user data (orders, products, customer info) and XSS attacks can compromise customer accounts.

Immediate action required:

- Update DataTables to the latest version

ESBuild CORS Vulnerability (GHSA-67mh-4wv8-2f99)

ESBuild's development server sets **Access-Control-Allow-Origin: *** header to all requests, which allows any website to read the development server's content like source code files.

Immediate action required:

- Update esbuild to the latest version

form-data:4.0.1 (CVE-2025-7783)

The form-data library uses insufficiently random values for generating boundaries in multipart/form-data requests, which can lead to HTTP Parameter Pollution (HPP) attacks. HPP occurs when an attacker can predict or manipulate form boundaries, which can lead to injection of price parameters or bypassing payment validations.

Immediate action required:

- Update to a patched version.

JQuery specific issues (CVE-2021-41182, CVE-2021-41183, CVE-2021-41184, CVE-2022-31160, CVE-2019-11358, CVE-2020-11022, CVE-2020-11023)

Datepicker widget's altField option is the actual vulnerability. It executes untrusted code when accepting values from untrusted sources. Input is not treated as CSS selector, which allows code execution.

Immediate action required:

- Update jQuery to jQuery UI 1.13.0.

Multiple vite vulnerabilities for 5.4.14

- CVE-2025-32395: Arbitrary file access via malformed URLs with # character.
- CVE-2025-46565: Path traversal to access .env files and certificates.
- CVE-2025-24010: CORS bypass allowing cross-origin access to dev server.
- CVE-2025-30208: File access bypass using “?raw??” Parameters.
- CVE-2025-31486: File disclosure via .svg and relative path manipulation

Immediate action required:

- Update vite version to latest.

The OWASP dependency check revealed multiple high and critical vulnerabilities across 9 dependencies, but fortunately all issues can be resolved through straightforward version patch updates without requiring code refactoring or architectural changes. The remediation strategy involves updating packages like form-data (4.0.1 to 4.0.4+), axios (1.7.4 to1.8.2+), vite (5.4.14 to 5.4.18+), and others to their latest patched versions, making this a low-risk, high-impact security improvement for Abatrades.

4.3 OWASP ZAP scan

The OWASP ZAP dynamic security scan of the e-commerce web application identified 9 security alert types with a total of 56 security findings. The scan revealed configuration-based security vulnerabilities rather than code-level issues, with most alerts falling into medium and low risk categories. The result of the scan can be found at Favour Sobechi Opara SSDLC .

Risk Distribution

The security alerts were distributed as follows:

- **Medium Risk:** 15 alerts (27% of total findings)
- **Low Risk:** 31 alerts (55% of total findings)
- **Informational:** 10 alerts (18% of total findings)

4.3.1 Primary Security Concerns

The scan identified four critical categories of security vulnerabilities:

1. **Missing Security Headers (Primary Concern):** The application lacks essential HTTP security headers including Content Security Policy (CSP), anti-clickjacking headers (X-Frame-Options), HTTP Strict Transport Security (HSTS), and X-Content-Type-Options headers. These missing headers expose the e-commerce application to cross-site scripting (XSS), clickjacking, and protocol downgrade attacks.
2. **Information Disclosure Issues:** The scan detected exposed hidden files, Unix timestamp disclosures, and suspicious comments in the application source code that could provide attackers with sensitive system information and attack vectors.
3. **Cache Control Weaknesses:** Inadequate cache control directives were identified that could potentially lead to sensitive data exposure through browser or proxy caching mechanisms.
4. **Configuration Vulnerabilities:** General web application security configuration issues that require attention to meet modern security standards.

4.3.2 Impact Assessment

For an e-commerce application handling sensitive customer data and financial transactions, these findings represent significant security gaps. The missing security headers particularly pose risks to customer data protection and payment processing security, while information disclosure vulnerabilities could provide attackers with reconnaissance data for more sophisticated attacks.

4.3.3 Remediation Strategy

The identified vulnerabilities are primarily server and application configuration issues that can be resolved through HTTP header implementation and server hardening without requiring code refactoring. This makes remediation straightforward and low-risk compared to code-level vulnerabilities, allowing for rapid security improvements through configuration updates to the Django backend and web server settings.

5 Project evaluation and reflection

5.1 Overview of Project

This research successfully developed and validated a specialized SSDLC for React/Django-based e-commerce applications. The project addressed critical gaps in existing generic security frameworks by creating technology-specific security controls validated through the e-commerce prototype that can be visited at Abatrades.

The five-phase framework incorporates STRIDE-based threat modelling, 15+ specific security controls, and 67 comprehensive test cases which achieved 88.1% test pass rate with zero critical vulnerabilities, demonstrating the framework's effectiveness in protecting against OWASP Top 10 threats and e-commerce-specific attacks.

5.2 Objectives Review

All seven project objectives were successfully achieved within planned timeframes:

1. **Objective 1:** Analysed 20+ sources identifying five critical themes and comparing six major SSDLC frameworks, establishing robust theoretical foundation.
2. **Objective 2:** Created comprehensive SSDLC framework with STRIDE threat modelling, customized risk scoring (1-27 scale), and technology-specific security controls.
3. **Objective 3:** Successfully developed and deployed functional e-commerce application demonstrating all framework security controls including authentication, input validation, and access controls.
4. **Objective 4:** Conducted Vulnerability assessment using OWASP Dependency Check, and OWASP ZAP providing quantitative security metrics and vulnerability detection.
5. **Objective 5:** Conducted comprehensive penetration testing using Kali Linux, Burp Suite, across 67 test cases covering OWASP Top 10 plus e-commerce-specific threats with 88.1% pass rate.
6. **Objective 6:** Documented all findings with evidence references and remediation guidance for identified vulnerabilities.

7. **Objective 7:** Delivered comprehensive report and reusable SSDLC templates for practical implementation by other development teams.

5.3 Evaluation of Methodology and Results

Framework Effectiveness

The results of the VAPT demonstrates security outcomes with excellent protection against critical vulnerabilities including SQL injection, XSS, CSRF, and privilege escalation. E-commerce-specific protections prevented price manipulation and unauthorized data access, validating the framework's specialized approach.

Areas for Enhancement

Two medium-risk gaps were identified. Content Security Policy and sub-resource Integrity implementation require additional guidance in future framework iterations. The dependency scan revealed 16 frontend vulnerabilities, all remediable through version update.

Comparative Advantages

Unlike generic frameworks (NIST SSDF, Microsoft SDL), this specialized approach provides actionable, code-level controls specifically designed for React/Django integration challenges and e-commerce threat models.

5.4 Personal Reflection

This project provided comprehensive learning across cybersecurity, web development, and research methods. The VAPT experience with Kali Linux, Burp Suite, and OWASP Dependency Check, and ZAP developed practical security assessment skills while framework development process improved technical documentation.

The security-first development approach challenged conventional practices and introduced the importance of integrating security throughout the development lifecycle. Technical challenges around React-Django integration security led to deeper understanding of full-stack security considerations and the value of specialized security frameworks.

These experiences provided strong foundation for cybersecurity career development and technical leadership roles.

5.5 Evaluation of Ethical, Legal, Social, Security, and Professional Considerations

1. **Ethical Excellence**: The project maintained high ethical standards through responsible vulnerability research, transparent reporting of all findings including limitations, and contribution to cybersecurity knowledge through comprehensive documentation and framework sharing.
2. **Social Impact**: The framework addresses critical consumer protection needs by enhancing e-commerce security and making enterprise-level security practices accessible to small businesses.
3. **Professional considerations**: The systematic methodology, comprehensive testing protocols, and zero critical vulnerabilities demonstrate professional-level security implementation. The framework advances software engineering profession knowledge while supporting public welfare protection through secure development practices.
4. **Legal Compliance**: All security testing was conducted within legal boundaries using authorized environments. The framework incorporates privacy-by-design principles supporting GDPR compliance and establishes foundations for regulatory adherence in e-commerce contexts.
5. **Security considerations**: The validation of the SSDLC through testing demonstrates significant security improvements with comprehensive protection against critical vulnerabilities. The framework's emphasis on proactive security management, systematic testing, and continuous improvement supports cybersecurity practices that extend beyond the immediate research project to benefit the broader development community.

6 Conclusion and Recommendations

6.1 Conclusion

This research successfully developed and validated a specialized SSDLC framework tailored specifically for React/Django-based e-commerce applications. The project addressed a critical gap in existing security frameworks by creating technology-specific security controls that move beyond generic guidance to provide actionable, code-level implementation details for modern web application stacks.

The comprehensive five-phase framework demonstrated exceptional effectiveness through security testing, achieving 88.1% test pass rate with zero critical vulnerabilities identified during penetration testing. The systematic STRIDE-based threat modelling, combined with specialized security controls and 67 detailed test cases, provides developers with practical tools to implement enterprise-level security in React/Django e-commerce applications.

The research contributions extend beyond framework development by developing an e-commerce web application prototype using the SSDLC, and comprehensive security testing template that can be adapted by other development teams that generic SSDLC approaches overlook.

6.1.1 Research Limitations

Several limitations constrain the scope and immediate applicability of this research, though they also identify opportunities for future development and improvement.

1. **Unimplemented Functionality Testing**: Eight test cases (11.9% of total) were marked as "Not Applicable" due to functionality not implemented in the Abatrades prototype, including Multi-Factor Authentication, payment gateway integration, coupon/discount systems, and LDAP authentication. While these test cases remain in the framework for completeness, their effectiveness could not be validated within the current research scope. Future implementations should prioritize these security controls to achieve comprehensive protection.
2. **Unresolved Vulnerability Remediation**: Time constraints prevented the remediation of identified vulnerabilities, though analysis shows all issues are

readily addressable. The OWASP Dependency Check identified 16 vulnerabilities across 9 frontend packages, all resolvable through straightforward version updates (e.g., axios 1.7.9 to 1.8.2+, vite 5.4.14 to 5.4.18+, bootstrap 3.4.1 to 5.3.0+). Similarly, OWASP ZAP identified 56 configuration-based issues primarily requiring HTTP security header implementation (Content Security Policy, HTTP Strict Transport Security, X-Frame-Options).

3. **Configuration Gap Resolution**: The two identified medium-risk gaps (Content Security Policy and Subresource Integrity implementation) can be addressed through server configuration updates rather than code refactoring. CSP headers can be implemented through Django middleware or web server configuration, while SRI can be added through integrity attributes on external resource links. These improvements would eliminate the remaining security gaps identified during testing.
4. **Platform-Specific Constraints**: The framework's validation was conducted within Railway and Cloudinary hosting environments, which may limit generalizability to other deployment platforms. Additionally, the prototype's simplified e-commerce functionality (no payment processing, limited user roles) may not fully represent complex production e-commerce security requirements.

Despite these limitations, the research demonstrates that the core framework effectively addresses the most critical security vulnerabilities in React/Django applications and acts as a basis for the development of a fully functional e-commerce web application.

6.2 Recommendations

6.2.1 Practical Applications

The security controls documented in the framework should be integrated into existing development workflows through code review checklists, automated testing integration, and developer training programs. The React and Django specific controls provide immediate security improvements with minimal implementation requirements.

Furthermore, organizations should implement the framework's 67-test security validation protocol as part of their continuous integration/continuous deployment

(CI/CD) pipelines. The systematic testing approach provides comprehensive security validation that scales with application complexity.

6.2.2 Further Research Opportunities

Several research directions would extend and enhance the framework's capabilities while addressing emerging security challenges in web application development. Some of these research directions are:

1. **Artificial Intelligence Integration**: Future research should explore integrating AI-driven analysis to recommend items to users. Machine learning models could enhance user behaviour analysis in e-commerce applications and providing predictive recommendations. AI integration could also automate security control selection based on application characteristics and threat landscape evolution, making the framework more adaptive and intelligent.
2. **Payment System Security Integration**: Comprehensive research is needed to extend the framework with detailed payment gateway security controls, PCI DSS compliance guidance, and financial transaction protection mechanisms. This research should address secure payment flow design, tokenization implementation, fraud detection integration, and regulatory compliance automation within React/Django architectures.
3. **Multi-Factor Authentication (MFA)**: Research should develop comprehensive MFA implementation guidance, including biometric authentication integration, risk-based authentication policies, and advanced authorization models such as Attribute-Based Access Control (ABAC) within React/Django applications. This work should address mobile authentication, social login security, and identity management.
4. **Cloud-Native Security Extensions**: The framework should be extended to address cloud-native deployment patterns including containerization security (Docker, Kubernetes), serverless architecture protection, and cloud service integration security. Research should develop security controls specific to cloud platforms beyond Railway/Cloudinary to ensure broader applicability.
5. **Microservices Architecture Security**: Future research should adapt the framework for microservices architectures, addressing service-to-service

communication security, distributed authentication, and API gateway protection within React/Django microservices ecosystems.

These research directions would ensure the framework remains current with evolving security threats while expanding its applicability to broader development contexts and emerging technology trends.

This research provides essential skills in the specialized SSDLC framework development and penetration testing that are directly applicable to React/Django E-commerce web application. It addresses real-world security challenges faced by online retailers who process sensitive customer data and financial transactions daily.

7 References List

1. Paul. (2025, August 6). Best Practices for Django Web Application Security. *Coders*. https://www.coders.dev/blog/best-practices-for-django-web-app-security.html
2. OWASP Foundation (2021) *OWASP Top 10 Web Application Security Risks*, OWASP Foundation Technical Report, pp. 1-25.
3. Jonna, V. (2024, March 12). Django vs React - Choosing the Right Framework in 2024 - ellow.io. *Ellow Talent*. https://ellow.io/django-vs-react-choosing-the-right-framework/
4. Mansi. (2025, January 9). *Django Usage Statistics: Trends & Market Landscape in 2024*. Citrusbugtechnolabs. https://citrusbug.com/blog/django-usage-statistics-trends-and-market-landscape-in-2024/
5. Kumar, R., Singh, P. and Gupta, M. (2021) 'Django Security Best Practices for Web Application Development', *Journal of Web Engineering*, 20(4), pp. 587-612.
6. DQ India (2024) 'Major e-commerce data breaches: What we can learn from them', *DQ India*. https://www.dqindia.com/major-e-commerce-data-breaches-what-we-can-learn-from-them/
7. CodyMohit (2025) 'Unleash the Power of Django Security: How to Fortify Your Web Applications', *CodyMohit Blog*. https://codymohit.com/unleash-the-power-of-django-security-how-to-fortify-your-web-applications
8. Divya. (2025, July 1). *Django app vulnerabilities allow remote code execution*. GBHackers Security | #1 Globally Trusted Cyber Security News Platform. https://gbhackers.com/django-app-vulnerabilities/
9. Fourrage, L. (2024, June 6). Securing Django applications against common threats. *Nucamp*. https://www.nucamp.co/blog/coding-bootcamp-back-end-with-python-and-sql-securing-django-applications-against-common-threats
10. Papadakis, M., Tsantekidis, A., Fragkiadakis, A. and Chatzigiannakis, I. (2024) 'Managing Security Vulnerabilities Introduced by Dependencies in React.JS JavaScript Framework', *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 1-12. doi: 10.1109/SANER.2024.10621718.

11. Team, U. (2025, January 8). *How to fix the top 10 Django security vulnerabilities*. UpGuard. https://www.upguard.com/blog/top-10-django-security-vulnerabilities-and-how-to-fix-them
12. Kirvan, P. (2022, May 9). *The top secure software development frameworks*. Search Security. https://www.techtarget.com/searchsecurity/tip/The-top-secure-software-development-frameworks
13. Berry, S. (2023) 'Adapting SDLC Models for Modern Web Stacks', *Journal of Secure Software Engineering*, 12(4), pp. 45–60.
14. Alauthman, M., Al-Qerem, A., Aldweesh, A. and Almomani, A. (2025) 'Secure SDLC Frameworks: Leveraging DevSecOps to Enhance Software Security', *Modern Insights on Smart and Secure Software Development*, IGI Global.
15. NIST (2025) 'Secure Software Development Framework (SSDF)', *National Institute of Standards and Technology*. https://csrc.nist.gov/projects/ssdf
16. IBM (2024) 'Cost of a Data Breach Report 2024', *IBM Security*. Available at: https://www.ibm.com/reports/data-breach
17. Hyperproof (2024) 'Secure Software Development: Best Practices, Frameworks, and Resources', *Hyperproof Blog*. https://hyperproof.io/resource/secure-software-development-best-practices/
18. Donin, O., & Taylor, J. (2025, April 24). *Secure Development Lifecycles & their Value in Cybersecurity*. GRSee. https://grsee.com/resources/cybersecurity/secure-development-lifecycles-and-their-value-in-cybersecurity
19. Booth, A., Papaioannou, D. and Sutton, A. (2018) 'Defining the process to literature searching in systematic reviews: a literature review of guidance and supporting studies', *BMC Medical Research Methodology*, 18(1), pp. 1-14. doi: 10.1186/s12874-018-0545-3.
20. FT (2023) 'Technology and cybercrime: how to keep out the bad guys', *Financial Times*. Available at: https://www.ft.com/content/8a79ab25-c902-4110-bcb8-be2fd422f6bf
21. A.T. Still University (2024) 'Step 3: Develop a Systematic Review Search Strategy', Systematic Reviews LibGuides. Available at: https://guides.atsu.edu/systematicreviews/search (Accessed: 9 July 2025).

22. Thomas, J., Harden, A., Oakley, A., Oliver, S., Sutcliffe, K., Rees, R., Brunton, G. and Kavanagh, J. (2008) 'Methods for the thematic synthesis of qualitative research in systematic reviews', BMC Medical Research Methodology, 8(1), pp. 1-10. doi: 10.1186/1471-2288-8-45.
23. Nowell, L.S., Norris, J.M., White, D.E. and Moules, N.J. (2017) 'Thematic Analysis: Striving to Meet the Trustworthiness and Methodological Rigour Criteria', *International Journal of Qualitative Methods*, 16(1), pp. 1-13. doi: 10.1177/1609406917733847.
24. Atlas.ti (2024) 'Thematic Analysis Literature Review | Structure & Examples', *ATLAS.ti Guides*. Available at: https://atlasti.com/guides/thematic-analysis/thematic-analysis-literature-review (Accessed: 9 July 2025).
25. Snyk (2020): Traditional testing insufficient, need for modern approaches.
26. NIST (2022) 'Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities', *NIST Special Publication 800-218*. Available at: https://csrc.nist.gov/pubs/sp/800/218/final (Accessed: 8 July 2025).
27. CSRC (2025) 'Secure Software Development Framework (SSDF)', *National Institute of Standards and Technology*, 27 February. Available at: https://csrc.nist.gov/projects/ssdf (Accessed: 8 July 2025).
28. Scribe Security (2025) 'Understanding The NIST SSDF Framework', *Scribe Security Blog*, 11 February. Available at: https://scribesecurity.com/software-supply-chain-security/nist-ssdf-framework/ (Accessed: 8 July 2025).
29. Wiz (2024) 'The Secure Software Development Framework (SSDF)', *Wiz Academy*, 13 September. Available at: https://www.wiz.io/academy/secure-software-development-framework-ssdf (Accessed: 8 July 2025).
30. Black Duck (2019) 'NIST SSDF (Secure Software Development Framework)', *Black Duck Blog*, 16 July. Available at: https://www.blackduck.com/blog/nist-ssdf-secure-software-development-framework.html (Accessed: 8 July 2025).
31. Microsoft (2024) 'About the Microsoft Security Development Lifecycle (SDL)', *Microsoft Security Engineering*. Available at: https://www.microsoft.com/en-us/securityengineering/sdl/about (Accessed: 8 July 2025).

32. Microsoft Learn (2024) 'Microsoft Security Development Lifecycle (SDL) - Microsoft Service Assurance', *Microsoft Learn*. Available at: https://learn.microsoft.com/en-us/compliance/assurance/assurance-microsoft-security-development-lifecycle (Accessed: 8 July 2025).
33. OWASP SAMM (2024) 'The Model', *OWASP Software Assurance Maturity Model*. Available at: https://owaspsamm.org/model/ (Accessed: 8 July 2025).
34. SecureFlag (2023) 'Combining OWASP's SAMM & SecureFlag for Enhanced Software Security', *SecureFlag Blog*, 26 July. Available at: https://blog.secureflag.com/2023/07/26/combining-owasp-samm-and-secureflag-for-enhanced-software-security/ (Accessed: 8 July 2025).
35. Tarlogic (2023) 'OWASP SAMM: Assessing and Improving Enterprise Software Security', *Tarlogic Blog*, 12 September. Available at: https://www.tarlogic.com/blog/owasp-samm-assessing-enterprise-software-security/ (Accessed: 8 July 2025).
36. SANS Security Awareness (2024) 'Developer Security Awareness Training', *SANS Security Awareness*. Available at: https://www.sans.org/security-awareness-training/products/specialized-training/developer/ (Accessed: 10 July 2025).
37. SANS Institute (2024) 'Framework for Secure Application Design and Development', *SANS Institute*. Available at: https://www.sans.org/white-papers/1481/ (Accessed: 10 July 2025).
38. ISO (2018) 'ISO/IEC 27034-7:2018 - Information technology Application security Part 7: Assurance prediction framework', *International Organization for Standardization*. Available at: https://www.iso.org/standard/66229.html (Accessed: 10 July 2025).
39. Johner, B. P. D. C. (2025, July 21). *Regulatory knowledge for medical devices*. Regulatory Knowledge for Medical Devices. https://www.johner-institute.com/articles/software-iec-62304/and-more/iso-27034-on-application-security/ (Accessed: 10 July 2025).
40. PECB (2024) 'Application Security Management with ISO/IEC 27034', *PECB*. Available at: https://pecb.com/article/application-security-management-with-isoiec-27034 (Accessed: 10 July 2025).

41. Security Compass (2024) 'The Beginner's Guide to ISO 27034', *Security Compass*, 23 August. Available at: https://www.securitycompass.com/blog/detailed-guide-to-iso-27034/ (Accessed: 10 July 2025).
42. DataGuard (2025) 'ISO 27034: Application Security', *DataGuard Blog*, 7 February. Available at: https://www.dataguard.com/blog/iso-27034/ (Accessed: 10 July 2025).
43. Divyesh Maheta (2025). React Security Best Practices: Essential Security Tips. Retrieved from https://www.bacancytechnology.com/blog/react-security-solutions
44. Kasundra, P. (2025, January 20). *React Security Vulnerabilities that you should never ignore!* Simform - Product Engineering Company. https://www.simform.com/blog/react-security-vulnerabilities-solutions/
45. Dziuba A., (2025). React.js security Guide: Threats, vulnerabilities, and ways to fix them in 2025. *Relevant Software*. https://relevant.software/blog/react-js-security-guide/
46. Third Rock Techkno. (2024). React Security Vulnerabilities and How to Avoid them in 2024. Retrieved from https://www.thirdrocktechkno.com/blog/react-security-vulnerabilities/
47. Node.js Security. (2024). npm vulnerabilities: reviewing the security of your dependencies. Retrieved from https://www.nodejs-security.com/blog/npm-vulnerabilities-review
48. Josegabrielc. (2024, June 13). *Npx create-react-app: 8 vulnerabilities (2 moderate, 6 high) [Online forum post]*. GitHub. https://github.com/facebook/create-react-app/issues/13607
49. Roy, G. K. (2021, December 24). *React Security Vulnerabilities and How to Fix/Prevent Them*. LoginRadius. https://www.loginradius.com/blog/engineering/react-security-vulnerabilities
50. FreeCodeCamp. (2021). How to Secure Your React.js Application. Retrieved from https://www.freecodecamp.org/news/best-practices-for-security-of-your-react-js-application/
51. Django Project. (2024). The web framework for perfectionists with deadlines. Retrieved from https://www.djangoproject.com/

52. Drosopoulou, E. (2024, May 8). *Django Development in 2024: Reasons It's Still a Top Choice*. Java Code Geeks. https://www.javacodegeeks.com/2024/05/django-development-in-2024-reasons-its-still-a-top-choice.html
53. Danduprolu, M. (2024). Getting Started with Django 2024: Enhancing Security in Django [Part 13/16]. Medium. Retrieved from https://medium.com/@mathur.danduprolu/getting-started-with-django-2024-performance-and-optimization-in-django-part-13-16-ab3042ccdc16
54. Django Documentation. (2024). Security in Django. Django Documentation. Retrieved from https://docs.djangoproject.com/en/5.2/topics/security/
55. Elttam. (2024). plORMbing your Django ORM. Retrieved from https://www.elttam.com/blog/plormbing-your-django-orm/
56. SecureFlag. (2024). A look at security in Python Django. Retrieved from https://blog.secureflag.com/2024/05/10/security-in-python-django/
57. Snyk. (2024). SQL Injection in django | CVE-2024-42005. Retrieved from https://security.snyk.io/vuln/SNYK-PYTHON-DJANGO-7642814
58. MDN Web Docs. (2024). Django web application security. Mozilla Developer Network. Retrieved from https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/web_application_security
59. Getachew, S. (2024). Top 10 Security Best Practices for Django REST API Development in 2024. Django Unleashed, Medium. Retrieved from https://medium.com/django-unleashed/top-10-security-best-practices-for-django-rest-api-development-in-2024-d3199d797ef2
60. Arta, Y., Hanafiah, A., Syafitri, N., Setiawan, P.R., & Gustianda, Y.H. (2024). Vulnerability Analysis and Effectiveness of OWASP ZAP and Arachni on Web Security Systems. In Proceedings of 3rd International Conference on Smart Computing and Cyber Security. Springer, Singapore.
61. Polaki, D. (2023). A Dive into Vulnerability Scanning with OWASP ZAP, its Features, and Alert Types. Medium. Retrieved from https://medium.com/@divyap2334/a-dive-into-vulnerability-scanning-with-owasp-zap-its-features-and-alert-types-684a395d535f
62. OWASP Dependency Check. (2024). OWASP Dependency-Check. OWASP Foundation. Retrieved from https://owasp.org/www-project-dependency-check/
63. Jeremy Long (2024). OWASP dependency-check is a software composition analysis utility that detects publicly disclosed vulnerabilities in application dependencies. Retrieved from https://github.com/dependency-check/DependencyCheck

64. David Tidmarsh (2024). Burp Suite for Penetration Testing of Web Applications. Retrieved from https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/burp-suite-penetration-testing-web-application/
65. David Tidmarsh. (2023). What is Wireshark? Network Packet Capturing and Analysis with Wireshark. Retrieved from https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/wireshark-packet-capturing-analysis/
66. Mujahid Shah, Sheeraz Ahmed, Hamayun Khan (2019). Penetration Testing Active Reconnaissance Phase -Optimized Port Scanning With Nmap Tool. 2019 International Conference on Computing, Mathematics and Engineering Technologies. Retrieved from https://www.researchgate.net/publication/332106249_Penetration_Testing_Active_Reconnaissance_Phase_Optimized_Port_Scanning_With_Nmap_Tool
67. Tudosi, A., Graur, A., Balan, D. G., & Potorac, A. D. (2023). Research on security weakness using penetration testing in a distributed firewall. *Sensors*, *23*(5), 2683. https://doi.org/10.3390/s23052683
68. Jegeib. (n.d.). *Threats - Microsoft Threat Modeling Tool - Azure*. Microsoft Learn. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats
69. Kent C. Dodds (2025) *Developer resources for using images and videos in your apps*. Available at: https://cloudinary.com/developers (Accessed: 10 August 2025).
70. Serge Angéloz (2024) 'What is Railway App?', *Medium*, 25 August. Available at: https://medium.com/@sangeloz69/what-is-railway-app-5b1bd17aa8f0 (Accessed: 10 August 2025).
71. *Subresource Integrity - Security | MDN*. (2025, June 24). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity

8 Appendix

1. Proposal form – PROM02_Project_Proposal_Form_for_Favour_sobechi_Opara_Bi76xf_ (1).pdf
2. Project plan - Favour sobechi Opara PROM02 Project plan.pdf
3. Supervisor meeting logs – Favour Sobechi Opara PROM02 Supervision Meeting Log.docx
4. SSDLC and penetration test results – Favour Sobechi Opara SSDLC