Exercise Work

# COMP.SEC.300-2024-2025-1 Secure Programming

Vimal Wilson 153081141

## Table of Contents

## Abstract

As we all know that digital transactions are getting very important day by day, a website that might help us to keep track of all the digital financial activities would be pretty good. A website that will analyse our spending habits from credit cards, bank statements and tell us where we can come up with some savings would be very good. As it involves financial data their security is very important. This paper explores how we can keep the Expense Management Website safe on the basis of OWASP Top 10. by looking at issues like unauthorized access, weak encryption etc. In the early stages of design process, we are hoping to come up with a security plan that will keep the website safe from most of the security flaws. Through this paper we are showing how building security from the ground up with taking care of the OWASP Top 10 can ensure user data safety and industry standards..

## Introduction

In today's world due to the widespread of internet and the useability of online tools, more and more people are ok with using them to manage their money, keeping track of daily expenses as well as ways to maximize the credit card rewards. I believe that this new free expense management website will be a user friendly, feature rich one where people can store and organize financial information such as their bank statements and credit card details and better manage their finances. But the safety of financial records is very important as the loss of this can cause so much damage to an individual. This is where we are gonna apply the principles of secure programming.

In this Exercise work we are gonna deal with the question called *"How can we use secure programming ideas, based on the OWASP Top 10, to protect a new expense management website from cyber threats?"*. The OWASP Top 10 can be said as a checklist of the most common ways a hacker tries to break the website so that they can steal passwords, sneak in harmful codes and even accessing data that they shouldn't. By studying these checklist, we can design our website in a better manner to avoid these situations. for example, we can set up the website in such a way that only the right people can view their own financial records and their data is locked with strong encryption that even the admin can't decrypt. This paper will look at the potential dangers, share simple examples and also show the practical steps which can keep the website safe. The goal is to create a website that will help the users to better manage their money and gives them peace of mind knowing the information is protected. This paper will help developers to build a secure website from start to finish ensuring it's ready to handle the user data in the most secure way.

## Research Question

*How can we use secure programming ideas, based on the OWASP Top 10, to protect a new expense management website from cyber threats?*

## Background

### 4.1 Overview of Secure Programming

Secure programming can be said as designing and writing of the code that resists against the attack of hackers. It involves lots of practices such as input validation to prevent harmful data entries, encrypting data to protect confidentiality, proper session management to maintain user authenticity and proper error handling to avoid exposing sensitive data. For the proper safety of a website, these are very important as this will resist against threats like phishing, malware and other attacks. It's common knowledge now that secure programming is quite important in protecting sensitive data, and if we are talking about financial point of view its importance increases again.

### 4.2 OWASP Top 10

The OWASP Top 10, updated in 2021, is a globally recognized standard identifying the most critical security risks to web applications.

1. **Broken Access Control**: Unauthorized access to resources or functions.
2. **Cryptographic Failures**: Weak encryption exposing sensitive data.
3. **Injection**: Malicious code execution via untrusted input.
4. **Insecure Design**: Architectural flaws introducing vulnerabilities.
5. **Security Misconfiguration**: Improper setup increasing attack surfaces.
6. **Vulnerable and Outdated Components**: Risks from unpatched software.
7. **Identification and Authentication Failures**: Weak authentication mechanisms.
8. **Software and Data Integrity Failures**: Unverified code or data alterations.
9. **Security Logging and Monitoring Failures**: Inadequate attack detection.
10. **Server-Side Request Forgery (SSRF)**: Server manipulation for unauthorized requests.

### 4.3 Expense Management Website Context

An expense management website enables users to log, track, and manage expenses through a web browser. Typical features include expense categorization, report generation, receipt uploads, and data synchronization across devices via cloud services. Given its handling of financial data such as credit card numbers, transaction histories, and personal details the website could be a  prime target for cybercriminals.

## Methods

This exercise work combines the information regarding the OWASP Top 10 from the OWASP foundations website with the principles of secure programming for an Expense Management Website. We looked at each of the ones in the OWASP Top 10 and tried to come up with solutions for each of them with respect to the Website. ChatGPT-4o was used to come up with practical situations as well as some of the solutions for overcoming them. for example, we looked how a broken access control could affect in one's profile and how we can overcome it. Most of the AI provided ones had some errors or were not very particularly good, sometimes we had to refine it to suit our website. Canva was utilized to come up with a homepage for the website as a starting point to the website if I ever plan to make this into a fully working one.

## Analysis

In this part we are going to look into the Top 10 OWASP vulnerabilities that could affect the Expense Management Website, where people can track their spending habits. we will look into each of them with examples and solutions that could keep the website safe.

### 6.1 Broken Access Control

This Vulnerability occurs when the website doesn’t properly check who’s allowed to see or do certain things. It is just like keeping the room open and then anyone can look into what’s going on in there. This could be a problem for a website which handles things related to money.

Website Example:

Someone changes the web address from like ***/user/123/expenses*** to ***/user/456/expenses*** and sees their private info because the website didn't lock the accounts in a proper manner. This could be a problem as someone else can easily change the entered details of someone else which in some cases can totally break the family budget and can cause even legal trouble in the future (if tax filing is done based on this)

Solutions:

- Set up clear rules about who can do what ( ie what an admin can do and what the normal user can)
- The website should always check its own list to confirm whether the user is allowed to do something instead of just trusting your browser

- it should generate a super-secure login key that stops working fast and gets tossed out when you log off

**6.2 Cryptographic Failures**

This happens when the website doesn’t lock up private info (like transaction details and even credit card numbers) in a good manner, so the attackers can look at it or break in.

Website Examples:

Sending confidential information such as credit card numbers over an unsafe connection (like ***HTTP***, not ***HTTPS***). Keeping passwords as plain words can make it easy for hackers to break them. Using old encryptions like ***128-bit DES*** could totally make the website less secure. stolen info like credit card numbers could cause huge financial losses.

Solutions:

- Always use a secure connection (***HTTPS***) and enforce it.
- Lock financial documents like receipts, card numbers with a strong safe (***AES-256***). Check encryption schemes often to make sure they’re not old or broken, and switch to better ones if needed**.**

**6.3 Injection**

Injection vulnerabilities occur when untrusted user input is not handled properly and passed to a backend interpreter—such as a database engine, operating system, or browser—allowing attackers to inject malicious commands that are executed as part of the system’s logic.

Website Examples:

SQL Injection via Login Form: An attacker inputs '***OR 1=1; --*** 'into a username field. If the backend constructs a query like ***SELECT * FROM users WHERE username = '[input]' AND password = '[hash]',*** it becomes ***SELECT * FROM users WHERE username = '' OR 1=1; --' AND password = '[hash]'***. The ***1=1*** condition is always true, and **--** comments out the password check, granting access without valid credentials

Solutions:

- Input Validation with Allowlisting: Restrict inputs to expected formats using strict allowlists (e.g., usernames limited to ***[a-zA-Z0-9]*** via regex ***^[a-zA-Z0-9]+$)***. Reject or strip any characters outside this set to block malicious payloads like scripts or SQL operators.

- Parameterized Queries: Use prepared statements or parameterized queries (e.g., ***SELECT * FROM expenses WHERE user_id =? AND category =?***) to separate user input from executable code.

### 6.4 Insecure Design

Insecure design refers to fundamental flaws in a website's architecture or planning that fail to incorporate security controls from the start, making it vulnerable to exploitation. These are systemic issues—such as missing authentication mechanisms, poor data isolation, that cannot be easily patched later without significant rework. For an expense management website, insecure design compromises the ability to protect sensitive financial data, which could totally make the data very insecure.

Website Examples:

Lack of Multi-Factor Authentication (MFA): If the website relies solely on username-password pairs without requiring a second factor (e.g., a one-time code), a stolen credential (e.g., via phishing) grants immediate access. For instance, an attacker with a user's password could log in via ***/login*** and access all expense records without additional verification. No Rate Limiting on Authentication Endpoints: Allowing unlimited login attempts on /login enables brute-force attacks. An attacker could script thousands of passwords guesses per minute (e.g., using tools like ***Hydra***), eventually cracking weak credentials without triggering any defenses.

Solutions:

Multi-Factor Authentication (MFA): Require a second authentication factor, such as a time-based one-time password (TOTP) via apps like Google Authenticator or SMS-based codes. Implement this using standards like ***RFC 6238***, ensuring that even if a password is compromised, an attacker needs a dynamic code (e.g., refreshed every 30 seconds) to access ***/user/dashboard***.

Rate Limiting and CAPTCHA: Enforce rate limits on sensitive endpoints (e.g., 5 login attempts per minute per IP). Add CAPTCHA (e.g., ***Google reCAPTCHA***) after repeated failures to block automated scripts, reducing brute-force success rates by requiring human interaction.

### 6.5 Security Misconfiguration

Security misconfiguration occurs when a website or its underlying infrastructure (e.g., web server, application server, or framework) is improperly configured, exposing vulnerabilities that attackers can exploit. This stems from failing to apply secure settings, leaving default configurations unchanged, enabling unnecessary features, or mishandling error reporting.

For an expense management website, such misconfigurations create exploitable gaps in the system akin to leaving unsecured entry points or exposing internal system details potentially compromising sensitive financial data and operational integrity.

Website Examples:

Directory Listing Enabled: If the web server (e.g., ***Apache***) has directory indexing turned on for /uploads/, a request to ***https://example.com/uploads/*** displays a clickable list of files, such as ***receipt_123.pdf***, exposing private user receipts. An attacker could harvest these files without authentication.

Verbose Error Messages: A misconfigured application returns detailed stack traces, such as ***Database connection failed: Access denied for user 'admin' at host 'db.example.com'***, when a query fails. This reveals database credentials, server names, or software versions (e.g., ***MySQL 5.7***), giving attackers reconnaissance data for targeted exploits.

Solutions:

- Disable Directory Listing: Configure the web server to prevent indexing by setting ***Options -Indexes*** in Apache's ***.htaccess*** or ***httpd.conf***, or equivalent in Nginx (***autoindex off***). This ensures requests to ***/uploads/*** return a ***403 Forbidden*** response instead of a file list, protecting stored assets like receipts.
- Custom Error Handling: Configure the application to suppress detailed error output in production (e.g., PHP's ***display_errors = Off and log_errors = On***). Return generic messages like "***Something went wrong, please try again***" to users, while logging full stack traces to a secure file (e.g.***, /var/log/app_errors.log*** with ***600*** permissions) inaccessible via the web root.

**6.6 Vulnerable and Outdated Components**

Vulnerable and outdated components refer to the use of software libraries, frameworks, plugins, or server software within a website that have known security flaws (e.g., ***CVE***s) and are no longer supported or patched.

For an expense management website, relying on such components is akin to using obsolete security hardware with well-known bypass methods, exposing sensitive financial data to exploitation due to the lack of timely updates or maintenance.

Solutions:

- Component Inventory Management: Maintain a comprehensive inventory of all software components (e.g., libraries, frameworks, server software) using a Software

Bill of Materials (SBOM). Track versions (e.g., ***jQuery 3.6.0, Apache 2.4.58***) and update schedules.

- Dependency Scanning Tools: Use automated tools like ***OWASP Dependency-Check*** or ***Snyk*** to scan the website's codebase and dependencies (e.g., ***package.json, pom.xml***) for known vulnerabilities. Integrate these into the CI/CD pipeline (e.g***., GitHub Actions***) to flag issues like ***jQuery 1.9.1*** before deployment, providing actionable reports with remediation steps.

**6.7 Identification and Authentication Failures**

Identification and authentication failures occur when a website's mechanisms for verifying user identities are weak or improperly implemented, allowing attackers to impersonate legitimate users. This vulnerability arises from inadequate password policies, poor session management, or the absence of multi-factor authentication (MFA), compromising the integrity of the authentication process. For an expense management website, such failures expose sensitive financial accounts to unauthorized access, as the system fails to robustly distinguish between genuine users and malicious actors exploiting stolen credentials or session data.

<u>Website Examples:</u>

Weak Password Policies: Permitting simple passwords like ***password123*** on ***/login*** reduces the computational effort needed for brute-force attacks.

Persistent Session Tokens: If session cookies (e.g., ***session_id=abc123***) lack expiration or aren't invalidated on logout, an attacker who steals a cookie via XSS (e.g., ***<script>document.cookie</script>***) can reuse it indefinitely on ***/dashboard***, bypassing re-authentication.

<u>Solutions:</u>

- Enforce Strong Password Policies: Require passwords with a minimum length of 12 characters, including uppercase, lowercase, numbers, and special characters (e.g., enforced via regex ***^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{12,}$***). Use server-side validation on ***/register*** and ***/change_password*** to reject weak inputs

- Account Lockout and Notification: Implement lockout mechanisms after 10 failed login attempts within 5 minutes, tracked via a rate-limiting store. On lockout, trigger an email or SMS alert to the user with details like timestamp and IP, and require CAPTCHA or MFA to unlock, thwarting automated attacks.

### 6.8 Software and Data Integrity Failures

Software and data integrity failures occur when a website fails to verify the authenticity and unaltered state of its code, dependencies, or data, allowing attackers to introduce malicious modifications. This vulnerability often stems from insecure software supply chains, unverified third-party integrations, or unprotected update processes, compromising the trustworthiness of the system. For an expense management website, integrity failures could enable attackers to alter financial data, inject malicious code, or deploy tampered software, undermining the reliability of expense tracking and reporting functions.

Website Examples:

Insecure CI/CD Pipeline: A misconfigured Continuous Integration/Continuous Deployment (CI/CD) system (e.g., ***Jenkins*** with weak credentials) allows an attacker to push a malicious update to ***/api/expenses***, embedding a backdoor (e.g., ***eval($_GET['cmd']) in PHP***) that grants remote code execution on the production server.

Unverified Data Import: An expense import feature accepting CSV uploads (***/import***) processes a file with a crafted field (e.g., ***amount=999; DROP TABLE users; --***), which, if executed as a query, corrupts the database instead of just storing the data.

Solutions:

- Secure CI/CD Pipelines: Lock down CI/CD systems (e.g., ***GitHub Actions, Jenkins***) with strong authentication (e.g., MFA), role-based access control (RBAC), and immutable audit logs.
- Trusted Dependencies with Scanning: Limit third-party dependencies to reputable sources (e.g., ***npm, PyPI***) and scan them with tools like ***Snyk*** (e.g., ***snyk test***) or ***OWASP Dependency-Check*** during builds.

### 6.9 Security Logging and Monitoring Failures

Security logging and monitoring failures occur when a website does not adequately record security-relevant events or actively monitor them, leaving it blind to malicious activities. This vulnerability arises from insufficient log collection, lack of real-time analysis, or unprotected log storage, preventing timely detection, response, and forensic investigation of attacks. For an expense management website, such failures mean missing critical indicators of compromise like unauthorized logins or data tampering, allowing threats to escalate undetected and undermining the ability to protect financial data or comply with audit requirements.

Website Examples:

No Logging of Failed Login Attempts: The ***/login*** endpoint lacks logging for failed authentication attempts (e.g., incorrect passwords). An attacker running a brute-force script (e.g., ***100 guesses per minute via hydra -l user -P wordlist.txt example.com***) goes unnoticed, eventually cracking a weak password like ***user123*** without triggering alerts.

Insecure Log Storage: Logs stored in plaintext at ***/var/log/app.log*** with weak permissions (e.g., ***666***) allow an attacker who gains server access (e.g., via a prior exploit) to edit or delete entries (e.g., ***echo "" > app.log***).

Solutions:

- Comprehensive Event Logging: Capture all security-relevant events—successful and failed logins, permission changes, data modifications, and errors—across endpoints like ***/login, /expenses***, and ***/admin***. Include contextual data (e.g., timestamp, user_id, source_ip, event_type) in a structured format (e.g., ***JSON: {"time": "2025-04-10T12:00:00Z", "user_id": 123, "ip": "192.168.1.1", "event": "login_failed"}***), written to a secure log file (e.g., ***/var/log/expense_app.log*** with ***640*** permissions).
- Tamper-Proof Log Storage: Encrypt logs at rest using tools like logrotate with ***AES-256*** (e.g., ***logrotate -e /var/log/*.log***) and store backups off-site (e.g., ***AWS S3*** with server-side encryption). Restrict access with file permissions (e.g., ***chmod 640, chown root:adm***) and forward logs to a Write-Once-Read-Many (WORM) system (e.g., a remote ***syslog*** server), ensuring attackers can't alter them even with server access.

### 6.10 Server-Side Request Forgery (SSRF)

Server-Side Request Forgery (SSRF) is a vulnerability where attackers manipulate a website's server into making unintended HTTP requests to internal or external resources, exploiting the server's trust and network position. This occurs when user-supplied input (e.g., URLs) is passed to server-side functions (e.g., ***curl, fetch***) without proper validation, allowing attackers to access restricted systems, retrieve sensitive data, or trigger malicious actions.

Website Examples:

Internal Admin Access via Import Feature: An expense import tool at ***/import*** accepts a user-provided URL (e.g., ***http://localhost:8080/admin***). If the server fetches this using ***curl http://localhost:8080/admin***, it retrieves sensitive admin panel data (e.g., user lists, API keys) from an internal service inaccessible to external clients, exposing it to the attacker.

Solutions:

- URL Validation with Allowlisting: Restrict server-side requests to a predefined list of trusted domains using an allowlist (e.g., ***["https://api.trusted.com", "https://receipts.example.com"]***). Implement this in code on ***/import***, rejecting inputs like ***localhost*** or ***attacker.com*** outright.

This website stores bank statements, credit card statements, card information, track the rewards for each card, give suggestions for expenses etc.

## Security Testing

Security testing is vital to validate the website's defenses against exploitation. This section expands on four manual testing methods, detailing their application and relevance to the OWASP Top 10 vulnerabilities.

### Exploratory Security Testing

Testers manually go through the website without plans and looking for weaknesses by replicating user actions and analyzing the responses that they are getting. this will help us to see if there are an access problems such as broken access control and all. for example, attempting to access /admin without credentials to check for unrestricted endpoints.

### Penetration Testing

In this testing we are simulating real-world attacks to look at the effect of resilience and looking to exploit vulnerabilities like injection and cryptographic weakness. This is very important as this will help us to detect injection flaws by submitting malicious SQL/XSS payloads and to check if there is any cryptographic failures by intercepting traffic for plaintext.

### Input Validation Testing

in this we are assessing the input handling by giving unexpected or malicious data to see if it produces vulnerabilities. If this is successful it confirms that there is protection against injection and confirms secure coding practices for the user inputs.

## Results

The Analysis section in the paper gives us a detailed look into how we can secure the expense management website, according to the benefits of secure programming principles.

### 8.1 Effectiveness of Secure Programming Principles

- **Input Validation:** Prevents injection and SSRF by rejecting malicious inputs, as demonstrated by blocking SQL payloads in login forms.
- **Encryption:** Protects data with AES-256 and TLS, resisting cryptographic failures and ensuring confidentiality during browser-server communication.
- **Access Control:** RBAC and server-side checks reduce unauthorized access risks, validated by testing URL manipulation scenarios.
- **Component Management:** Regular updates eliminate vulnerabilities in libraries, confirmed by patch application simulations.
- **Monitoring:** Real-time logging detects attacks like brute-force attempts, enhancing incident response capabilities.

### 8.2 Challenges and Limitations

- **Usability vs. Security:** MFA strengthens authentication but may frustrate users with additional steps, requiring careful UX design.
- **Emerging Threats:** Zero-day exploits and advanced persistent threats (APTs) demand continuous updates beyond OWASP's scope, challenging static defenses.
- **Resource Constraints:** Small teams may lack time or expertise for comprehensive audits, necessitating automated tools or outsourcing.

### 8.3 Practical Implications

Implementing these principles protects financial data from breaches and attacks, it will help us to build user trust by demonstrating we are committed towards the user data safety and also reduces legal and reputational risks. For example, preventing injection attacks avoids costly data leaks.

## Conclusion

This report shows that Secure programming aligned with OWASP Top 10 is pretty important for a safe and reliable website. Doing things like input validation, encryption and monitoring critical vulnerabilities could be very beneficial for the working of a safe and reliable website. If we can make a website after taking into consideration all the OWASP risks, we can be certain that it will be somewhat a safe and secure website. key takeaways can be said as we need a secure development cycle, good threat modeling and regular audit as well as updates in the current scenarios.

We can say that secure programming isn't just a technical necessity, it will help the users to have more trust, which is very important when we are dealing with a website that deals with financial data. even though there are certain minor troubles like a little less usability with the addition of MFA and all, but using these principles will always outweigh the bad. Developers can use these principles to create a secure, reliable and attack resistant Expense management website which will safely guard the sensitive data in current world.

## Reference

- OWASP Foundation. (2021). *OWASP Top Ten*. https://owasp.org/www-project-top-ten/

## Acknowledgement

### AI Usage

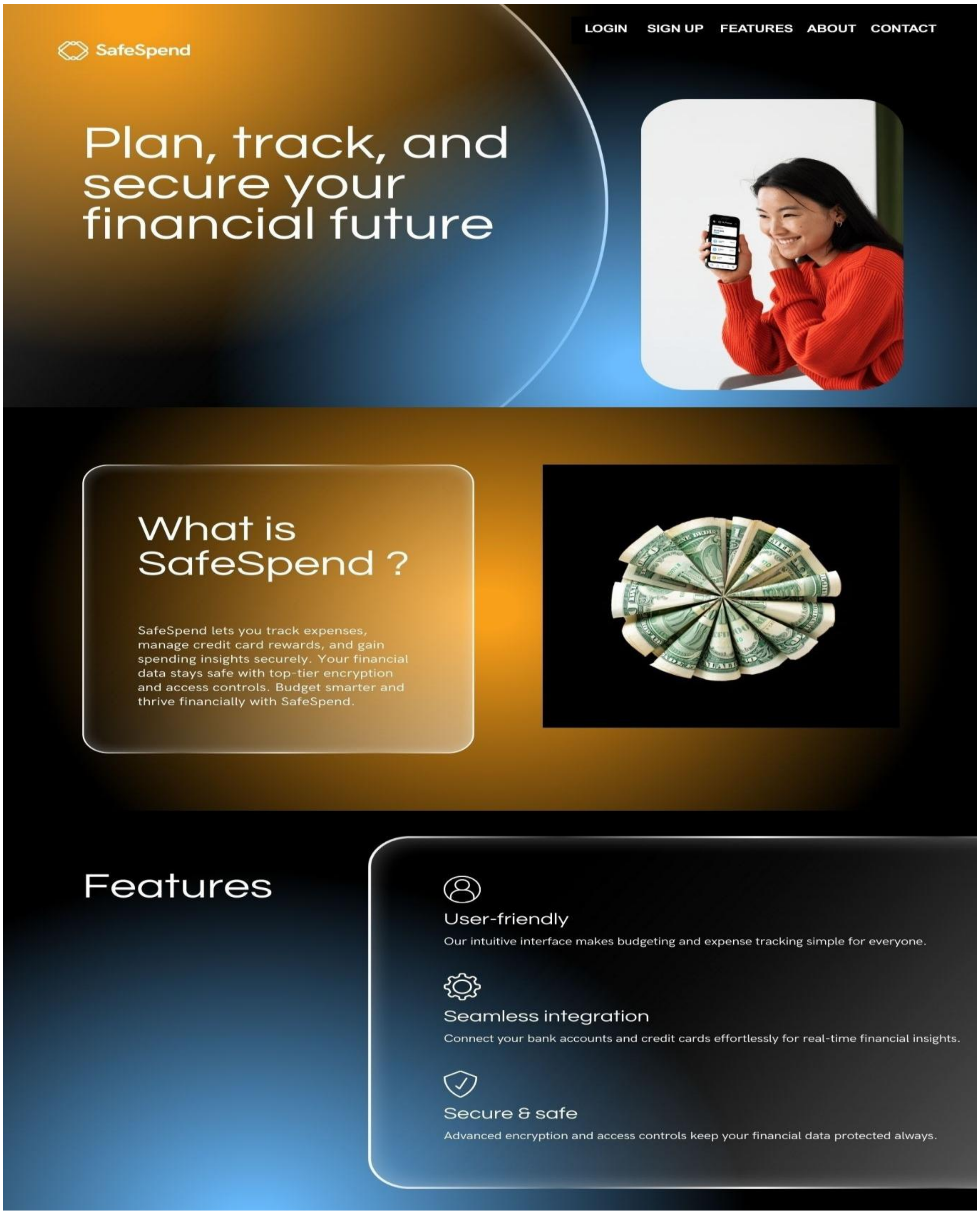
I would like to acknowledge the use of ChatGPT-4o for the theoretical analysis of OWASP Top 10 in this work. ChatGPT-4o was used in the 'Analysis' part for examples of link address as well in the 'Results' section in the subheading 'Effectiveness of Secure Programming Principles' and 'Challenges and Limitations' in generating simple sentences to convey the findings.

### Others

I have used the Canva website version for designing the Home page for the Expense Management Website.

## Appendix

1. Expense Management Website Homepage



Plan, track, and secure your financial future

What is SafeSpend ?

SafeSpend lets you track expenses, manage credit card rewards, and gain spending insights securely. Your financial data stays safe with top-tier encryption and access controls. Budget smarter and thrive financially with SafeSpend.

Features

User-friendly

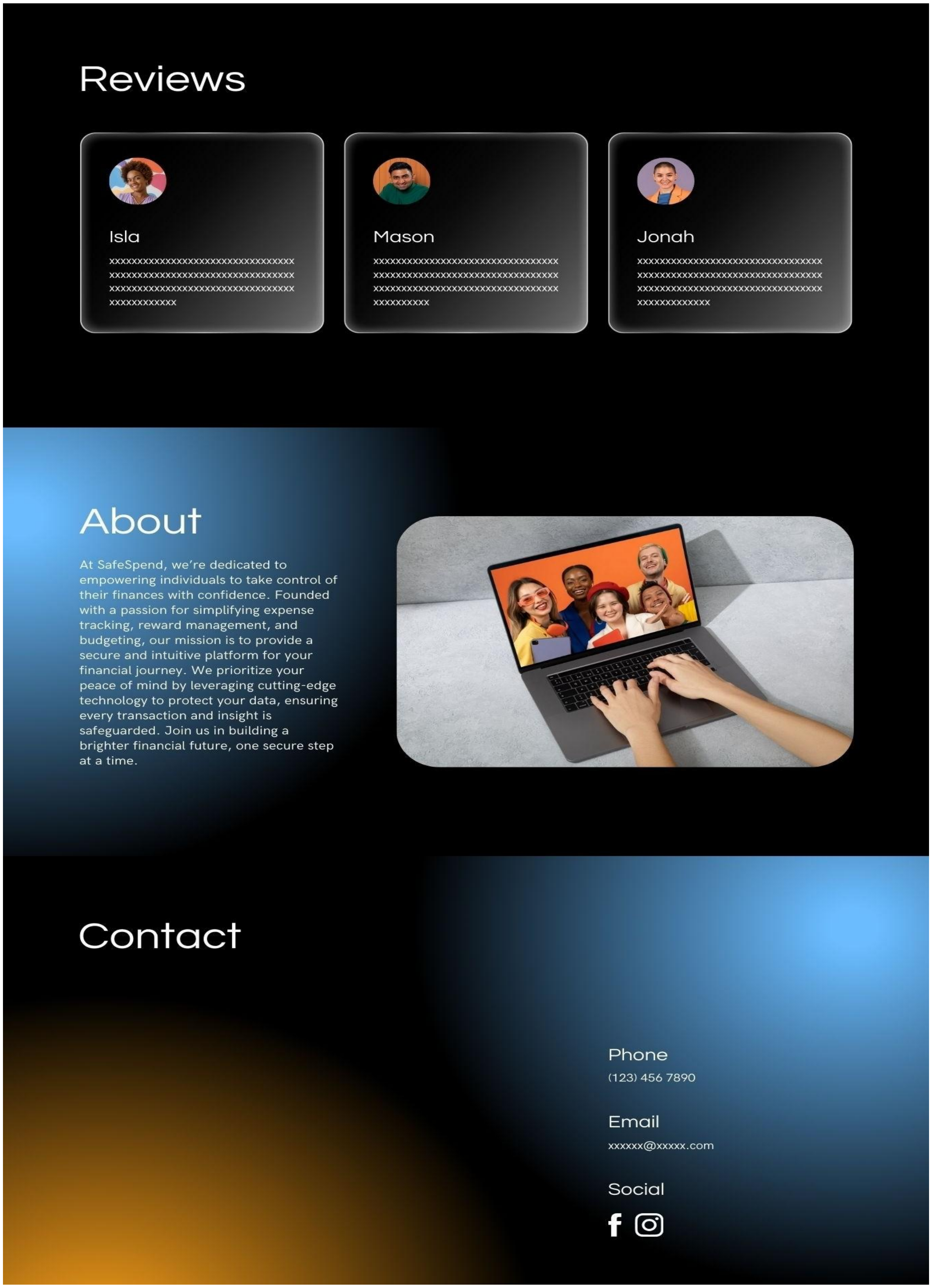
Our intuitive interface makes budgeting and expense tracking simple for everyone.

Seamless integration

Connect your bank accounts and credit cards effortlessly for real-time financial insights.

Secure & safe

Advanced encryption and access controls keep your financial data protected always.



## Reviews

Isla

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxx

Mason

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxx

Jonah

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxx

## About

At SafeSpend, we're dedicated to empowering individuals to take control of their finances with confidence. Founded with a passion for simplifying expense tracking, reward management, and budgeting, our mission is to provide a secure and intuitive platform for your financial journey. We prioritize your peace of mind by leveraging cutting-edge technology to protect your data, ensuring every transaction and insight is safeguarded. Join us in building a brighter financial future, one secure step at a time.

## Contact

Phone

(123) 456 7890

Email

xxxxxx@xxxxx.com

Social